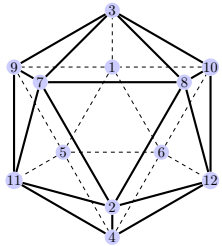
Metaheuristicas 9 - LNS & ALNS

Alexandre Checoli Choueiri

13/02/2025



Problemática

Já vimos que as vizinhanças são extremamente importantes para o desenvolvimento de meta-heurísticas de solução única. Mais ainda, **quanto maior a vizinhança, maior a chance de encontrarmos o ótimo global**.

Problemática

Já vimos que as vizinhanças são extremamente importantes para o desenvolvimento de meta-heurísticas de solução única. Mais ainda, **quanto maior a vizinhança, maior a chance de encontrarmos o ótimo global**.

No entanto, realizar a busca completa por vizinhanças muito grandes pode se **tornar inviável** computacionalmente

Problemática

Já vimos que as vizinhanças são extremamente importantes para o desenvolvimento de meta-heurísticas de solução única. Mais ainda, **quanto maior a vizinhança, maior a chance de encontrarmos o ótimo global**.

No entanto, realizar a busca completa por vizinhanças muito grandes pode se **tornar inviável** computacionalmente

Vamos então entender mais sobre o conceito de **tamanho de vizinhança**.

Tamanho de vizinhança

Relembrando a definição de vizinhança, visto na aula "**5 - Algoritmos de solução única: Conceitos**":

Definição

Vizinhança (Neighborhood): Uma função vizinhança N é um mapeamento $N : S \rightarrow 2^S$ que atribui a cada solução $s \in S$ um conjunto de soluções $N(s) \subset S$

Em que S é o conjunto de soluções factíveis para o problema.

Tamanho de vizinhança

Relembrando a definição de vizinhança, visto na aula "**5 - Algoritmos de solução única: Conceitos**":

Definição

Vizinhança (Neighborhood): Uma função vizinhança N é um mapeamento $N : S \rightarrow 2^S$ que atribui a cada solução $s \in S$ um conjunto de soluções $N(s) \subset S$

Em que S é o conjunto de soluções factíveis para o problema.
Sabemos que as soluções de um problema estão ligadas a uma instância em particular (I), chamemos então as soluções factíveis de de uma determinada instância como $S(I)$.

Tamanho de vizinhança

Podemos definir então o **tamanho** de uma vizinhança $N(\cdot)$ para uma instância I como:

Definição

Tamanho de vizinhança: $max\{|N(s)| : s \in S(I)\}$

Tamanho de vizinhança

Podemos definir então o **tamanho** de uma vizinhança $N(\cdot)$ para uma instância I como:

Definição

Tamanho de vizinhança: $max\{|N(s)| : s \in S(I)\}$

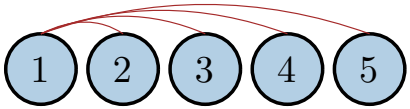
Ou seja, considerando uma instancia de um problema, o tamanho da vizinhança é dado pelo **maior número de vizinhos que uma solução pode gerar**.

Tamanho de vizinhança

Vejamos um exemplo para o problema do TSP com a vizinhança SWAP e 5 cidades:

Tamanho de vizinhança

Vejamos um exemplo para o problema do TSP com a vizinhança SWAP e 5 cidades:



Podemos fazer o SWAP do ponto um com todos os outros, gerando um total de **4** vizinhos.

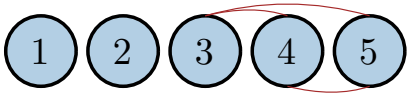
Tamanho de vizinhança

Vejamos um exemplo para o problema do TSP com a vizinhança SWAP e 5 cidades:

Podemos fazer o SWAP do ponto um com todos os outros, gerando um total de **4** vizinhos.
Da mesma forma, com o ponto 2, gerando mais **3** vizinhos.

Tamanho de vizinhança

Vejamos um exemplo para o problema do TSP com a vizinhança SWAP e 5 cidades:

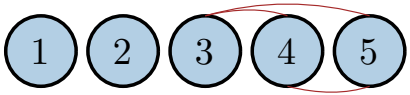


Podemos fazer o SWAP do ponto um com todos os outros, gerando um total de **4** vizinhos.

Da mesma forma, com o ponto 2, gerando mais **3** vizinhos. E com o 3 e 4, gerando mais **3** vizinhos.

Tamanho de vizinhança

Vejamos um exemplo para o problema do TSP com a vizinhança SWAP e 5 cidades:



Podemos fazer o SWAP do ponto um com todos os outros, gerando um total de **4** vizinhos. Da mesma forma, com o ponto 2, gerando mais **3** vizinhos. E com o 3 e 4, gerando mais **3** vizinhos.

Temos então um total de $4 + 3 + 3 = 10$ vizinhos. Como todas as soluções factíveis do TSP vão gerar esse mesma quantidade de vizinhos, então o tamanho da vizinhança (para esta instancia) é 10. A fórmula para determinar o tamanho da vizinhança para um problema de 5 pontos é então: $\frac{5(5-1)}{2} = 10$

Tamanho de vizinhança

Vamos definir agora o conjunto (possivelmente infinito) de todas as instâncias de um problema de tamanho n como $\mathcal{F}(n)$. É possível então definir o **tamanho de uma vizinhança como uma função $f(n)$ do tamanho da instância**:

Tamanho de vizinhança

Vamos definir agora o conjunto (possivelmente infinito) de todas as instâncias de um problema de tamanho n como $\mathcal{F}(n)$. É possível então definir o **tamanho de uma vizinhança como uma função $f(n)$ do tamanho da instância**:

Definição

Função tamanho de vizinhança: $f(n) = max\{|N(s)| : I \in \mathcal{F}(n), s \in S(I)\}$

Tamanho de vizinhança

Vamos definir agora o conjunto (possivelmente infinito) de todas as instâncias de um problema de tamanho n como $\mathcal{F}(n)$. É possível então definir o **tamanho de uma vizinhança como uma função $f(n)$ do tamanho da instância**:

Definição

Função tamanho de vizinhança: $f(n) = max\{|N(s)| : I \in \mathcal{F}(n), s \in S(I)\}$

Podemos então definir o tamanho da vizinhança do SWAP em função do número de pontos, como:

$$f(n) = \frac{n(n-1)}{2} \tag{1}$$

Tamanho de vizinhança

Agora, para determinar a ordem de crescimento de uma função f, usamos a notação "Big-O". De forma simplificada, essa notação reduz a função ao seu termo de crescimento mais acelerado. Por exemplo, no caso do SWAP, temos:

Tamanho de vizinhança

Agora, para determinar a ordem de crescimento de uma função f, usamos a notação "Big-O". De forma simplificada, essa notação reduz a função ao seu termo de crescimento mais acelerado. Por exemplo, no caso do SWAP, temos:

$$f(n) = \frac{n(n-1)}{2}$$

Tamanho de vizinhança

Agora, para determinar a ordem de crescimento de uma função f, usamos a notação "Big-O". De forma simplificada, essa notação reduz a função ao seu termo de crescimento mais acelerado. Por exemplo, no caso do SWAP, temos:

$$f(n) = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

Tamanho de vizinhança

Agora, para determinar a ordem de crescimento de uma função f, usamos a notação "Big-O". De forma simplificada, essa notação reduz a função ao seu termo de crescimento mais acelerado. Por exemplo, no caso do SWAP, temos:

$$f(n) = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Tamanho de vizinhança

Agora, para determinar a ordem de crescimento de uma função f, usamos a notação "Big-O". De forma simplificada, essa notação reduz a função ao seu termo de crescimento mais acelerado. Por exemplo, no caso do SWAP, temos:

$$f(n) = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = \frac{n^2}{2} - \frac{n}{2}$$

O termo de maior crescimento é $\frac{n^2}{2}$, portanto dizemos que a ordem de crescimento de $f(n)$ é:

$$f(n) = O(n^2)$$

Tamanho de vizinhança

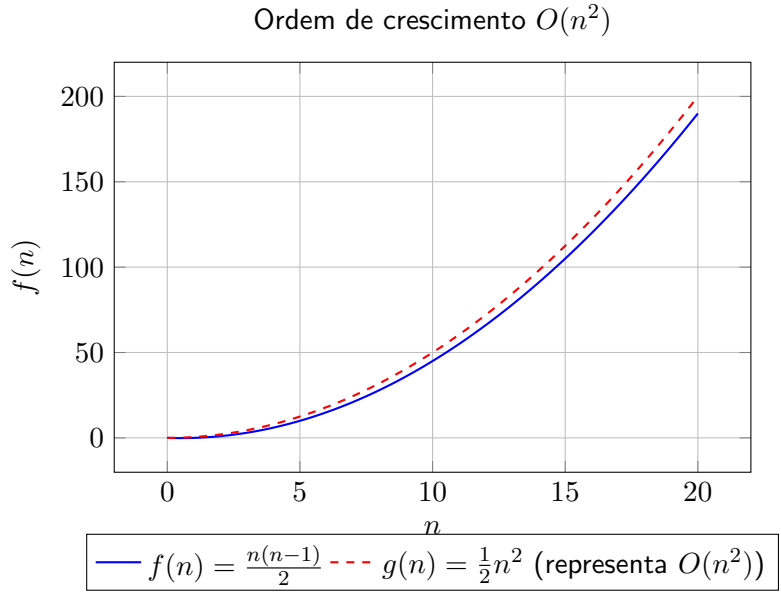
Agora, para determinar a ordem de crescimento de uma função f, usamos a notação "Big-O". De forma simplificada, essa notação reduz a função ao seu termo de crescimento mais acelerado. Por exemplo, no caso do SWAP, temos:

$$f(n) = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = \frac{n^2}{2} - \frac{n}{2}$$

O termo de maior crescimento é $\frac{n^2}{2}$, portanto dizemos que a ordem de crescimento de $f(n)$ é:

$$f(n) = O(n^2)$$

Ainda, o tamanho da vizinhança SWAP cresce de forma quadrática em função do tamanho da instância n.


Ordem de crescimento $O(n^2)$
200
150
100
50
0
$f(n)$
0
5
10
15
20
n
$f(n) = \frac{n(n-1)}{2}$
$g(n) = \frac{1}{2}n^2$ (representa $O(n^2)$)

Tamanho de vizinhança

Com essa terminologia em mente, Ahuja et. al define o conceito de **Very-Large Neighborhoods** como vizinhanças que **crescem de forma exponencial em função do tamanho das instâncias** n, e propõe uma classificação de algoritmos que resolvem problemas com essas vizinhanças.

1

[1] Para um estudo de vizinhanças exponenciais para o TSP veja *2000 - A study of exponential neighborhoods for the Travelling Salesman Problem and for the Quadratic Assignment Problem*

Tamanho de vizinhança

Com essa terminologia em mente, Ahuja et. al define o conceito de **Very-Large Neighborhoods** como vizinhanças que **crescem de forma exponencial em função do tamanho das instâncias** n, e propõe uma classificação de algoritmos que resolvem problemas com essas vizinhanças.

1

Nós estudaremos 2 deles:

1. LNS - Large Neighborhood Search
2. ALNS - Adaptive Large Neighborhood Search

[1] Para um estudo de vizinhanças exponenciais para o TSP veja *2000 - A study of exponential neighborhoods for the Travelling Salesman Problem and for the Quadratic Assignment Problem*

LNS - A idéia

A meta-heurística **L**arge **N**eighborhood **S**earch foi proposta por Shaw (*1998 - Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems - Shaw*)

LNS - A idéia

A meta-heurística **L**arge **N**eighborhood **S**earch foi proposta por Shaw (*1998 - Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems - Shaw*)

O LNS usa vizinhanças de forma implicita, por meio de operadores de **destruição** e de **reconstrução**

LNS - A idéia

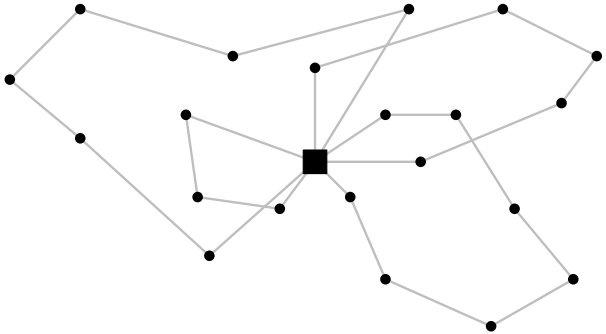
A meta-heurística **L**arge **N**eighborhood **S**earch foi proposta por Shaw (*1998 - Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems - Shaw*)

> O LNS usa vizinhanças de forma implicita, por meio de operadores de **destruição** e de **reconstrução**

1. DESTRUIÇÃO: Destrói parte da solução, usualmente de forma estocástica.
2. RECONSTRUÇÃO: Reconstrói a solução destruída, retornando-a para a factibilidade.

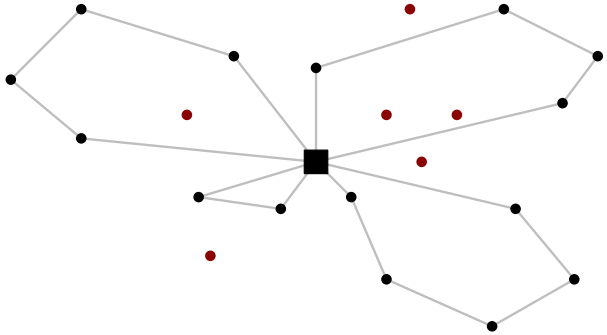
A vizinhança $N(s)$ de uma solução é definida então pelas soluções que podem ser alcançadas pela aplicação do operador de destruição e depois de reconstrução.

LNS - Exemplo VRP



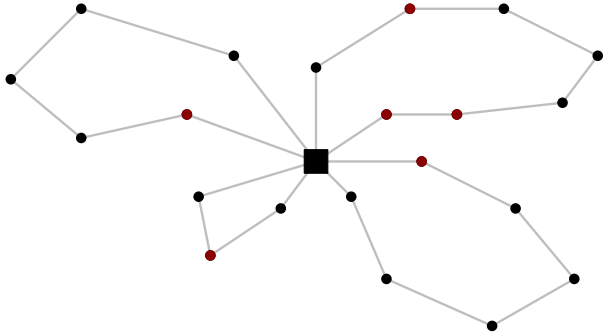
Considere uma solução do VRP.

LNS - Exemplo VRP



Um operador de destruição poderia simplesmente selecionar 15% dos clientes de forma aleatória, e removê-los de suas rotas.

LNS - Exemplo



Em seguida um operador de reconstrução poderia reinseri-los novamente, usando alguma heurística gulosa, por exemplo.

LNS - Pseudocódigo

Algorithm 1 Template genérico LNS

$x =$ GeraSolInicial() ▷ Gera uma solução inicial
$x^b = x$
repeat
  $x^t = r(d(x))$ ▷ destrói e reconstrói
  if aceita(x^t, x) **then** ▷ critério de aceite
    $x = x^t$
  end if
  if $c(x^t) < c(x^b)$ **then**
    $x^b = x^t$
  end if
  $x = x^b$
until Criterio de parada
return x^b.

LNS - Considerações

Algumas considerações:

LNS - Considerações

Algumas considerações:

1. No método original, a função "aceita"só aceita soluções melhores, artigos posteriores usaram critérios como o SA.

LNS - Considerações

Algumas considerações:

1. No método original, a função "aceita"só aceita soluções melhores, artigos posteriores usaram critérios como o SA.
2. É importante implementar uma **taxa de destruição** para o operador de destruição: se somente uma pequena parte da solução é destruída, a busca pode ficar muito limitada, se muito for destruída, é como um reinicio aleatório.

LNS - Considerações

Algumas considerações:

1. No método original, a função "aceita"só aceita soluções melhores, artigos posteriores usaram critérios como o SA.
2. É importante implementar uma **taxa de destruição** para o operador de destruição: se somente uma pequena parte da solução é destruída, a busca pode ficar muito limitada, se muito for destruída, é como um reinicio aleatório.
3. Shaw propõe um aumento gradativo da taxa de destruição, já Ropke and Pisinger selecionam de forma aleatória, para cada iteração, dentro de um range definido de acordo com o tamanho da instância.

ALNS - Diferença

A meta-heurística **A**daptive **L**arge **N**eighborhood **S**earch (ALNS), proposta em *2006 - An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows - Ropke, Pisinger*, é uma generalização da LNS, permitindo **múltiplos operadores de destruição e reconstrução**.

ALNS - Diferença

A meta-heurística **A**daptive **L**arge **N**eighborhood **S**earch (ALNS), proposta em *2006 - An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows - Ropke, Pisinger*, é uma generalização da LNS, permitindo **múltiplos operadores de destruição e reconstrução**.

A escolha de qual operador usar é controlada dinamicamente durante a busca (de forma **adaptativa** - daí o nome), usando a performance de cada operador em iterações passadas. O mecanismo de adaptação tende a usar mais os operadores que mais contribuem para a melhoria das soluções

Notação

Considere a seguinte notação:

1. Ω^-: Conjunto de todos os operadores de destruição.
2. Ω^+: Conjunto de todos os operadores de reconstrução.
3. ρ^-: Conjunto com os pesos para cada operador de destruição.
4. ρ^+: Conjunto com os pesos para cada operador de construção.

Notação

Considere a seguinte notação:

1. Ω^-: Conjunto de todos os operadores de destruição.
2. Ω^+: Conjunto de todos os operadores de reconstrução.
3. ρ^-: Conjunto com os pesos para cada operador de destruição.
4. ρ^+: Conjunto com os pesos para cada operador de construção.

Vejamos o algoritmo completo, junto ao LNS previamente visto. As partes em vermelho são as contribuições do ALNS, todo o **resto** é exatamente o mesmo LNS já visto.

ALNS - Pseudocódigo

x = GeraSolInicial(); $x^b = x$ ▷ Gera uma solução inicial
$\rho^- = (1, ..., 1); \rho^+ = (1, ..., 1)$ ▷ Pesos iguais no começo
repeat
  $d = selecionaD(\Omega^-, \rho^-); r = selecionaR(\Omega^+, \rho^+)$
  $x^t = r(d(x))$ ▷ destrói e reconstrói
  if aceita(x^t, x) **then** ▷ critério de aceite
    $x = x^t$
  end if
  if $c(x^t) < c(x^b)$ **then**
    $x^b = x^t$
  end if
  $x = x^b$
  atualiza ρ^- e ρ^+ ▷ Atualiza os pesos
until Criterio de parada
return x^b.

Mecanismo adaptativo

Agora precisamos entender como o mecanismo de escolha de operadores é auto regulado. Inicialmente os pesos ρ^+ e ρ^- são iguais, de forma que todas as vizinhanças possuem as mesmas probabilidades de serem selecionadas.

Mecanismo adaptativo

Agora precisamos entender como o mecanismo de escolha de operadores é auto regulado. Inicialmente os pesos ρ^+ e ρ^- são iguais, de forma que todas as vizinhanças possuem as mesmas probabilidades de serem selecionadas. A escolha é feita pelo método da roleta, por exemplo, considere ρ^+ para 5 vizinhanças:

$$\rho^+ = (1, 1, 1, 1, 1)$$

Mecanismo adaptativo

Agora precisamos entender como o mecanismo de escolha de operadores é auto regulado. Inicialmente os pesos ρ^+ e ρ^- são iguais, de forma que todas as vizinhanças possuem as mesmas probabilidades de serem selecionadas. A escolha é feita pelo método da roleta, por exemplo, considere ρ^+ para 5 vizinhanças:

$$\rho^+ = (1, 1, 1, 1, 1)$$

As **probabilidades de escolha** ficam:

$$\rho^+ = (0.2, 0.2, 0.2, 0.2, 0.2)$$

Mecanismo adaptativo

Agora precisamos entender como o mecanismo de escolha de operadores é auto regulado. Inicialmente os pesos ρ^+ e ρ^- são iguais, de forma que todas as vizinhanças possuem as mesmas probabilidades de serem selecionadas. A escolha é feita pelo método da roleta, por exemplo, considere ρ^+ para 5 vizinhanças:

$$\rho^+ = (1, 1, 1, 1, 1)$$

As **probabilidades de escolha** ficam:

$$\rho^+ = (0.2, 0.2, 0.2, 0.2, 0.2)$$

Para usar a roleta basta construirmos um **vetor de probabilidades acumuladas**:

$$\rho^+ = (0.2, 0.4, 0.6, 0.8, 1.0)$$

E selecionarmos um número aleatório entre 0 e 1, determinando o operador da faixa selecionada

Mecanismo adaptativo

Os pesos ρ são ajustados dinamicamente baseado no desempenho armazenado de cada operador de destruição e reconstrução. Ao fim de cada iteração, um score ψ é calculado pela fórmula:

$$\psi = max \begin{cases} \omega_1 : \text{se a nova solução é a melhor global} \\ \omega_2 : \text{se a nova solução é melhor que a atual} \\ \omega_3 : \text{se a nova solução é aceita} \\ \omega_4 : \text{se a nova solução é rejeitada} \end{cases}$$

Mecanismo adaptativo

Os pesos ρ são ajustados dinamicamente baseado no desempenho armazenado de cada operador de destruição e reconstrução. Ao fim de cada iteração, um score ψ é calculado pela fórmula:

$$\psi = max \begin{cases} \omega_1 : \text{se a nova solução é a melhor global} \\ \omega_2 : \text{se a nova solução é melhor que a atual} \\ \omega_3 : \text{se a nova solução é aceita} \\ \omega_4 : \text{se a nova solução é rejeitada} \end{cases}$$

Em que $\omega_1, \omega_2, \omega_3, \omega_4$ são parâmetros, geralmente com $\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4$. Ou seja, **um alto valor de ψ indica que a iteração (portanto operador de destruição e reconstrução) foi um sucesso**.

Mecanismo adaptativo

Agora, com o valor de ψ os pesos ρ^+ e ρ^- podem ser atualizados. Considerando a e b os índices dos operadores de destruição e reconstrução usados na última iteração, temos que:

$$\rho_a^- = \alpha\rho_a^- + (1-\alpha)\psi$$
$$\rho_b^+ = \alpha\rho_b^+ + (1-\alpha)\psi$$

Com $\alpha \in [0,1]$. O parâmetro α é chamado de decaimento (*decay*), e controla o quão sensível os pesos são às mudanças na performance dos operdores. Altos valores de α são mais **conservadores**, mantendo a maior parte dos novos pesos como estavam antes da iteração. Já valores baixos favorecem mais alterações.

Mecanismo adaptativo

Vejamos um exemplo. Suponha que estamos na primeira iteração, portanto :

$$\rho^+ = (1, 1, 1, 1, 1)$$
$$\rho^- = (1, 1, 1, 1, 1)$$

Mecanismo adaptativo

Vejamos um exemplo. Suponha que estamos na primeira iteração, portanto :

$$\rho^+ = (1, 1, 1, 1, 1)$$

$$\rho^- = (1, 1, 1, 1, 1)$$

Considerando ainda que neste iteração usamos a vizinhança de destruição 0 ($a = 0$) e de reconstrução 2 ($b = 2$.)

Mecanismo adaptativo

Vejamos um exemplo. Suponha que estamos na primeira iteração, portanto :

$$\rho^+ = (1, 1, 1, 1, 1)$$

$$\rho^- = (1, 1, 1, 1, 1)$$

Considerando ainda que neste iteração usamos a vizinhança de destruição 0 ($a = 0$) e de reconstrução 2 ($b = 2$.)
Com os seguintes valores de ω

$$\psi = max \begin{cases} \omega_1 = 8 : \text{se a nova solução é a melhor global} \\ \omega_2 = 4 : \text{se a nova solução é melhor que a atual} \\ \omega_3 = 2 : \text{se a nova solução é aceita} \\ \omega_4 = 1 : \text{se a nova solução é rejeitada} \end{cases}$$

Mecanismo adaptativo

Vejamos um exemplo. Suponha que estamos na primeira iteração, portanto :

$$\rho^+ = (1, 1, 1, 1, 1)$$
$$\rho^- = (1, 1, 1, 1, 1)$$

Considerando ainda que neste iteração usamos a vizinhança de destruição 0 ($a = 0$) e de reconstrução 2 ($b = 2$.)
Com os seguintes valores de ω

$$\psi = max \begin{cases} \omega_1 = 8 : \text{se a nova solução é a melhor global} \\ \omega_2 = 4 : \text{se a nova solução é melhor que a atual} \\ \omega_3 = 2 : \text{se a nova solução é aceita} \\ \omega_4 = 1 : \text{se a nova solução é rejeitada} \end{cases}$$

E finalmente, após essa iteração a nova solução ficou melhor que a atual.

Mecanismo adaptativo

Assim, precisamos usar as fórmulas:

$$\rho_a^- = \alpha \rho_a^- + (1 - \alpha)\psi$$
$$\rho_b^+ = \alpha \rho_b^+ + (1 - \alpha)\psi$$

Mecanismo adaptativo

Assim, precisamos usar as fórmulas:

$$\rho_a^- = \alpha\rho_a^- + (1-\alpha)\psi$$
$$\rho_b^+ = \alpha\rho_b^+ + (1-\alpha)\psi$$

Com $\psi = 4$, $a = 0$ e $b = 2$. Só faltando determinar o coeficiente de sensibilidade α. Vamos calcular com 2 valores de α e ver o que ocorre (0.2 e 0.8):

Mecanismo adaptativo

Assim, precisamos usar as fórmulas:

$$\rho_a^- = \alpha\rho_a^- + (1-\alpha)\psi$$
$$\rho_b^+ = \alpha\rho_b^+ + (1-\alpha)\psi$$

Com $\psi = 4$, $a = 0$ e $b = 2$. Só faltando determinar o coeficiente de sensibilidade α. Vamos calcular com 2 valores de α e ver o que ocorre (0.2 e 0.8):

$$\rho_0^- = 0.2 \cdot 1 + (1-0.2)4 = 3.4$$
$$\rho_2^+ = 0.2 \cdot 1 + (1-0.2)4 = 3.4$$

Mecanismo adaptativo

Assim, precisamos usar as fórmulas:

$$\rho_a^- = \alpha\rho_a^- + (1-\alpha)\psi$$
$$\rho_b^+ = \alpha\rho_b^+ + (1-\alpha)\psi$$

Com $\psi = 4$, $a = 0$ e $b = 2$. Só faltando determinar o coeficiente de sensibilidade α. Vamos calcular com 2 valores de α e ver o que ocorre (0.2 e 0.8):

$$\rho_0^- = 0.2 \cdot 1 + (1-0.2)4 = 3.4$$
$$\rho_2^+ = 0.2 \cdot 1 + (1-0.2)4 = 3.4$$

$$\rho_0^- = 0.8 \cdot 1 + (1-0.8)4 = 1.59$$
$$\rho_2^+ = 0.8 \cdot 1 + (1-0.8)4 = 1.59$$

Mecanismo adaptativo

Dessa forma, os novos vetores de pesos (para os dois valores de α) ficam:

$$\rho^+ = (3.4, 1, 1, 1, 1)$$
$$\rho^- = (1, 1, 3.4, 1, 1)$$

$$\rho^+ = (1.59, 1, 1, 1, 1)$$
$$\rho^- = (1, 1, 1.59, 1, 1)$$

Mecanismo adaptativo

Dessa forma, os novos vetores de pesos (para os dois valores de α) ficam:

$$\rho^+ = (3.4, 1, 1, 1, 1)$$
$$\rho^- = (1, 1, 3.4, 1, 1)$$

$$\rho^+ = (1.59, 1, 1, 1, 1)$$
$$\rho^- = (1, 1, 1.59, 1, 1)$$

O que confirma que o α **determina a sensibilidade dos pesos a mudanças** - um α menor gera pesos mais sensíveis, e um α maior gera menos sensibilidade.

Conclusões

Conclusões

1. As meta-heurísticas LNS e ALNS são usadas quando as vizinhanças para um problema são muito grandes, de forma que os algoritmos trabalham com **vizinhanças implícitas**, por meio dos operadores de destruição e reconstrução.

Conclusões

1. As meta-heurísticas LNS e ALNS são usadas quando as vizinhanças para um problema são muito grandes, de forma que os algoritmos trabalham com **vizinhanças implícitas**, por meio dos operadores de destruição e reconstrução.
2. Também podem ser usadas em problemas que a **definição de vizinhanças não é tão clara**, ou mesmo não é possível.

Conclusões

1. As meta-heurísticas LNS e ALNS são usadas quando as vizinhanças para um problema são muito grandes, de forma que os algoritmos trabalham com **vizinhanças implícitas**, por meio dos operadores de destruição e reconstrução.
2. Também podem ser usadas em problemas que a **definição de vizinhanças não é tão clara**, ou mesmo não é possível.
3. Se a solução inicial foi criada por um **algoritmo guloso**, podemos aproveitar a própria heurística de escolha dos elementos gulosos como um operador de reconstrução. Assim, somente determinando um operador de destruição já temos uma meta-heurística muito poderosa de forma simples a partir do *greedy*.

Conclusões

1. As meta-heurísticas LNS e ALNS são usadas quando as vizinhanças para um problema são muito grandes, de forma que os algoritmos trabalham com **vizinhanças implícitas**, por meio dos operadores de destruição e reconstrução.
2. Também podem ser usadas em problemas que a **definição de vizinhanças não é tão clara**, ou mesmo não é possível.
3. Se a solução inicial foi criada por um **algoritmo guloso**, podemos aproveitar a própria heurística de escolha dos elementos gulosos como um operador de reconstrução. Assim, somente determinando um operador de destruição já temos uma meta-heurística muito poderosa de forma simples a partir do *greedy*.
4. Devido a alta flexibilidade da escolha de operadores no ALNS, esta meta-heurística se encaixa muito bem para ser usada em *mathheuristics* (por exemplo, destruindo uma parte da solução e **reconstruindo-a usando um modelo matemático** com métodos exatos)

Atividade

1. Considerando o artigo *2000 - A study of exponential neighborhoods for the Travelling Salesman Problem and for the Quadratic Assignment Problem*, entenda a vizinhança ASSIGN para o TSP e crie um exemplo numérico explicando como a mesma funciona, e a sua relação com o problema de designação.
2. Pisisnger e Ropke usaram ALNS como um algoritmo que resolve diversas variantes do problema VRP de uma só vez, em *2007 - A general heurisitc for vehicle routing problems - Ropke, Pisinger*. No artigo eles também fazem uma explicação da ALNS. Leia o artigo e as partes destacadas, relacionando com a aula. Identifique os operadores de destruição e reconstrução utilizados.