Resumo artigo PAA (metaheurística)

Introdução

As metaheur´ısticas representam um conjunto de t´ecnicas de otimiza¸c˜ao adaptadas para lidar com problemas complexos e que apresentam a caracter´ıstica da explos˜ao combinat´oria.

Aplica em: problemas de otimiza¸c˜ao matem´atica complexa e que requerem tempo de processamento elevado. Uma heur´ıstica ´e uma t´ecnica de otimiza¸c˜ao que atrav´es de passos muito bem definidos encontra uma solu¸c˜ao de boa qualidade de um problema complexo. o, uma heur´ıstica n˜ao tem capacidade de encontrar a solu¸c˜ao otima ´ global de um problema complexo. uma heur´ıstica encontra apenas um otimo ´ local, geralmente de pobre qualidade. Entretanto, uma heur´ıstica apresenta a vantagem de ser muito simples de formular e de implementar computacionalmente.

Uma heur´ıstica realiza um conjunto de transi¸c˜oes atrav´es do espa¸co de solu¸c˜oes do problema, iniciando o processo de um ponto do espa¸co de busca e terminando em um ponto de otimo ´ local. A diferen¸ca entre os diferentes algoritmos heur´ısticos est´a relacionada com a escolha do ponto inicial para iniciar as transi¸c˜oes, a caracteriza¸c˜ao da vizinhan¸ca e o crit´erio usado para escolher o pr´oximo ponto, isto ´e, o melhor vizinho. O algoritmo heur´ıstico mais popular ´e o algoritmo heur´ıstico construtivo onde a cada passo ´e escolhida uma componente da solu¸c˜ao e o processo termina quando ´e encontrada uma solu¸c˜ao fact´ıvel para o problema.

2 Id´eia Fundamental das Metaheur´ısticas

Espaço de busca: espa¸co de busca o conjunto de solu¸c˜oes candidatas de um problema de acordo com a forma escolhida para representar ou codificar uma proposta de solu¸c˜ao de um problema.

A id´eia fundamental de uma metaheur´ıstica consiste em analisar ou visitar apenas um conjunto reduzido do espa¸co de busca, considerando que o

espa¸co de busca ´e absurdamente grande. Você parte um ponto escolhido com algum critério e segue rumo a uma solução. uma metaheur´ıstica ´e uma estrat´egia que especifica a forma em que deve ser realizada a busca de forma inteligente, isto ´e, a forma em que devem ser realizadas as transi¸c˜oes atrav´es do espa¸co de busca partindo de um ponto inicial ou de um conjunto de pontos.

(1) especificar uma forma de identificar ou representar um elemento do espa¸co de busca, isto ´e, uma proposta de solu¸c˜ao do problema, (2) especificar a forma de encontrar a fun¸c˜ao objetivo ou seu equivalente para cada proposta de solu¸c˜ao, (3) especificar a vizinhan¸ca da solu¸c˜ao corrente, (4) especificar se a forma de realizar as transi¸c˜oes deve ser realizada a partir de um unico ´ ponto ou de um conjunto de pontos (ou propostas de solu¸c˜ao) e, (5) se o processo de busca deve ser realizado atrav´es de solu¸c˜oes fact´ıveis ou podem ser consideradas tamb´em solu¸c˜oes infact´ıveis no processo de busca. A forma de resolver cada um desses problemas depende da metaheur´ıstica usada e do tipo de problema que se pretende resolver. Analisamos brevemente esses assuntos.

Codificação: Define-se codifica¸c˜ao como uma forma consistente de representar uma proposta de solu¸c˜ao de um problema do espa¸co de solu¸c˜oes do problema. Uma proposta de solu¸c˜ao est´a adequadamente codificada quando, a partir dessa informa¸c˜ao, ´e poss´ıvel encontrar o valor da fun¸c˜ao objetivo (o seu equivalente) do problema e determinar se a proposta ´e fact´ıvel ou infact´ıvel.

Vizinhança: Para o codifica¸c˜ao proposta, pode-se definir v´arias formas de vizinhan¸ca, isto ´e, v´arias formas de identificar solu¸c˜oes vizinhas da solu¸c˜ao corrente. No processo de transi¸c˜oes de uma metaheur´ıstica, a partir da solu¸c˜ao corrente, passa-se a uma solu¸c˜ao vizinha de acordo com os crit´erios de cada tipo de algoritmo. Para a codifica¸c˜ao proposta, as seguintes propostas representam estruturas de vizinhan¸ca ou caracteriza¸c˜ao de vizinhan¸ca: (1) toda solu¸c˜ao obtida trocando um elemento de 0 para 1; (2) toda solu¸c˜ao obtida trocando um elemento de 1 para 0; (3) toda solu¸c˜ao obtida

trocando, simultaneamente, um elemento de 0 para 1 e outro elemento de 1 para 0; ou (4) todas as anteriores.

Deve-se observar que o numero ´ de vizinhos pode ser muito grande. Este **fato n˜ao representa problemas para algoritmos como simulated annealing** mas pode representar s´erios problemas de esfor¸co adicional de processamento para algoritmos tipo tabu search.

Problema de Infactibilidades: Normalmente ´e f´acil encontrar uma solu¸c˜ao fact´ıvel ou um conjunto de solu¸c˜oes fact´ıveis para o problema da mochila com a codifica¸c˜ao proposta. **Entretanto, operadores como a recombina¸c˜ao no algoritmo gen´etico podem produzir solu¸c˜oes infact´ıveis.** Nesse caso, a proposta mais simples pode consistir em eliminar essas solu¸c˜oes infact´ıveis. Entretanto, quando o tamanho do problema cresce o numero ´ de solu¸c˜oes infact´ıveis pode ser proibitivo descaracterizando o desempenho do operador de recombina¸c˜ao.

3 Simulated Annealing

3.1 Introdu¸c˜ao

O processo p´ara quando o material atinge seu estado de energia m´ınima, no qual se transforma num cristal perfeito. Assim, o algoritmo de simulated annealing tenta simular um processo equivalente para encontrar a configura¸c˜ao otima ´ de um problema complexo.

3.2 Id´eia Fundamental em Simulated Annealing

O algoritmo SA pode ser considerado como um processo de transi¸c˜oes atrav´es do espa¸co de busca de um problema complexo. Assim, comparado com outras metaheur´ısticas, SA apresenta duas caracter´ısticas fundamentais:

(1) a forma de escolha do vizinho mais interessante;

(2) o controle do processo de transi¸c˜oes.

SA escolhe o vizinho mais interessante atrav´es de uma l´ogica baseada no processo de annealing. Assim, escolhe-se aleatoriamente um vizinho da topologia corrente. Se esse vizinho for de melhor qualidade ent˜ao ´e realizada a transi¸c˜ao escolhendo esse vizinho como a nova topologia corrente. A escolha de um vizinho de pior qualidade, embora seja probabil´ıstica, est´a controlado por dois parˆametros:

(1) a temperatura;

(2) a varia¸c˜ao da fun¸c˜ao objetivo.

Essa probabilidade diminui atrav´es do processo chegando a ser praticamente nula nas fases finais do processo em que s˜ao realizadas apenas transi¸c˜oes para topologias vizinhas de melhor qualidade.

Neste algoritmo, aplica-se uma a¸c˜ao combinada do mecanismo de gera¸c˜ao de alternativas e do crit´erio de aceita¸c˜ao. Tk ´e o parˆametro de controle ou temperatura e Nk ´e o numero ´ de alternativas geradas na k-´esima itera¸c˜ao do algoritmo. Inicialmente quando T ´e grande, grandes deteriora¸c˜oes da fun¸c˜ao objetivo podem ser aceitas; quando T decresce, somente pequenas deteriora¸c˜oes s˜ao aceitas e finalmente quando T tende a zero, nenhuma deteriora¸c˜ao ´e aceita.

A partir da proposta de solu¸c˜ao i com custo f(i) gera-se o vizinho j com custo f(j) usando um mecanismo de gera¸c˜ao de vizinhan¸ca. O crit´erio de aceita¸c˜ao determina se este novo estado ´e aceito; assim deve-se calcular a seguinte probabilidade:

$$P_T\{aceita_j\} = \begin{cases} 1 & se \leftrightarrow f(j) \leq f(i) \\ e^{(\frac{f(i)-f(j)}{T})} & se \leftrightarrow f(j) > f(i) \end{cases}$$

Exemplo: Problema do Caixeiro Viajante

Uma particularidade muito importante do algoritmo SA para o PCV ´e que, para a codifica¸c˜ao proposta e com vizinhan¸ca tipo k-opt, todas as topologias vizinhas s˜ao fact´ıveis. **Assim n˜ao acontece o aparecimento de tours infact´ıveis como pode acontecer, por exemplo, com o algoritmo gen´etico ou algoritmos evolutivos em geral.**

4 O Algoritmo Gen´etico (15 slide)

O algoritmo gen´etico ´e uma metaheur´ıstica originalmente formulado usando os mecanismos da evolu¸c˜ao e da gen´etica natural. Este algoritmo foi inventado por Holland na d´ecada de 70 [17].

A evolu¸c˜ao das esp´ecies est´a determinada por um processo de sele¸c˜ao que leva a` sobrevivˆencia dos indiv´ıduos gen´eticamente melhor adaptados para superar os problemas do meio ambiente que geralmente s˜ao vari´aveis. O conceito de geneticamente melhor adaptado tem um valor relativo porque depende do fator problem´atico que existe no meio ambiente. Assim, as mudan¸cas do meio ambiente determinam, de maneira significativa, a mudan¸ca na composi¸c˜ao gen´etica dos elementos de uma popula¸c˜ao.

Os elementos da popula¸c˜ao gen´eticamente melhor qualificados tˆem maior possibilidade de chegar a` fase adulta e gerar descendentes, transmitindo suas caracter´ısticas gen´eticas para os descendentes e aumentando essas caracter´ısticas gen´eticas na popula¸c˜ao. Neste processo, existe uma componente aleat´oria muito importante na gera¸c˜ao dos indiv´ıduos da nova popula¸c˜ao.

4.1 Id´eia Fundamental do Algoritmo Gen´etico

O algoritmo gen´etico usa uma popula¸c˜ao de indiv´ıduos, que nos problemas combinat´orios significam um conjunto de configura¸c˜oes, para resolver um problema de otimiza¸c˜ao complexo. Portanto, o algoritmo gen´etico deve fazer o seguinte:

1. Representar adequadamente uma configura¸c˜ao do problema. A representa¸c˜ao mais popular ´e a representa¸c˜ao em codifica¸c˜ao bin´aria onde s˜ao facilmente simulados os operadores gen´eticos de recombina¸c˜ao e muta¸c˜ao.

2. Deve encontrar uma forma adequada para avaliar a fun¸c˜ao objetivo ou seu equivalente (fitness value). Assim, as melhores configura¸c˜oes s˜ao aquelas que apresentam fun¸c˜oes objetivo de melhor qualidade.

3. Deve existir uma estrat´egia de sele¸c˜ao das configura¸c˜oes com direito a participar na conforma¸c˜ao das configura¸c˜oes da nova popula¸c˜ao.

4. Deve existir um mecanismo que permita implementar o operador gen´etico de recombina¸c˜ao.

5. Deve existir um mecanismo que permita implementar o operador gen´etico de muta¸c˜ao. Muta¸c˜ao geralmente ´e considerado um operador gen´etico secund´ario nos algoritmos gen´eticos mas pesquisas recentes mostram que este operador ´e muito mais importante do que se imaginava inicialmente.

6. Deve-se especificar o tamanho da popula¸c˜ao, isto ´e, o numero ´ de configura¸c˜oes em cada gera¸c˜ao.

4.2 Algoritmo Gen´etico B´asico

Exemplo: O problema da mochila

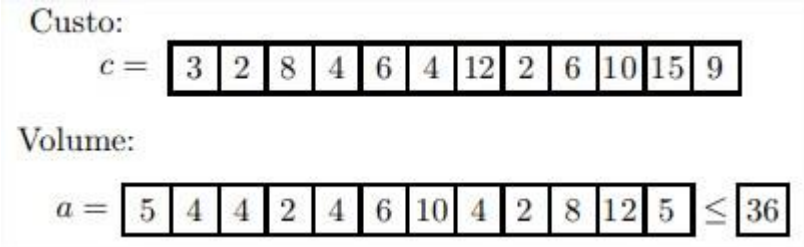
Nesta se¸c˜ao s˜ao analisadas, em detalhe, as principais caracter´ısticas de um algoritmo gen´etico b´asico (AGB). Para ilustrar as caracter´ısticas fundamentais de um AGB usamos um exemplo do problema da mochila. Exemplo: O problema da mochila Seja o problema da mochila para n = 12 mostrado a seguir:

max z(x) = 3x1 + 2x2 + 8x3 + 4x4 + 6x5 + 4x6 +12x7 + 2x8 + 6x9 + 10x10 + 15x11 + 9x12

s.a. 5x1 + 4x2 + 4x3 + 2x4 + 4x5 + 6x6 + 10x7 +4x8 + 2x9 + 8x10 + 12x11 + 5x12 ≤ 36

xj ∈ {0, 1}, j = 1, . . . , 12.

Os vetores de trabalho podem ser facilmente identificados:



Este exemplo deve ser usado para ilustrar a implementa¸c˜ao do AGB.

4.2.1 Algoritmo Gen´etico Elementar

Um algoritmo gen´etico elementar realiza a seguinte sequˆencia de opera¸c˜oes:

1. Gera a popula¸c˜ao inicial ap´os escolher o tipo de codifica¸c˜ao.

2. Calcula a fun¸c˜ao objetivo de cada configura¸c˜ao da popula¸c˜ao e, armazena e atualiza a incumbente (melhor configura¸c˜ao encontrada durante o processo).

3. Implementa a sele¸c˜ao.

4. Implementa a recombina¸c˜ao.

5. Implementa a muta¸c˜ao e termina de gerar a popula¸c˜ao da nova gera¸c˜ao.

6. Se o crit´erio de parada (ou crit´erios de parada) n˜ao for satisfeito, repetir os passos 2 a 6. A seguir s˜ao analisadas todas as etapas do AGB usando exemplos ilustrativos.

4.3 O Problema da Codifica¸c˜ao

O AGB exige que a codifica¸c˜ao seja em codifica¸c˜ao bin´aria. Portanto, independentemente da forma das vari´aveis de decis˜ao, deve-se transformar em uma codifica¸c˜ao bin´aria. Entretanto, quando as vari´aveis de decis˜ao do problema s˜ao reais ou inteiros, a codifica¸c˜ao bin´aria pode apresentar problemas de desempenho adequado. Os algoritmos gen´eticos mais avan¸cados e os algoritmos evolutivos em geral recomendam que a codifica¸c˜ao seja realizada respeitando as caracter´ısticas espec´ıficas do problema e, portanto, quando necess´ario, deve-se redefinir os operadores gen´eticos para permitir trabalhar com formas de codifica¸c˜ao n˜ao bin´arias. Neste trabalho usamos como exemplo o problema da mochila em que as vari´aveis de decis˜ao s˜ao bin´arias e a codifica¸c˜ao bin´aria ´e a mais adequada.

Exemplo: Fun¸c˜ao objetivo:

Para o problema da mochila mostrado no exemplo, uma popula¸c˜ao t´ıpica de 4 elementos e usando a codifica¸c˜ao bin´aria apresenta a seguinte forma:

		Recurso usado	f.o.
P_1 :	1 1 1 1 1 1 0 0 1 1 0 0	$b' = 35$	43
P_2 :	0 1 0 1 1 0 0 1 0 1 1 0	$b' = 34$	39
P_3 :	0 1 1 0 0 0 1 1 0 0 0 1	$b' = 29$	33
P_4 :	0 0 1 0 0 0 1 0 0 1 1 0	$b' = 34$	45

População inicial: 4 configurações factíveis.

em que ´e indicada a fun¸c˜ao objetivo de cada proposta de solu¸c˜ao. Essa popula¸c˜ao deve ser usada para implementar os operadores gen´eticos.

4.5 O Operador de Sele¸c˜ao

A sele¸c˜ao ´e o operador gen´etico que permite selecionar as configura¸c˜oes da popula¸c˜ao corrente que devem participar da forma¸c˜ao da nova popula¸c˜ao. Portanto, o operador de sele¸c˜ao termina ap´os decidir o numero ´ de descendentes que deve ter cada configura¸c˜ao da popula¸c˜ao corrente.

Na sele¸c˜ao proporcional, cada configura¸c˜ao tem direito de gerar um numero ´ de descendentes que ´e proporcional ao valor de sua fun¸c˜ao de adapta¸c˜ao. Esta proposta tem a desvantagem de gerar um numero ´ de descendentes n˜ao inteiro. Esse problema ´e contornado usando uma roleta.

4.5.1 Sele¸c˜ao Usando Torneio (tournament selection)

Nesta estrat´egia, os descendentes s˜ao escolhidos realizando np jogos sendo np o tamanho da popula¸c˜ao. Em cada jogo s˜ao escolhidos aleatoriamente k configura¸c˜oes e a configura¸c˜ao ganhadora do jogo ´e aquela que tem fun¸c˜ao de adapta¸c˜ao de melhor qualidade. O valor de k ´e geralmente pequeno, tipicamente $k \in \{2, 3, 4\}$. Ap´os np jogos se termina o processo de sele¸c˜ao.

Este tipo de sele¸c˜ao ´e atrativo porque ´e computacionalmente muito r´apido e, tamb´em, porque a estrat´egia ´e a mesma para problemas de maximiza¸c˜ao e minimiza¸c˜ao, apenas o crit´erio de fun¸c˜ao de adapta¸c˜ao de melhor qualidade ´e diferente. Outra vantagem do m´etodo ´e que elimina os problemas existentes na sele¸c˜ao proporcional tradicional porque a sele¸c˜ao depende apenas dos valores relativos das fun¸c˜oes de adapta¸c˜ao. Finalmente, o processo encontra um numero ´ de descendentes inteiro e, portanto, dispensa o uso da roleta. Embora muito simples, este m´etodo ´e r´apido e eficiente. Usando a sele¸c˜ao baseada em torneio para a popula¸c˜ao mostrada na p´agina anterior, P1 e P2 tem direito a gerar um descendente e P4 dois descendentes.

4.6 Recombina¸c˜ao

As configura¸c˜oes escolhidas no processo de sele¸c˜ao devem ser submetidas ao operador de recombina¸c˜ao. No algoritmo gen´etico, a recombina¸c˜ao consiste em trocar parcelas de duas configura¸c˜oes para formar duas novas configura¸c˜oes candidatas. Em outras palavras, duas configura¸c˜oes candidatas s˜ao geradas com parcelas de duas configura¸c˜oes geradoras. Essas novas configura¸c˜oes geradas devem ainda ser submetidas ao operador de muta¸c˜ao para que se transformem em configura¸c˜oes da nova popula¸c˜ao ou gera¸c˜ao. O operador de recombina¸c˜ao tenta simular o fenˆomeno de crossing over na gen´etica.

T**axa de recombinação:** A taxa de recombina¸c˜ao determina, em forma aleat´oria, se duas configura¸c˜oes selecionadas devem ser submetidas a recombina¸c˜ao.

4.6.1 Recombina¸c˜ao de um Simples Ponto (single point recombination)

A mais simples forma de recombina¸c˜ao consiste em escolher um unico ´ ponto para fazer recombina¸c˜ao. Supor que uma configura¸c˜ao tem k elementos ou casas bin´arias. Ent˜ao, uma vez escolhidas as duas configura¸c˜oes para implementar a recombina¸c˜ao, devese gerar em forma aleat´oria um numero ´ entre 1 e (k − 1) e, esse numero ´ indica o ponto de recombina¸c˜ao. A parcela que est´a na direita de ambas as configura¸c˜oes s˜ao trocadas para formar as duas novas configura¸c˜oes candidatas. As configura¸c˜oes que devem ser submetidas a recombina¸c˜ao s˜ao escolhidas aleatoriamente do conjunto de configura¸c˜oes selecionadas que ainda tem direito a gerar descendentes. Para que as configura¸c˜oes selecionadas sejam submetidas a recombina¸c˜ao, deve-se gerar um numero ´ aleat´orio $p \in [0, 1]$. Se p ´e menor que a taxa de recombina¸c˜ao ρc ent˜ao, deve-se proceder a` recombina¸c˜ao; em caso contr´ario as duas configura¸c˜oes selecionadas n˜ao s˜ao recombinadas.

Neste processo de recombina¸c˜ao, novamente, est˜ao presentes trˆes decis˜oes de car´ater aleat´orio:

(1) as duas configura¸c˜oes que devem ser submetidas a recombina¸c˜ao s˜ao escolhidas aleatoriamente;

(2) o ponto de recombina¸c˜ao ´e escolhido aleatoriamente;

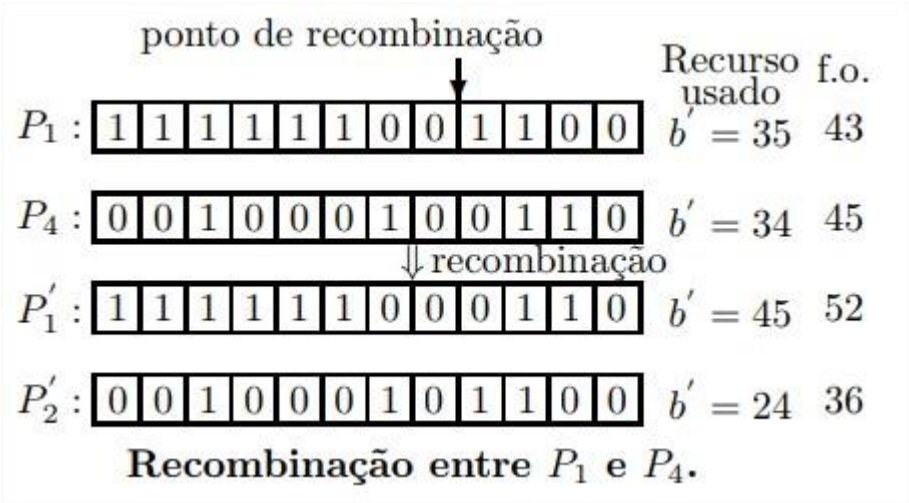
(3) deve-se gerar um numero ´ aleat´orio p que determina se as configura¸c˜oes selecionadas devem ser submetidas a recombina¸c˜ao.

Exemplo: Recombina¸c˜ao de um ponto:

Implementar a recombina¸c˜ao de um ponto para as configura¸c˜oes selecionadas no problema da mochila do exemplo que est´a sendo analisado. Considere uma taxa de recombina¸c˜ao de ρc = 1, 0. Em forma aleat´oria s˜ao escolhidos os seguintes pares.

{P1 com P4} e {P2 com P4}

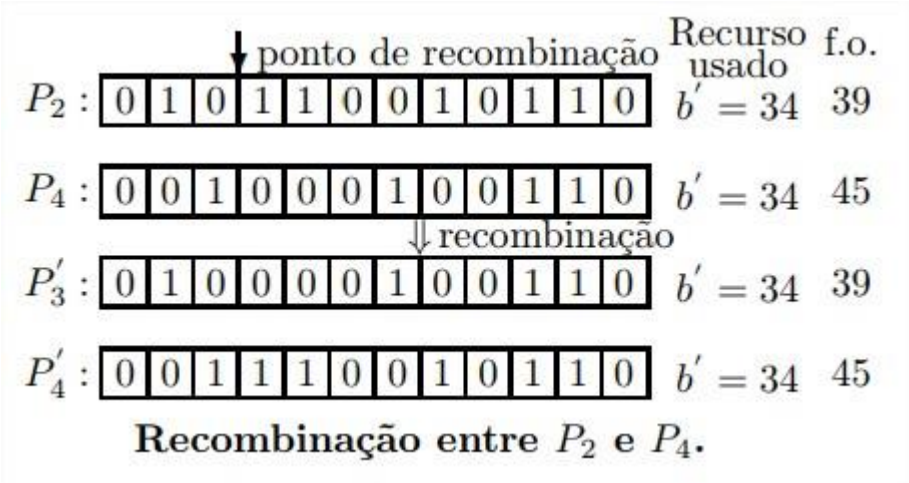
1. P1 e P4 s˜ao submetidas a recombina¸c˜ao para gerar duas configura¸c˜oes candidatas. E´ gerado um numero ´ aleat´orio p = 8 ∈ {1, 11}. Portanto, a recombina¸c˜ao gera as seguintes configura¸c˜oes candidatas:



Recombinação entre P_1 e P_4.

Deve-se observar que a configura¸c˜ao candidata P′ 1 est´a infact´ıvel porque 45 > 36 (a restri¸c˜ao est´a violada).

2. P2 e P4 s˜ao submetidas a recombina¸c˜ao para gerar duas configura¸c˜oes candidatas. E´ gerado o numero ´ aleat´orio p = 3 ∈ {1, 11}. Portanto, a recombina¸c˜ao gera as seguintes configura¸c˜oes candidatas:



Recombinação entre P_2 e P_4.

Neste caso as duas configura¸c˜oes candidatas s˜ao fact´ıveis.

se vc reparar nessa recombinação, vai notar que a partir do índice p sorteado aleatoriamente, todos os outros em diante são recombinados. A Recombinação nesse olha para o mesmo índice dos dois vetores e se comporta da seguinte forma:

- Se Px(p) == Py (p), mantem o valor nos dois;
- Se Px(p) != Py (p), muda 1 para 0 e zero para um em seus respectivos vetores.

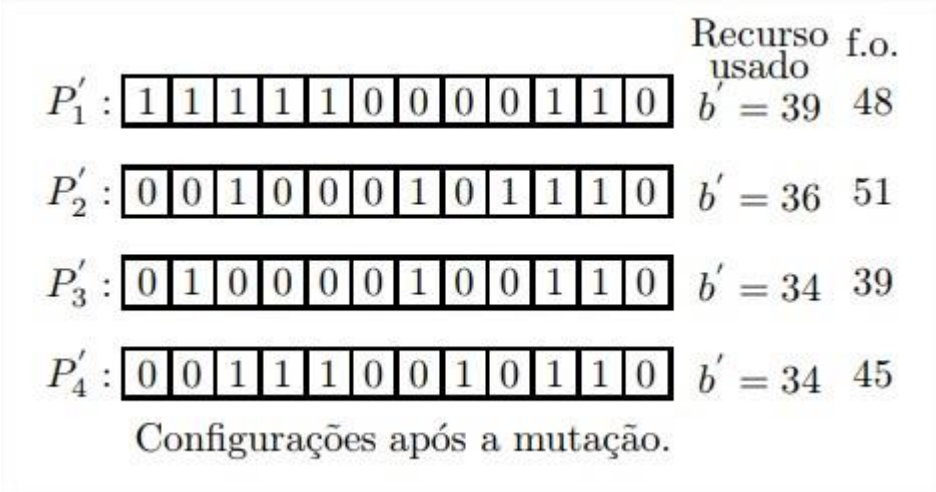
4.7 Muta¸c˜ao (mutation)

Na codifica¸c˜ao bin´aria o operador de muta¸c˜ao simplesmente troca o valor de uma vari´avel de 0 para 1 ou vice-versa.

Pm: A taxa de muta¸c˜ao ρm indica a probabilidade de que uma posi¸c˜ao ou casa bin´aria seja modificada. Na an´alise te´orica e nas propostas originais, sugerese que a muta¸c˜ao deve ser realizada bit por bit, casa por casa, e portanto a decis˜ao de mutar uma posi¸c˜ao deve ser independente da decis˜ao realizada em outras posi¸c˜oes bin´arias de uma configura¸c˜ao. Assim, supor que ´e escolhida uma taxa de muta¸c˜ao de ρm = 0, 05, ent˜ao cada bit de uma configura¸c˜ao ´e submetido a muta¸c˜ao com esta probabilidade. Desta forma, ´e gerado um numero ´ aleat´orio p ∈ [0, 1] e se esse numero ´ ´e menor que ρm = 0, 05 ent˜ao ´e realizada a muta¸c˜ao. Na implementa¸c˜ao da muta¸c˜ao tamb´em existe necessidade de gerar numeros ´ aleat´orios introduzindo, novamente, uma componente aleat´oria na implementa¸c˜ao do algoritmo gen´etico.

Exemplo: Muta¸c˜ao:

Implementar a muta¸c˜ao das configura¸c˜oes candidatas obtidas ap´os a recombina¸c˜ao no exemplo que est´a sendo analisado, usando uma taxa de muta¸c˜ao de ρm = 0, 05 e terminar de gerar as configura¸c˜oes da nova popula¸c˜ao. Na forma pr´atica de implementar a muta¸c˜ao, deve-se fazer 0, 05(4)(12) = 2, 4 muta¸c˜oes, isto ´e, 2 muta¸c˜oes. Assim, s˜ao escolhidos dois números aleat´orios entre 1 e 48, gerando-se os numeros ´ 6 e 23. Portanto, deve-se mutar a sexta posi¸c˜ao da primeira configura¸c˜ao e a d´ecima primeira posi¸c˜ao da segunda configura¸c˜ao. As novas configura¸c˜oes candidatas, ap´os a muta¸c˜ao assumem a seguinte forma:



Configurações após a mutação.

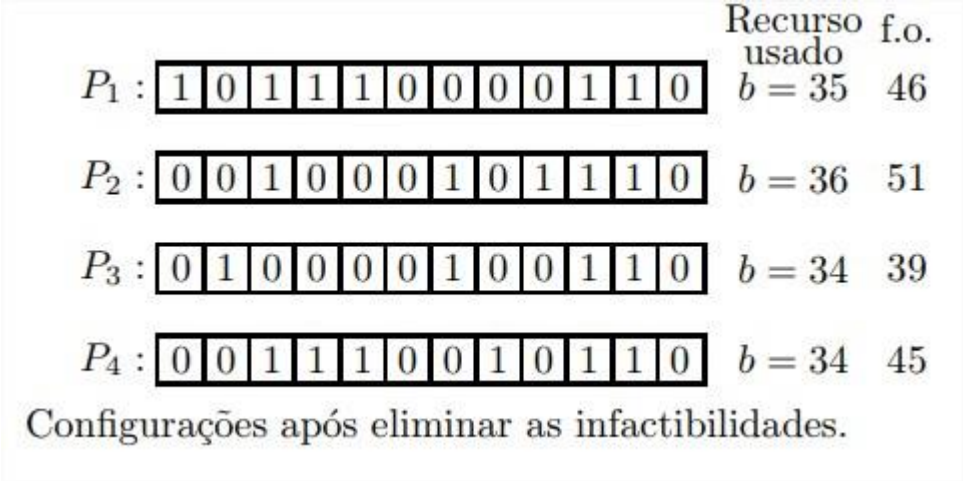
No caso particular do problema da mochila, com o tipo de codifica¸c˜ao escolhida, tanto a recombina¸c˜ao assim como a muta¸c˜ao podem gerar configura¸c˜oes infact´ıveis. Neste tipo de casos existem as seguintes alternativas:

(1) transformar as configura¸c˜oes infact´ıveis em fact´ıveis usando uma heur´ıstica r´apida;

(2) eliminar as configura¸c˜oesinfact´ıveis;

(3) considerar todas as configura¸c˜oes como sendo “fact´ıveis” e penalizar aquelas infact´ıveis na fun¸c˜ao objetivo.

Para gerar a nova popula¸c˜ao ´e escolhida a primeira alternativa e para eliminar a infactibilidade, deve-se passar uma vari´avel com valor 1 para 0 at´e retomar a factibilidade. A vari´avel que deve passar de 1 para 0 ´e aquela que tiver a menor rela¸c˜ao cj/aj . Assim, na configura¸c˜ao candidata P′ 1 passamos x2 = 1 para x2 = 0 e retomamos a factibilidade. Portanto, termina o processo de gera¸c˜ao da nova popula¸c˜ao constitu´ıda pelas seguintes configura¸c˜oes:



Quando a codifica¸c˜ao for de outro tipo, diferente da codifica¸c˜ao bin´aria ent˜ao, deve-se desenvolver estrat´egias equivalentes de muta¸c˜ao levando em conta a natureza do problema, do tipo de codifica¸c˜ao escolhida e a essˆencia da muta¸c˜ao na gen´etica natural, isto ´e, pequenas mudan¸cas no conteudo ´ gen´etico mas que produzem um gen´otipo que n˜ao existia com o consequente aparecimento de um novo fen´otipo.

4.8 Programa de Controle do Algoritmo Gen´etico

E´ o conjunto de parˆametros que define o tamanho da popula¸c˜ao, a taxa de recombina¸c˜ao e a taxa de muta¸c˜ao e que define, em forma significativa, o comportamento do algoritmo gen´etico. Este conjunto de parˆametros ´e chamado de programa de controle do algoritmo gen´etico. Valores t´ıpicos recomendados pela literatura especializada s˜ao os seguintes:

- Tamanho da popula¸c˜ao: np ∈ [30 ; 200].
- Taxa de recombina¸c˜ao: ρc ∈ [0, 5 ; 1, 0].
- Taxa de muta¸c˜ao: ρm ∈ [0, 001 ; 0, 050].

4.9 Crit´erio de Parada

Existem v´arios crit´erios de parada que podem ser implementados. Assim, pode-se parar o algoritmo gen´etico quando:

- Foi executado um numero ´ especificado de gera¸c˜oes.
- A incumbente (melhor solu¸c˜ao encontrada) assume um valor pelo menos de igual qualidade que um valor previamente especificado.
- A incumbente n˜ao melhora durante um numero ´ especificado de gera¸c˜oes.
- As configura¸c˜oes da popula¸c˜ao ficam muito homogˆeneas, isto ´e, as configura¸c˜oes s˜ao muito parecidas e praticamente j´a n˜ao existe evolu¸c˜ao.
- Usando um crit´erio que depende do tipo de problema analisado.

4.10 Gera¸c˜ao da Popula¸c˜ao Inicial

- Gerar uma popula¸c˜ao inicial usando um processo aleat´orio controlado. Esta estrat´egia ´e eficiente em muitos problemas em que se conhece as caracter´ısticas da solu¸c˜ao final. Este fato acontece, por exemplo, no problema de aloca¸c˜ao de bancos de capacitores em sistemas de distribui¸c˜ao. Nesse tipo de problema ´e conhecido que nas melhores topologias s˜ao alocados bancos num numero ´ muito reduzido de barras, tipicamente, entre 2 a 8 barras mesmo em sistemas de grande porte. Portanto, pode-se gerar uma popula¸c˜ao inicial para um sistema de 200 barras do problema antes mencionado alocando, por exemplo, bancos de capacitores em at´e 10 barras do sistema. Em outras palavras, escolhe-se um numero ´ reduzido de barras, 10 de 200 no exemplo, e o processo aleat´orio identifica aquelas barras onde devem ser alocados bancos de capacitores e o numero ´ de bancos. Um processo totalmente aleat´orio produziria configura¸c˜oes com aloca¸c˜ao de capacitores em muitas barras.
- Gerar uma popula¸c˜ao inicial usando algoritmos heur´ısticos construtivos r´apidos e eficientes. Para problemas muito complexos esta proposta ´e a melhor alternativa. A id´eia consiste em gerar configura¸c˜oes de boa qualidade mas tamb´em significativamente diferentes entre elas. Estes objetivos podem ser atingidos usando algoritmos heur´ısticos construtivos e pequenas modifica¸c˜oes nesses algoritmos.

Observa¸c˜oes importantes:

- o algoritmo gen´etico b´asico, deve-se observar que o algoritmo gen´etico imita ape nas de forma grosseira a evolu¸c˜ao e a gen´etica natural.

Continuação resumo PAA2 (slide 22)

Tabu Search

5.1 Introdu¸c˜ao

A busca tabu consiste em um conjunto de princ´ıpios (fun¸c˜oes) que, de forma integrada, permitem resolver um problema complexo da maneira inteligente.

A busca tabu difere de um algoritmo de busca local do tipo descrito acima em dois aspectos fundamentais:

1. A partir da configura¸c˜ao corrente pode-se passar para uma configura¸c˜ao para a qual a fun¸c˜ao objetivo de fato aumenta (estamos imaginando um problema de minimiza¸c˜ao).

2. O conjunto de vizinhos de x n˜ao ´e caracterizado de maneira est´atica. Assim, a busca tabu define uma nova estrutura de vizinhan¸ca, $N*(x)$ que varia dinamicamente em composi¸c˜ao e tamanho durante todo o processo de otimiza¸c˜ao. estrat´egia permite a busca tabu realizar uma busca eficiente e inteligente. Assim, $N*(x)$ pode ser modificado de v´arias formas:

• Usando uma lista tabu que armazena atributos de configura¸c˜oes consideradas tabu (proibidas). Neste caso $N*(x) \subset N(x)$ pois alguns vizinhos definidos pela estrutura de vizinhan¸ca e cujos atributos fazem parte da lista tabu est˜ao proibidos. Esta estrat´egia evita retornar as` configura¸c˜oes j´a visitadas evitando assim a ciclagem.

• Usando estrat´egias para diminuir a vizinhan¸ca ou a lista de configura¸c˜oes candidatas. Geralmente o numero ´ de configura¸c˜oes $x' \in N(x)$ pode ser muito grande e avaliar a fun¸c˜ao objetivo de todas essas configura¸c˜oes, para encontrar aquela que apresenta melhor desempenho, pode requerer elevado esfor¸co computacional. A busca tabu disp˜oe de pelo menos quatro m´etodos diferentes para encontrar uma vizinhan¸ca de tamanho reduzido tal que $N*(x) \subset N(x)$.

• Usando configura¸c˜oes de elite e path relinking para caracterizar e encontrar novas configura¸c˜oes candidatas. Esta estrat´egia visa encontrar novas configura¸c˜oes de alta qualidade que dificilmente seriam encontradas a partir da defini¸c˜ao de $N*(x)$. E´ f´acil verificar que neste caso a rela¸c˜ao $N*(x) \subset N(x)$ j´a n˜ao ´e mais verdadeira.

• Redefinir o conjunto N(x) durante o processo de otimiza¸c˜ao. Geralmente esta proposta pode ser implementada aproveitando as caracter´ısticas espec´ıficas do problema.

Em rela¸c˜ao aos conceitos b´asicos, tr^es aspectos devem ser analisados em detalhe para uma melhor compreens˜ao dos aspectos b´asicos do algoritmo:

(1) a forma de realizar as transi¸c˜oes no espa¸co de busca;

(2) as caracter´ısticas de uma configura¸c˜ao ou elemento do espa¸co de busca;

(3) as caracter´ısticas das configura¸c˜oes vizinhas da configura¸c˜ao corrente.

Tabu search com Mem´oria de Curto Prazo - Tabu Search Recency Based

O algoritmo TS mais elementar ´e o chamado algoritmo TS com mem´oria de curto prazo que usa uma lista de atributos proibidos e um crit´erio de aspira¸c˜ao. Neste algoritmo, o processo de busca ´e iniciada de uma configura¸c˜ao inicial (que pode ser obtida aleatoriamente) e realiza um numero ´ determinado de transi¸c˜oes at´e satisfazer um crit´erio de parada.

a. Em cada passo este algoritmo TS analisa um conjunto de vizinhos (todos os vizinhos ou um conjunto reduzido deles) e passa para a melhor configura¸c˜ao vizinha n˜ao proibida ou que estando proibida satisfaz o crit´erio de aspira¸c˜ao.

Essa configura¸c˜ao vizinha se transforma na nova configura¸c˜ao corrente.

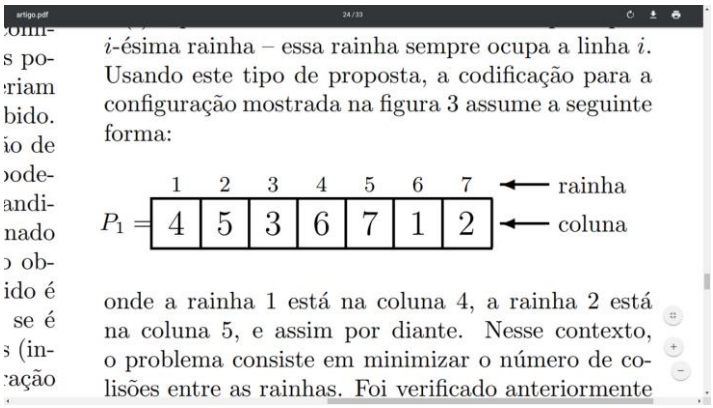
O car´ater agressivo da estrat´egia TS de passar sempre para a melhor configura¸c˜ao ou menos pior permite a TS sair de eventuais otimos ´ locais mas tamb´em exige a presen¸ca de uma lista de transi¸c˜oes proibidas para evitar retornar a uma configura¸c˜ao j´a visitada.

A lista tabu pode armazenar configura¸c˜oes completas (que exigiria muita mem´oria para armazenamento) ou melhor ainda se pode armazenar atributos de configura¸c˜oes visitadas, que ´e a estrat´egia mais adequada em problemas reais.

ravelmente os requerimentos de mem´oria mas tamb´em acrescenta um problema: o atributo proibido, que corresponde a uma configura¸c˜ao j´a visitada, pode ser compartilhado por outras configura¸c˜oes vizinhas candidatas algumas das quais podem ser altamente atrativas mas que n˜ao poderiam ser visitadas porque possuem um atributo proibido. Este problema ´e contornado usando uma fun¸c˜ao de TS chamado crit´erio de aspira¸c˜ao.

Exemplo:

5.4.1 Codifica¸c˜ao do Problema



i-ésima rainha – essa rainha sempre ocupa a linha i. Usando este tipo de proposta, a codificação para a configuração mostrada na figura 3 assume a seguinte forma:

	1	2	3	4	5	6	7	
								← rainha
$P_1 =$	4	5	3	6	7	1	2	← coluna

onde a rainha 1 está na coluna 4, a rainha 2 está na coluna 5, e assim por diante. Nesse contexto, o problema consiste em minimizar o número de colisões entre as rainhas. Foi verificado anteriormente

Nesse contexto, o problema consiste em minimizar o numero ´ de colis˜oes entre as rainhas. Foi verificado anteriormente que nesse tipo de codifica¸c˜ao aparece apenas ataques na diagonal. Portanto, para encontrar o valor da fun¸cc˜ao objetivo, o numero ´ de colis˜oes, deve-se apenas percorrer as diagonais do tabuleiro. Esse trabalho pode ser realizado de forma eficiente usando a propriedade de diagonais positivas e negativas no tabuleiro que permite implementar de forma eficiente o c´alculo da fun¸c˜ao objetivo no computador [24].

Para o problema das n rainhas, um algoritmo TS com mem´oria de curto prazo assume a seguinte forma:
1. Escolher o crit´erio de aspira¸c˜ao e a dura¸c˜ao da proibi¸c˜ao dos atributos.
2. Definir a estrutura de vizinhan¸ca. 3. Escolher a topologia inicial.
4. Gerar todos os vizinhos, calcular a fun¸c˜ao objetivo e ordenar esses vizinhos colocando primeiro o de melhor qualidade.
5. Passar para o melhor vizinho desde que n˜ao se encontre proibido ou estando proibido cumpra com o crit´erio de aspira¸c˜ao.
6. Atualizar a lista de atributos proibidos.
7. Repetir os passos 4 a 6 at´e encontrar uma topologia com fun¸c˜ao objetivo zero, isto ´e, sem colis˜oes.

O problema de n = 7 rainhas pode ser facilmente resolvido usando os seguintes crit´erios [24]:
(1) dura¸c˜ao da proibi¸c˜ao: 3 transi¸c˜oes;
(2) crit´erio de aspira¸c˜ao: eliminar a proibi¸c˜ao se aparece uma configura¸c˜ao candidata melhor que a incumbente;
(3) caracteriza¸c˜ao da vizinhan¸ca: ´e vizinho da configura¸c˜ao corrente toda configura¸c˜ao encontrada trocando as colunas de duas rainhas, isto ´e, existem 21 vizinhos;
(4) escolhendo a topologia inicial mostrada na figura 3.

5.5 Componentes Avan¸cadas de Tabu Search

Existem outros componentes do algoritmo tabu search que podem ser usados em algoritmos mais sofisticados. Os principais componentes s˜ao os seguintes:
(1) t´ecnica de redu¸c˜ao de vizinhan¸ca;
(2) uso de outros tipos de mem´oria;
(3) uso de intensifica¸c˜ao e diversifica¸c˜ao;
(4) uso de path relinking, (5) oscila¸c˜ao estrat´egica;
(5) uso de topologias de elite. Comentamos brevemente esses componentes.

6 Sistema de Colˆonia de Formigas(ACS)

As formigas conseguem decidir o caminho mais curto até o alimento, mesmo que um obstáculo apareça, com isso, todas elas seguem uma trilha até o alimento. O que ocorre é que as formigas seguem o caminho com mais feromônio depositado, isto é, a primeira formiga a fazer a trilha solta seu feromônio, a segunda a escolher essa trilha faz o mesmo e assim por diante, a intensidade do feromônio chega a um ponto que todas as formigas acabam seguindo o mesmo caminho.

6.3 Sistema de Colˆonia de Formigas(ACS) - Algoritmo B´asico

Um algoritmo geral da colˆonia de formigas ´e estabelecido a seguir [28, 29].

• Inicializa¸c˜ao: Nesta fase todas as formigas s˜ao posicionadas nos n´os de partida gerados aleatoriamente e os valores iniciais para a intensidade do cheiro sobre a trilhas s˜ao atribu´ıdos. O ACS considera o espa¸co de busca atrav´es de multiplos ´ est´agios. Em cada est´agio s˜ao consideradas as solu¸c˜oes fact´ıveis para o problema sob an´alise.

• Fase de Despacho das Formigas: Cada formiga escolhe que caminho seguir, levando em considera¸c˜ao a intensidade do cheiro do caminho e uma fun¸c˜ao de m´erito pr´e estabelecida, do problema sob estudo, que deve estar abstra´ıda com uma rela¸c˜ao com a distˆancia. As formigas preferem se mover para estados que est˜ao conectados por arestas mais

curtas, ou com uma grande quantidade de feromˆonio. Este processo ´e repetido at´e que as formigas completem um tur.

• Avalia¸c˜ao ou fitness: Depois que todas as formigas completarem um tur, o valor de fitness de cada uma tem que ser calculado. Algumas fun¸c˜oes de fitness dos problemas de otimiza¸c˜ao podem ser usadas para calcular o desempenho das formigas. Qualquer restri¸c˜ao associada com o problema de otimiza¸c˜ao pode ser incorporada na fun¸c˜ao objetivo como termos de penalidades, para especificar a fun¸c˜ao de fitness. Estes valores de fitness s˜ao ent˜ao usados para atualizar a intensidade de feromˆonio dos caminhos em cada est´agio.

• Atualiza¸c˜ao da Intensidade do Cheiro: Dois fatores afetar˜ao a intensidade do cheiro de cada aresta. A intensidade de feromˆonio de cada aresta ir´a evaporar depois de um certo tempo (ou seja, o cheiro de feromˆonio perde a intensidade se outras formigas n˜ao passarem pelo caminho e depositarem mais feromˆonio). Para aqueles camimhos que as formigas passaram pela itera¸c˜ao atual, a intensidade de feromˆonio pode ser atualizada atrav´es de uma regra de transi¸c˜ao de estados.

• Convergˆencia: O processo iterativo termina se foi alcan¸cado o numero ´ m´aximo de itera¸c˜oes pr´e estabelecidos ou se todas as formigas escolherem o mesmo tur.

6.4 Algoritmo ACS - Problema do Caixeiro Viajante(PCV)

Inicializa¸c˜ao No tempo 0 as formigas s˜ao posicionadas em diferentes cidades e valores iniciais para a intensidade do cheiro τij (0) s˜ao atribu´ıdos para cada uma das arestas. O primeiro elemento da lista tabu de cada formiga ´e ajustado como sendo a cidade inicial. Fa¸ca: - t = 0 (t ´e o contador de tempo) - NC = 0 (NC ´e o contador de ciclos) - Para toda aresta (i, j) adote um valor inicial τij (t) = c e posicione as m formigas nos n n´os. Passo Principal

1. Toda formiga deve se movimentar da cidade i para a cidade j, escolhendo a cidade para se movimentar com uma probabilidade que ´e fun¸c˜ao de α e β. S = 1 (S ´e o ´ındice da lista tabu) - Para k = 1 at´e m fa¸ca: Insira a cidade de partida da k-´esima formiga na lista tabuk(s).

2. Repita este movimento das formigas de cada uma das cidades i para a cidades j, at´e que a lista tabu esteja cheia. Este processo deve ser repetido (n − 1) vezes, ou seja: - Para S = S + 1 fa¸ca, Para k = 1, m fa¸ca, Escolha a cidade j para se movimentar com uma probabilidade pk ij (t)

(No tempo t a k-´esima formiga est´a na cidade i = tabuk(s − 1)) - Movimente a k-´esima formiga para a cidade j - Insira a cidade j na lista tabuk(s).

3. Para k = 1, m fa¸ca, - Analise o tour da k-´esima formiga atrav´es dos movimentos registrados nas listas tabuk. - Calcule o comprimento Lk do tour percorrido pela k-´esima formiga. - Atualize o tour mais curto encontrado. - Para toda aresta (i, j), - Para k = 1, m fa¸ca; Δτk ij (t) = $\begin{cases} Q \\ Lk \end{cases}$ se (i, j) ∈ tour descrito por tabuk 0 caso contr´ario Δτij = Δτij + Δτk ij

4. Para toda aresta (i, j) calcule τ t+n ij segundo a equa¸c˜ao τ t+n ij = ρτt ij + Δτij Fa¸ca, t = t + n NC = NC + 1 Δτij = 0 para toda aresta (i, j)

5. Se NC < NCmax e sem comportamento de estagna¸c˜ao, fa¸ca, - Esvazie todas as listas tabu, - Volte ao passo (2); Caso contr´ario, - Imprima o menor tur, PARE.