Estudo de Métodos Estocásticos para Otimização Global de Problemas de Programação Não Linear

Érico Murilo Gozzi

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título de mestre
em
ciências

Área de Concentração: Ciência da Computação

Orientador: Prof. Dr. Ernesto G. Birgin

São Paulo, janeiro de 2007.

Estudo de Métodos Estocásticos para Otimização Global de Problemas de Programação Não Linear

Este exemplar corresponde à redação final
da dissertação de mestrado devidamente
corrigida e defendida por
Érico Murilo Gozzi
e aprovada pela comissão julgadora.

Banca examinadora:

- Prof. Dr. Ernesto G. Birgin (Orientador) - IME-USP
- Prof. Dr. Marcelo Gomes de Queiroz - IME-USP
- Prof. Dr. Roberto Andreani - IMECC-UNICAMP

Sumário

Resumo

O estudo de técnicas de otimização global encontra aplicação em uma grande diversidade de problemas, tais como: economia de escala, finanças, análise de risco, estatística, redes e problemas de transporte, banco de dados, biologia molecular, genética. No entanto, apesar de mais de quatro décadas de pesquisas nesta área, as soluções conhecidas para problemas de otimização global são aplicáveis, de forma eficiente, apenas a um conjunto restrito de problemas.

Os algoritmos de otimização global podem ser divididos em dois grandes grupos: determinísticos e estocásticos. Os métodos determinísticos garantem alcançar uma boa aproximação de um minimizador global em um número finito de passos; para os métodos estocásticos a probabilidade de encontrar um minimizador global tende a 1 quando o número de iterações tende a infinito.

O objetivo deste trabalho foi estudar três algoritmos estocásticos de otimização global: Random Linkage, Tunneling utilizando a curva de Lissajous e o GRASP adaptado para problemas de programação não linear. Para isso, implementamos os três métodos e realizamos experimentações com problemas clássicos de programação não linear. Em seguida, adaptamos os métodos para utilizar o algoritmo GENCAN como método de minimização local. Desta forma foi possível estabelecer um comparativo da eficácia de cada um dos métodos em escapar de minimizadores locais.

Abstract

The study of global optimization techniques finds application in a range sort of problems, such that: scale economy, finances, risk analysis, statistics, network and transportation problems, databases, molecular biology, genetics. However, in despite of more than four decades of research in this area, the known solutions for global optimization problems are applicable, in a efficient form, only to a few set of problems.

Global optimization algorithms can be divided in two major groups: deterministics and stochastics. The determistics methods guarantee to achieve a good approximation of the global optimizer in a finite number of steps; in the stochastics methods the probability of finding a good approximation of the global optimizer tends to 1 (one) when the number of steps tends to infinity.

The objective of this work was to study three stochastics global optimization algorithms: Random Linkage, Tunneling using the Lissajous curve and GRASP adapted for solving nonlinear optimization problems. We implemented the three methods and realized several computational experiments with classic nonlinear problems. Then, we adapted the methods to use GENCAN as the local minimization method. In such a way it was possible to compare the effectiveness of each method in scape of local minimizers.

Capítulo 1

Introdução

O estudo de técnicas de otimização global encontra aplicação em uma grande diversidade de problemas, tais como: economia de escala, finanças, análise de risco, estatística, redes e problemas de transporte, banco de dados, biologia molecular, genética. No entanto, apesar de mais de quatro décadas de pesquisas nesta área, as soluções conhecidas para problemas de otimização global são aplicáveis, de forma eficiente, apenas a um conjunto restrito de problemas.

O problema considerado neste trabalho consiste na minimização de uma função $f : \Re^n \rightarrow \Re$ suave e com limitantes nas variáveis. O conjunto factível Ω é definido por $\Omega = \{x \in \Re^n \mid l \leq x \leq u\}$. Um ponto x^* é uma solução global para o problema em questão, se e somente se, $f(x^*) \leq f(x)\ \forall x \in \Omega$. Este ponto é chamado de minimizador global de f, sendo $f(x^*)$ o mínimo global do problema.

Uma das primeiras estratégias sugeridas para a resolução de problemas de otimização global [9] consistiu em localizar e comparar o maior número possível de minimizadores locais do problema. Desta estratégia surgiu o método conhecido como *Multistart* onde, a partir de pontos iniciais aleatórios, eram utilizados métodos de minimização local visando descobrir o maior número possível de soluções locais. Com isto, este método atinge melhores aproximações do minimizador global a medida em que novas soluções locais são descobertas.

Uma outra estratégia contemporânea ao *Multistart* surgiu do princípio de que a *aproximação* do minimizador global não precisaria ser um minimizador local. Seguindo esta prerrogativa foi criado um método de "força bruta" conhecido como Busca em Malha (*Grid search method*) que consiste em mapear Ω em pontos equidistantes utilizando um passo fixo h. A partir deste mapeamento, o ponto com o menor valor de função é devolvido pelo algoritmo como sendo a aproximação da solução global do problema.

Destes dois exemplos é possível dividir os métodos de otimização global em dois grandes grupos [9, 21]:

- Estocásticos, cuja probabilidade de convergirem para uma solução global tende a 1 quando o número de passos do algoritmo tende a infinito, como no caso do *Multistart*.
- Determinísticos, que garantem alcançar uma ϵ-vizinhança de uma solução global em um

número finito de passos, como no caso da Busca em Malha.

A medida em que há um crescimento na complexidade e no porte dos problemas a serem resolvidos, o uso de algoritmos determinísticos torna-se proibitivo devido a quantidade de processamento necessário. Salvo casos de algoritmos determinísticos específicos, tais como os utilizados em programação côncava e métodos DC (desigualdade entre duas funções convexas), a melhor estratégia a ser seguida é a utilização de métodos estocásticos.

Os algoritmos estocásticos procuram, a cada passo, melhorar a sua aproximação da solução ótima de forma que, em um número infinito de passos, a probabilidade de encontrarem uma boa aproximação do ótimo global tende a 1.

Dentre as diversas estratégias utilizadas pelos Métodos Estocásticos, destacamos os Métodos de Duas Fases [7] que consistem em adaptar os métodos de otimização local para resolverem problemas de otimização global. Estes métodos são compostos por uma mistura de duas etapas:

1. Etapa de Exploração, ou Fase Global, cujo objetivo é encontrar pontos em diferentes regiões de Ω na tentativa de cobrir as diversas regiões de atração dos minimizadores locais do problema de otimização.

2. Etapa de Refinamento, ou Fase Local, onde é realizada uma exploração local em busca de uma boa aproximação do minimizador global do problema.

Após sucessivas execuções destas duas etapas, a melhor solução encontrada pela Etapa de Refinamento é considerada pelo algoritmo como a aproximação da solução ótima do problema proposto.

O grau de conhecimento dos elementos que compõem o problema a ser resolvido tem um impacto considerável nas estratégias seguidas pelos algoritmos. O objetivo deste trabalho é implementar e comparar três algoritmos estocásticos aplicados a problemas de programação não linear (PNL) com restrições de caixa. Os algoritmos apresentados nos próximos capítulos, trabalham com níveis diferentes de informação sobre a função objetivo e a região factível.

No Capítulo 2 descrevemos o método *Random Linkage*, apresentado em [19]. Esse método foi idealizado a partir de melhorias realizadas no algoritmo proposto em [24, 25], que utiliza-se da estratégia de clusterização da região factível com intuito de diminuir a quantidade de minimizações locais desnecessárias. Mostramos inicialmente o método original, proposto em [24, 25], e a seguir analisamos e implementamos as mudanças descritas em [19].

No Capítulo 3 detalhamos a técnica de *Tunneling*, descrita em [17, 16], e apresentamos uma implementação do algoritmo proposto em [23] que utiliza a curva de *Lissajous* na etapa de *Tunneling*. Descrevemos ainda a nossa implementação do algoritmo proposto em [23] e apresentamos a aplicação do mesmo em um problema de empacotamento de círculos.

No Capítulo 4 apresentamos o método GRASP aplicado a problemas de programação não linear com restrições de caixa. O GRASP, originalmente, proposto em [11, 12] para problemas de otimização combinatória, foi adaptado em [20] para problemas de otimização contínua e melhorado em [14, 15]. Para análise deste método, implementamos os algoritmos propostos em [20] e [14, 15].

No Capítulo 5 mostramos experimentos com os métodos descritos nos capítulos anteriores para um conjunto de 52 testes encontrados na literatura clássica de programação não linear. A partir dos resultados obtidos neste capítulo realizamos, no Capítulo 6, experimentos em problemas de empacotamento com os 3 métodos que obtiveram os melhores resultados nos testes gerais.

Concluímos este trabalho com uma análise geral do desempenho dos algoritmos considerando todos os experimentos realizados nos capítulos 5 e 6.

Capítulo 2

O algoritmo *Random Linkage*

2.1 Introdução

Os algoritmos estocásticos utilizados para a minimização de uma função f qualquer, têm como característica comum encontrar uma boa aproximação do mínimo global com probabilidade 1 quando o número de iterações do método tende a infinito. Com o intuito de explorarmos melhor esta característica consideremos um procedimento cuja única ação seja gerar pontos aleatórios factíveis em Ω e que obedeçam a uma distribuição uniforme. Queremos que este procedimento encontre um ponto $y \in B_\Omega(\epsilon, x^*)$ tal que $B_\Omega(\epsilon, x^*) = \{y \in \Omega \mid \|y - x^*\| \leq \epsilon, \text{ para } \epsilon > 0\}$, onde $x^* \in \Omega$ é um minimizador global.

A probabilidade de que até a K-ésima iteração tenha sido encontrado pelo menos um ponto em $B_\Omega(\epsilon, x^*)$ é

$$\begin{gathered} Pr(x_k \in B_\Omega(\epsilon, x^*) \text{ para } 1 \leq k \leq K) = \\ 1 - Pr(x_1, \ldots, x_K \notin B_\Omega(\epsilon, x^*)) = 1 - (1 - \varphi(B_\Omega(\epsilon, x^*)))^K, \end{gathered}$$

onde $\varphi(B_\Omega(\epsilon, x^*))$ corresponde à probabilidade do ponto ser sorteado nas regiões de Ω. Como $\varphi(B_\Omega(\epsilon, x^*)) > 0$, então a probabilidade acima tende a 1 quando K tende a ∞. Portanto, o procedimento acima apresenta características de um Método Estocástico.

Deste exemplo, deriva o algoritmo de otimização global conhecido como *Pure Random Search*, que utiliza-se apenas da Etapa de Exploração para a geração das possíveis soluções do problema de otimização.

Algoritmo 2.1. Pure Random Search

Dado $K \in \mathbb{N}$

Passo 0. $f_{\min} \leftarrow \infty$; $k \leftarrow 1$

Passo 1. Gere um ponto $x_k \in \Omega$ com distribuição uniforme

Se $f(x_k) < f_{\min}$ então

$f_{\min} \leftarrow f(x_k)$

$y \leftarrow x_k$

fim se

Passo 2. $k \leftarrow k+1$

Se $k < K$ volte ao Passo 1

O ponto y é retornado como aproximação de um minimizador global de f.

No entanto, este primeiro algoritmo não é considerado um Método de Duas Fases. Isto porque é formado apenas pela Etapa de Construção, onde são gerados os pontos x_k. Visando aumentar a eficiência do algoritmo foi incluída uma Fase de Refinamento composta por um procedimento de minimização local P que, a partir de um ponto inicial x_k, retorna um minimizador local $\tilde{x}_k$ (referenciaremos a execução deste procedimento utilizando a seguinte notação: $\tilde{x}_k \leftarrow P(x_k)$). Este algoritmo[1], conhecido como *Multistart*, encontra-se descrito a seguir:

Algoritmo 2.2. Multistart

Dado $K \in \mathbb{N}$

Passo 0. $f_{\min} \leftarrow \infty$; $k \leftarrow 1$

Passo 1. Gere um ponto $x_k \in \Omega$ com distribuição uniforme

Passo 2. $\tilde{x}_k \leftarrow P(x_k)$

Se $f(\tilde{x}_k) < f_{\min}$ então

$f_{\min} \leftarrow f(\tilde{x}_k)$

$y \leftarrow \tilde{x}_k$

fim se

Passo 3. $k \leftarrow k+1$

Se $k < K$ volte ao Passo 1

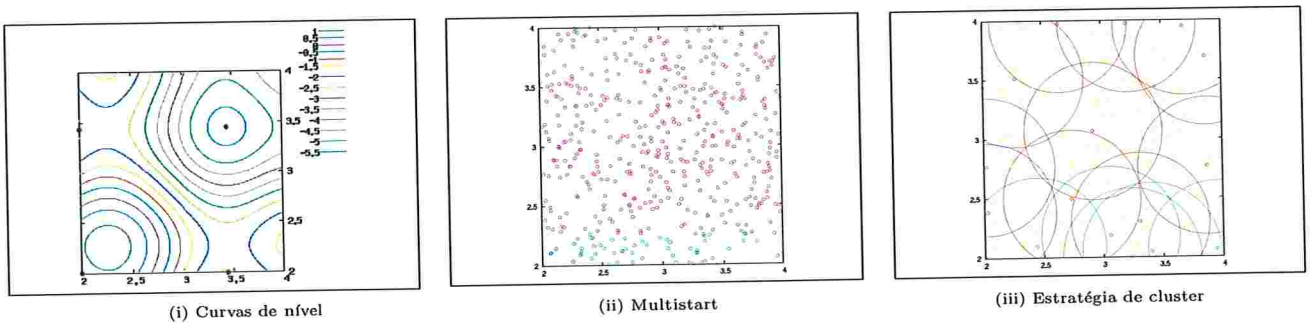
O ponto y é retornado como aproximação de um minimizador global de f.

Apesar deste método mostrar-se mais atrativo que o *Pure Random Search*, ainda assim apresenta baixa eficiência. Isto deve-se, principalmente, ao fato de que o mesmo minimizador local pode ser localizado várias vezes. Definindo a região de atração de um minimizador local $\tilde{x}$ como $R(\tilde{x}) = \{x \in \Omega \mid \tilde{x} \leftarrow P(x)\}$, o ideal é que P seja executado apenas uma vez para cada região de atração existente em Ω. Visando resolver este problema foram propostas adaptações ao *Multistart* com a utilização de estratégias de clusterização da região factível.

[1] Os fluxogramas de todos os algoritmos que foram implementados e testados encontram-se descritos no Apêndice B

2.2 Métodos de clusterização

A idéia básica dos algoritmos de clusterização é particionar o conjunto factível Ω em diversas regiões de atração e executar P uma única vez em cada uma destas regiões. A Figura (2.1) mostra uma comparação do *Multistart* com uma possível estratégia de clustering aplicados a função $f(x,y) = \sin(x) + \sin\left(\frac{10x}{3}\right) + \log(x) - 0.84x + \sin(y) + \sin\left(\frac{10y}{3}\right) + \log(y) - 0.84y$. Em (i) observamos, através das curvas de nível da função, a existência dos minimizadores locais $\{(3.4, 3.4)^t, (3.4, 2.0)^t, (2.0, 2.0)^t, (2.0, 3.4)^t\}$. Em (ii), mostramos a aplicação do método *Multistart* a 500 pontos iniciais distintos, onde cada cor refere-se aos pontos iniciais que fizeram o método de minimização local convergir ao mesmo minimizador local. Em (iii) mostramos a aplicação de uma estratégia de clustering a estes mesmos 500 pontos iniciais, indicando com círculos coloridos os clusters que foram formados e a qual minimizador local convergiram. É válido ainda mencionar que, enquanto o *Multistart* realizou 500 minimizações locais, o método de clustering realizou apenas 16, descartando os demais pontos iniciais (referenciados em amarelo).



(i) Curvas de nível (ii) Multistart (iii) Estratégia de cluster

Figura 2.1: Comparação entre o *Multistart* e uma estratégia de clustering utilizando $f(x,y) = \sin(x) + \sin\left(\frac{10x}{3}\right) + \log(x) - 0.84x + \sin(y) + \sin\left(\frac{10y}{3}\right) + \log(y) - 0.84y$ como função objetivo.

O algoritmo apresentado nesta seção, proposto em [24] com o nome de *Density Clustering*, restringe a execução do procedimento de minimização local P para pontos muito próximos de um minimizador local.

O ponto de partida do método consiste em montar um conjunto S de N pontos aleatórios em Ω, ordenados em ordem crescente de valor da função objetivo. Em seguida, para cada ponto $x_i \in S$, é verificado se existe no conjunto $\tilde{X}$ de minimizadores locais conhecidos um ponto $\tilde{x} \in \tilde{X}$ tal que a distância entre ambos é menor ou igual a uma distância crítica r. Caso não exista tal $\tilde{x}$, o procedimento P é executado utilizando x_i como ponto inicial.

Desta forma, a distância crítica r é responsável pela definição de quais pontos se encontram na mesma região de atração $R(\tilde{x})$. Após todos os pontos de S terem sido analisados, o algoritmo acrescenta N novos pontos em S, reordenando S em ordem crescente e repetindo todo o processo até que um número máximo de K iterações seja atingido.

Visando proporcionar que os clusters alcancem, a cada nova iteração, um espaço maior da

região factível, foi proposto em [24] a utilização do seguinte valor para a distância crítica r em uma iteração k

$$r_k = \pi^{-\frac{1}{2}} \left(k\, \Gamma\left(1 + \frac{n}{2}\right) \mu(\Omega) \frac{\sigma log kN}{kN} \right)^{\frac{1}{n}}, \tag{2.1}$$

onde $\Gamma(\cdot)$ corresponde à função Gama[2] e $\mu(\cdot)$ a medida de Lebesgue.

O algoritmo descrito a seguir utiliza ainda uma constante $\gamma \in (0, 1]$ para criar um subconjunto $S_R \subseteq S$. S_R conterá os pontos em S com menor valor da função objetivo.

Algoritmo 2.3 Density Clustering

Dados $K \in \mathbb{N}$, $N \in \mathbb{N}$, $\tilde{X} = \emptyset$, $S = \emptyset$ e $\gamma \in (0, 1]$

Passo 0. $k \leftarrow 1$

Passo 1. Calcule r_k de acordo com (2.1)

Passo 2. Inclua em S N pontos aleatórios de Ω com distribuição uniforme. Ordene os elementos de S em ordem crescente de valor da função objetivo

Determine um conjunto reduzido S_R com os γkN primeiros elementos de S

Passo 3. Se $S_R = \emptyset$ vá para o Passo 6. Senão selecione $x_i \in S_R$ e faça $S_R \leftarrow S_R - \{x_i\}$

Passo 4. Se existe $\tilde{x} \in \tilde{X}$ tal que $\|x_i - \tilde{x}\| \leq r_k$ então volte para o Passo 3

Passo 5. $\tilde{x} \leftarrow P(x_i)$

$\tilde{X} \leftarrow \tilde{X} \cup \{\tilde{x}\}$

Passo 6. $k \leftarrow k + 1$

Se $k < K$ volte ao Passo 1

O ponto $y = argmin\{f(\tilde{x}) \mid \tilde{x} \in X^*\}$ é retornado como aproximação de um minimizador global de f.

Este algoritmo, embora teoricamente mais eficaz que o *Multistart*, apresenta desempenho irregular para problemas com funções objetivo de grau muito maior que 2 ou que não sejam polinomiais. O principal motivo para esta ineficiência deve-se ao fato do algoritmo formar os clusters apenas a partir dos minimizadores locais. Com isto, os clusters formados pelo algoritmo

[2] A função Gama é definida como [13]:

$$\Gamma(\alpha) = \int_0^\infty e^{-t} t^{\alpha-1} dt$$

têm sempre o formato de um hipercírculo o que nem sempre é verificado para as regiões de atração dos minimizadores locais de f.

Com o intuito de resolver este problema, foi proposto o algoritmo *Single Linkage*. Este algoritmo foi criado como uma melhoria do *Density Clustering*. A principal alteração está relacionada a forma em que clusters são criados e atualizados. Para decidir se um dado ponto x_k pertence a algum cluster, é verificado se existe algum ponto $x \in S \cup \tilde{X}$ tal que $\|x_k - x\| \leq r_k$. Caso exista tal ponto, x_k é incluído no mesmo cluster de x; do contrário um novo cluster é criado sendo incluído nele x_k junto com o minimizador local $P(x_k)$.

Para o cálculo da distância crítica, desejamos reduzir as chances de que $y \in B_\epsilon(\Omega)$ não seja encontrado pelo algoritmo. Para isto, queremos que a probabilidade α de que, em uma iteração k, y ainda não tenha sido encontrado diminua a cada iteração. Supondo que um minimizador global x^* esteja dentro de um cluster hipotético C, a probabilidade de que, dados kN pontos iniciais, nenhum deles pertença a C é

$$\alpha = \left(1 - \frac{\mu(C)}{\mu(\Omega)}\right)^{kN},$$

onde $\mu(C)$ e $\mu(\Omega)$ são as medidas de Lebesgue para o cluster C e Ω, respectivamente.

Visando evitar que o algoritmo termine incorretamente, ou seja que y não tenha sido encontrado, definiremos um valor para $\mu(C)$, e conseqüentemente r, de tal forma que α diminua a cada iteração e que o valor de $\mu(C)$ seja o menor possível. Desta forma, para um certo $\sigma > 0$, escolhemos $\mu(C) = \frac{\mu(\Omega)\sigma log(kN)}{kN}$, conforme definido em [24].

Antes de avançarmos na definição de r, precisamos definir o cluster C corretamente. Para isto, considere que a região de atração de um minimizador local $\tilde{x}$ é dada por um subconjunto C do conjunto de nível

$$L(r) = \{x \mid x \in \Omega,\ f(x) \leq r\},$$

para algum $r \geq f(\tilde{x})$. Se considerarmos uma aproximação quadrática de $f(x)$ ao redor de $\tilde{x}$ podemos definir C como

$$C = \{x \in \Omega \mid f(\tilde{x}) + \frac{1}{2}(x - \tilde{x})^T B(x - \tilde{x}) \leq r\},$$

onde B é uma aproximação de $\nabla^2 f(\tilde{x})$.

Como C é um hipercírculo, então o sua medida de Lebesgue corresponde a $\frac{\pi^{\frac{n}{2}} r^n}{\Gamma(1+\frac{n}{2})|B|^{\frac{1}{2}}}$.

Temos então que

$$\mu(C) = \frac{\mu(\Omega)\sigma log(kN)}{kN} = \frac{\pi^{\frac{n}{2}} r^n}{\Gamma\left(1 + \frac{n}{2}\right)|B|^{\frac{1}{2}}}$$

implica

$$r = \frac{\left[\Gamma\left(1 + \frac{n}{2}\right)|B|^{\frac{1}{2}} \mu(\Omega)\sigma \frac{log(kN)}{kN}\right]^{\frac{1}{n}}}{\sqrt{\pi}}.$$

Tomando $B = I$ temos

$$r = \frac{1}{\sqrt{\pi}} \left[\Gamma\left(1 + \frac{n}{2}\right) \mu(\Omega) \sigma \frac{log(kN)}{kN} \right]^{\frac{1}{n}}.$$

Portanto, o valor da distância crítica utilizada pelo Single Linkage é

$$r_k = \pi^{-\frac{1}{2}} \left(\Gamma\left(1 + \frac{n}{2}\right) \mu(\Omega)\, \sigma\, \frac{logkN}{kN}. \right)^{\frac{1}{n}}, \tag{2.2}$$

onde $\mu(\cdot)$ representa a medida de Lebesgue, $\Gamma(\cdot)$ a função Gama e $\sigma > 0$.

O algoritmo descrito a seguir utiliza ainda uma constante $\gamma \in (0, 1]$ para criar um subconjunto S_R de S. Com isto, é possível restringir em S_R os pontos que apresentam os menores valores da função objetivo.

Algoritmo 2.4. Single Linkage

Dados $K \in \mathbb{N}$, $N \in \mathbb{N}$, $X = \emptyset$, $S = \emptyset$ e $\gamma \in (0, 1]$

Passo 0. $k \leftarrow 1$

Passo 1. Calcule r_k de acordo com (2.2)

Passo 2. Inclua em S N pontos aleatórios de Ω com distribuição uniforme. Ordene os elementos de S em ordem crescente de valor da função objetivo

Determine um conjunto reduzido S_R com os γkN primeiros elementos de S

Passo 3. Se $S_R = \emptyset$ então vá para o Passo 6. Senão selecione $x_i \in S_R$ e faça $S_R \leftarrow S_R - \{x_i\}$

Passo 4. Se existe $x_j \in S_R \cup \tilde{X}$ tal que $\|x_i - x_j\| \leq r_k$ e $f(x_i) > f(x_j)$ então volte para o Passo 3

Passo 5. $\tilde{x} \leftarrow P(x_i)$

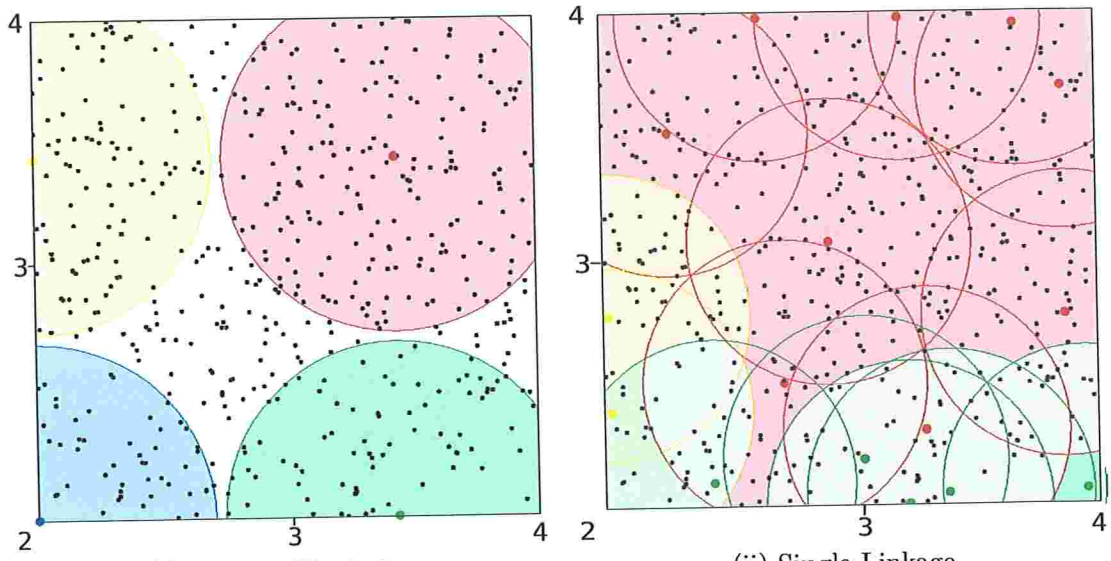
$\tilde{X} \leftarrow \tilde{X} \cup \{\tilde{x}\}$

Passo 6. $k \leftarrow k + 1$

Se $k < K$ volte ao Passo 1

O ponto $y = argmin\{f(\tilde{x}) \mid \tilde{x} \in X^*\}$ é retornado pelo algoritmo como aproximação de um minimizador global de f.

A Figura (2.2) mostra um exemplo da aplicação do *Density Clustering* e do *Single Linkage* utilizando a função $f(x, y) = \sin(x) + \sin\left(\frac{10x}{3}\right) + \log(x) - 0.84x + \sin(y) + \sin\left(\frac{10y}{3}\right) + \log(y) - 0.84y$. Neste exemplo, após a geração de 500 pontos, o *Density Clustering* identificou 4 clusters distintos, cada qual referente a uma região de atração. Para o mesmo número de pontos, o *Single Linkage* encontrou 16 clusters diferentes. Os pontos pretos dentro dos círculos coloridos representam os pontos gerados e que foram descartados, ou seja, que não foram utilizados como pontos iniciais pelo procedimento de minimização local. A partir desta figura é possível notar que a quantidade de pontos descartados no *Single Linkage* é bem maior que a do *Density Clustering*.



(i) Density Clustering (ii) Single Linkage

Figura 2.2: Comparação entre a abordagem de clusterização utilizada pelo *Density Clustering* e a utilizada pelo *Single Linkage*.

2.3 O algoritmo Random Linkage

Embora mais eficaz que o *Multistart* e o *Density Clustering*, o algoritmo *Single Linkage* ainda apresenta algumas deficiências, tais como a necessidade de armazenamento dos minimizadores encontrados na Etapa de Refinamento (conjunto $\tilde{X}$) e dos pontos aleatórios gerados no início de cada iteração (conjunto S).

Visando resolver a primeira situação, foram propostas em [19] algumas melhorias para o *Single Linkage*. Este algoritmo, chamado *Random Linkage*, substitui o conjunto S por uma função φ, que calcula a probabilidade da execução do procedimento de busca local a partir de um dado ponto x_{k+1}. Esta função é definida como:

$$\varphi = \begin{cases} 1 & se \;\; \delta_k > r_{k+1;1;\sigma} \\ 0 & \text{caso contrário.} \end{cases}$$

onde

$$r_{k+1;1;\sigma} = \pi^{-\frac{1}{2}} \left(\Gamma\left(1 + \frac{n}{2}\right) \mu(\Omega)\, \sigma \frac{logk}{k} \right)^{\frac{1}{n}}$$

e

$$\delta_k(x) = \min_{j=1,\ldots,k} \{\|x_{k+1} - x_j\| \mid f(x_j) < f(x_{k+1})\}.$$

Caso não exista j tal que $f(x_j) < f(x_{k+1})$ então fazemos $\delta_k(x) = \infty$.

Observe que quando $\Omega = \{x \in \Re^n \mid l \leq x \leq u\}$, $\mu(\Omega) = \prod_{i=1}^{n}(u_i - l_i)$ (veja [4]).

Algoritmo 2.5. Random Linkage

Dado $K \in \mathbb{N}$

Passo 0. Faça $k \leftarrow 0$

Passo 1. Gere um ponto x_{k+1} de uma distribuição uniforme sobre Ω

Passo 2. Inicie uma busca local em x_{k+1} com probabilidade $\varphi_k(\delta_k(x_{k+1}))$

Passo 3. $k \leftarrow k+1$

Se $k < K$ volte ao Passo 1

O algoritmo retorna y que corresponde ao ponto encontrado no Passo 2 com menor valor da função objetivo.

Capítulo 3

Utilização da curva de *Lissajous* como estratégia de *Tunneling*

3.1 Introdução

O método de *Tunneling* utiliza uma abordagem diferente dos métodos de clusterização na busca por uma boa aproximação de uma solução global. Sua estratégia consiste em minimizar uma função auxiliar T que, dado um minimizador local $\tilde{x}$, nos permita escapar de sua área de atração. Para isto, tomando como ponto inicial $x = \tilde{x} + p$, para $p \in \Re^n$ e $\|p\| \ll 1$, e utilizando T como função objetivo, é executado um procedimento de minimização local P em busca de um ponto $x^u \in \Omega$ tal que $f(x^u) \leq f(\tilde{x})$ e $x^u \neq \tilde{x}$. Para evitar que o procedimento de minimização local seja atraído por $\tilde{x}$, é criado um pólo positivo neste ponto de forma que $lim_{x \to \tilde{x}} T(x) = +\infty$. Para a criação deste pólo considere a seguinte função T, conforme indicado em [17, 16],

$$T(x) = \frac{f(x) - f(\tilde{x})}{\|x - \tilde{x}\|^{\lambda}} \quad , \lambda \geq 0. \tag{3.1}$$

Supondo $x = \tilde{x} + p$ e, utilizando a aproximação de 2ª ordem de Taylor, temos

$$T(x) \cong \frac{p^t \nabla f(\tilde{x}) + p^t \nabla^2 f(\tilde{x}) p}{\|p\|^{\lambda}}.$$

Para que se forme o pólo em $\tilde{x}$ queremos que, para pontos muito próximos de $\tilde{x}$, o denominador de T tenda a zero mais rapidamente que seu numerador, ou seja, para $lim_{\|p\| \to 0} T(x)$ queremos que

$$\|p\|^{\lambda} < p^t \nabla f(\tilde{x}) + p^t \nabla^2 f(\tilde{x}) p.$$

Aplicando ln nos dois lados da função temos

$$\lambda \, ln\|p\| < ln \left[p^t \nabla f(\tilde{x}) + p^t \nabla^2 f(\tilde{x}) p \right]$$

que implica

$$\lambda > \frac{ln \left[p^t \nabla f(\tilde{x}) + p^t \nabla^2 f(\tilde{x}) p \right]}{ln\|p\|}.$$

Desta forma, para λ suficientemente grande, teremos a formação de um pólo positivo em $\tilde{x}$. Na Figura 3.1 observamos um exemplo da função T aplicada a um minimizador local da função $f(x) = sen(x) + sen\left(\frac{10x}{3}\right) + log(x) - 0.84x$.

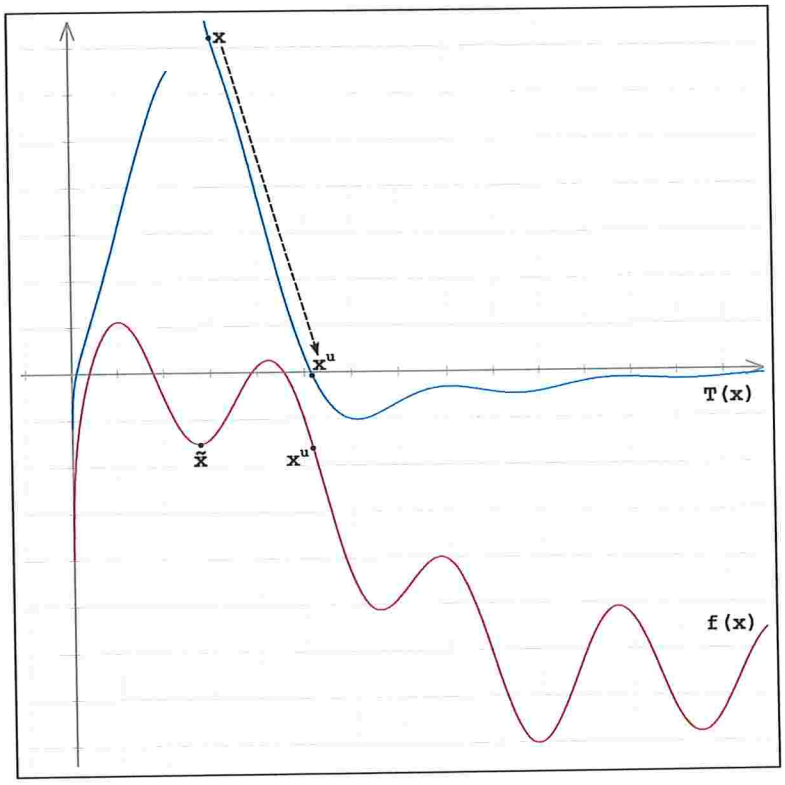


Figura 3.1: Gráfico da função $f(x) = sen(x) + sen\left(\frac{10x}{3}\right) + log(x) - 0.84x$ e um exemplo da função de *tunneling* $T(x)$ com um pólo em $\tilde{x}$.

Partindo de um ponto x próximo a $\tilde{x}$, é executado um procedimento de minimização local P onde tenta-se encontrar um ponto $x^u \in \Omega$ tal que $T(x^u) \leq 0$. O pólo em $\tilde{x}$ garante que $x^u \neq \tilde{x}$. Se x^u é encontrado, o processo de minimização da função objetivo é retomado utilizando x^u como ponto inicial. Caso contrário, o ponto $\tilde{x}$ é considerado pelo método como sendo um minimizador global da função objetivo. Tomando este exemplo como referência, podemos dividir o método de *Tunneling* em duas etapas principais:

Fase de Minimização: Sua finalidade é diminuir o valor da função objetivo através da procura de um minimizador local. Dado um ponto inicial $x^0 \in \Omega$, é utilizado um algoritmo de minimização local P para encontrar um minimizador local $\tilde{x} \in \Omega$.

Fase de *Tunneling:* Seu propósito é obter um bom ponto inicial para a próxima fase de minimização. Para isto, é utilizada uma função de *Tunneling* $T(x)$ onde, a partir do minimizador local $\tilde{x}$, é procurado um ponto $x^u \in \Omega$ que satisfaça a condição $T(x^u) \leq 0$ para $x^u \neq \tilde{x}$.

Para problemas com vários minimizadores em um mesmo nível, ou seja, um outro minimizador local x^u tal que $x^u \neq \tilde{x}$ e $f(x^u) = f(\tilde{x})$, pode ocorrer que a saída da fase de *Tunneling* seja $T(x^u) = 0$. Para que o algoritmo não fique alternando entre estes minimizadores locais, é inserido um novo pólo em x^u na próxima iteração do algoritmo. Com isto, a função $T(x)$ resultante é definida como:

$$T(x) = \frac{f(x) - f(x^u)}{(\|x - x^u\|^{\lambda_u})\{\prod_{i=1}^{l-1} \|x - \tilde{x}_i\|^{\lambda_i}\}}, \tag{3.2}$$

onde l corresponde ao número de minimizadores locais de mesmo valor encontrados pelo método de *Tunneling*. Quando a saída da fase de *Tunneling* não for um minimizador local da função objetivo, l deve voltar ao seu valor inicial ($l = 1$).

É possível que a função de *Tunneling* seja atraída por um ponto irrelevante ao problema, por exemplo um ponto estacionário x^m. Para que $T(x)$ saia da região de atração de x^m, é adicionado um novo termo em $T(x)$ tal que exista um pólo em x^m:

$$T(x) = \frac{f(x) - f(\tilde{x}_l)}{\{\prod_{i=1}^{l} \|x - x^i\|^{\lambda_i}\}(\|x - x^m\|^{\lambda_m})} \tag{3.3}$$

Em alusão ao método descrito até este ponto, o conceito de *Tunneling* foi generalizado como uma forma de escapar de um minimizador local $\tilde{x}$ através de um novo ponto x^u tal que $f(x^u) \leq f(\tilde{x})$. Mais ainda, caso a fase de *Tunneling* não seja bem sucedida, é possível recomeçar o algoritmo a partir de um novo ponto inicial. O procedimento completo é descrito a seguir:

Algoritmo 3.1. Método de Tunneling

Dados $J \in \mathbb{N}$ e $K \in \mathbb{N}$

Passo 0. $f_{\min} \leftarrow \infty$; $k \leftarrow 1$

Passo 1. Gere um ponto $x_k \in \Omega$ com distribuição uniforme

Passo 2. Utilizando x_k como ponto inicial, execute P para encontrar um minimizador local $\tilde{x}_k$

Passo 3. Utilizando $\tilde{x}_k$ como ponto inicial, execute o procedimento de *Tunneling* na tentativa de encontrar x_k^u tal que $f(x_k^u) < f(\tilde{x}_k)$. Se x_k^u for encontrado em até J tentativas então faça $x_k \leftarrow x_k^u$ e volte ao Passo 2

Passo 4. Se $f(\tilde{x}_k) < f_{\min}$ então

$f_{\min} \leftarrow f(\tilde{x}_k)$

$y \leftarrow \tilde{x}_k$

fim se

Passo 5. $k \leftarrow k + 1$

Se $k < K$ volte ao Passo 1

O ponto y é retornado como aproximação de um minimizador global de f.

3.2 A curva de Lissajous

A curva de *Lissajous* consiste na trajetória definida para cada solução da função $lis : \Re \longrightarrow \Re^n$ dada por

$$lis_i(t) = a_i \cos(\theta_i t + \varphi_i), \quad i = 1, \ldots, n \tag{3.4}$$

onde a_i é a amplitude do movimento da curva no componente i, θ_i sua velocidade angular, φ_i sua posição inicial e t refere-se ao instante atual da curva. Para que a curva de *Lissajous* não forme ciclos entre seus componentes, é necessário que eles sejam dois a dois independentes, ou seja, a expressão θ_i/θ_j não pode resultar em um número racional. Para evitar esta ocorrência foi criado o conjunto M, composto pelos n primeiros números primos tal que θ_i é a raiz quadrada do i-ésimo elemento de M.

Em [23] foi provado que a curva descrita em (3.5) é suave e densa na região factível Ω quando as componentes da curva são linearmente independentes. Em particular, foi mostrado que para t tendendo a infinito, a curva de *Lisssajous* cobrirá todas as regiões de Ω.

Implementamos o Algoritmo 3.1 utilizando o procedimento GENCAN, descrito em [6], para a etapa de minimização local e, no lugar da função (3.3), o procedimento *LissajousTunneling* na fase de *Tunneling*. A estratégia utilizada pelo procedimento *LissajousTunneling* é realizar um movimento por uma curva de *Lissajous* que passa pelo minimizador local $\tilde{x}$ encontrado pelo GENCAN.

Através da expressão (3.5) é possível construir uma curva densa em Ω tal que $L(0) = \tilde{x}$. A função $L(\cdot)$ é definida da seguinte forma:

$$L_i(t) = \frac{[l_i + u_i + (u_i - l_i)\, lis_i(t)]}{2}, \; i = 1, \ldots, n, \tag{3.5}$$

onde $a_i = 1$, $i = 1, \ldots, n$ e φ_i, $i = 1, \ldots, n$, é escolhido de tal forma que $L(0) = \tilde{x}$.

A Figura 3.2 mostra o gráfico de duas curvas geradas pela função descrita em (3.5).

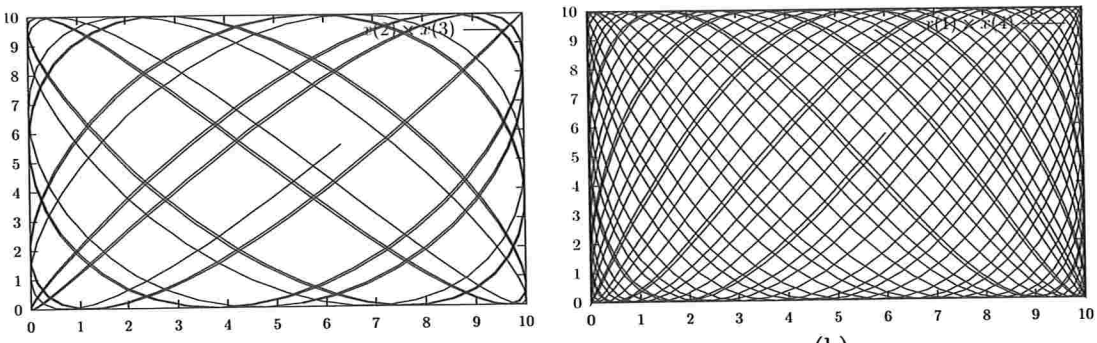
O movimento ao longo de L é realizado testando-se os candidatos:

$$t = \frac{\alpha}{(1 - \mid \alpha \mid)}, \text{ com } \alpha = \frac{1}{2}, -\frac{1}{2}, \frac{1}{3}, \frac{2}{3}, -\frac{1}{3}, -\frac{2}{3}, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, \ldots. \tag{3.6}$$

O algoritmo a seguir descreve o procedimento *LissajousTunneling*:

Algoritmo 3.2. LissajousTunneling

Dados $\tilde{x}$, l, u, n, M e $K \in \mathbb{N}$



(a) (b)

Figura 3.2: (a) Gráfico de uma curva de *Lissajous* após percorrer 100 pontos; (b) Gráfico de uma curva de *Lissajous* após percorrer 1000 pontos.

Passo 0. $x_{\min} \leftarrow \tilde{x}$; $k \leftarrow 1$

Passo 1. Para $i \leftarrow 1$ até n faça

$$\varphi_i \leftarrow \arccos\left(\frac{2\tilde{x}_i - l_i - u_i}{u_i - l_i}\right)$$

$$\theta_i \leftarrow \sqrt{M_i}$$

fim para

Passo 2. Calcule o próximo valor de t conforme indicado em (3.6)

Passo 3. $x_k \leftarrow \frac{(l+u)+(u-l)\cos(\theta t+\varphi)}{2}$

Passo 4. Se $f(x_{\min}) > f(x_k)$ então

$$x_{\min} \leftarrow x_k$$

senão

$$k \leftarrow k+1$$

se $k < K$ volte ao Passo 2

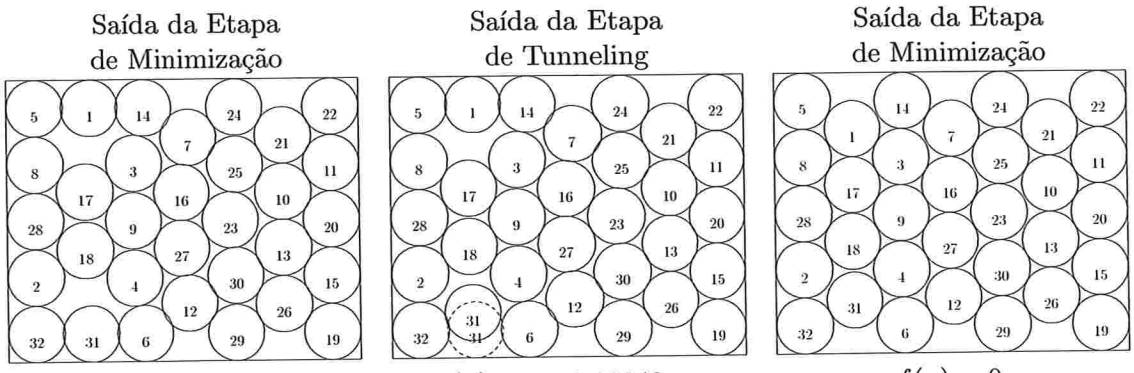
fim se

Passo 5. Retorne $x_{\min}$.

Ao término do Algoritmo 3.2, se $x_{\min} = \tilde{x}$ então é gerado um novo ponto inicial para a etapa de minimização, do contrário o ponto $x_{\min}$ será utilizado como ponto inicial (Passo 3 do Algoritmo 3.1).

3.2.1 Exemplo da utilização da curva de *Lissajous*

Para ilustrar uma aplicação prática da curva de *Lissajous* como estratégia de *Tunneling*, considere o problema de empacotar 32 círculos de raio 8 em uma caixa de dimensões 100×80 sem que haja sobreposição entre estes círculos, ou seja, que a maior interseção entre quaisquer dois círculos da caixa seja de no máximo um ponto (para maiores detalhes sobre a formalização deste problema de empacotamento, vide o Capítulo 6). Na Figura (3.3) observamos a execução da 6ª iteração do algoritmo: em (i) temos a aproximação de um minimizador local encontrado pela Etapa de Refinamento; em (ii), após movimentar o círculo 31, a Fase de *Tunneling* consegue escapar da região de atração do minimizador local devolvendo um novo ponto para a Etapa de Refinamento; em (iii), com a execução do procedimento de minimização local, o algoritmo devolve uma possível solução para o problema.



$f(x) = 4200.43422 \Rightarrow$ $f(x) = 3176.19842 \Rightarrow$ $f(x) = 0.$

Figura 3.3: Exemplo da execução do algoritmo de *Tunneling* utilizando a curva de *Lissajous* para um problema de empacotamento.

Capítulo 4

O algoritmo GRASP

4.1 Introdução

Um algoritmo guloso obtém uma solução ótima para um problema de otimização pela resolução de vários subproblemas que, possivelmente, mantêm um grau de dependência entre si. Partindo do problema geral o algoritmo age através de uma seqüência de decisões em que, para cada uma delas, escolhe a opção que julga ser a melhor no momento. A cada escolha o algoritmo avança na resolução dos subproblemas até que o resultado final resolva a questão originalmente proposta.

Dentre os vários algoritmos que utilizam esta técnica destaca-se o GRASP (*Greedy Randomized Adaptative Search Procedure*), proposto inicialmente em [11, 12] para a resolução de problemas de otimização combinatória e, posteriormente, adaptado em [20, 14, 15] para problemas de otimização contínua.

O GRASP é um algoritmo iterativo composto por duas fases, uma fase de construção e uma fase de busca local. A fase de construção é responsável por calcular, componente a componente, uma nova solução factível x. A cada iteração desta fase é escolhida uma componente $[x]_i$ de x a partir de uma lista de soluções candidatas, geradas por um procedimento guloso. A escolha da componente dentro do conjunto de soluções candidatas é realizada de forma aleatória, visando evitar que o algoritmo fique parado em um ponto estacionário. Durante a execução da fase de construção, cada componente é escolhida uma única vez, não sendo mais considerada nas iterações subseqüentes desta fase. A fase de busca local realiza um refinamento da solução x encontrada na fase de construção, pela verificação dos valores da função objetivo em um conjunto finito de pontos da vizinhança de x. No final do algoritmo, o GRASP retorna como solução o ponto que obteve o menor valor da função objetivo dentre todos os calculados.

4.2 Elementos principais do algoritmo

O GRASP surgiu a partir do algoritmo *Probalistic heuristic*, proposto em [11] como uma variação estocástica da abordagem de Chvátal [8] para o problema de cobertura de conjuntos (*set*

covering problem). Posteriormente, o GRASP foi adaptado em [12] para a resolução de problemas gerais de otimização combinatória. No Algoritmo 4.1 é possível observar a estrutura geral do GRASP para a resolução de problemas de combinatória. No Passo 2 é realizada a fase de construção, através da execução do procedimento *ConstructGreedyRandomizedSolution*. No Passo 3 é realizada a fase de busca local, onde a solução encontrada no passo anterior é refinada em busca de uma melhor aproximação do minimizador global do problema. Por último, é verificado no Passo 5 se o número máximo de iterações foi alcançado.

Algoritmo 4.1. GRASP

Dados $K \in \mathbb{N}; \alpha \in (0, 1]$

Passo 1. $f^* \leftarrow +\infty$

$k \leftarrow 1$

Passo 2. $ConstructGreedyRandomizedSolution(x, \alpha)$

Passo 3. $\tilde{x} \leftarrow LocalSearch(x)$

Passo 4. Se $f^* > f(\tilde{x})$ então

$f^* \leftarrow f(\tilde{x})$

$x^* \leftarrow \tilde{x}$

fim se

Passo 5. $k \leftarrow k + 1$

Se $k < K$ volte ao Passo 2

O ponto x^* é retornado pelo algoritmo como aproximação da solução do problema.

No Algoritmo 4.2 explicaremos a Etapa de Construção do GRASP aplicado ao problema do caixeiro viajante (*travelling salesman problem*). Dadas n cidades e m rotas distintas entre pares de cidades o problema consiste em calcular o custo mínimo de passar por todas as cidades retornando ao ponto de partida. Seja G o grafo tal que o seu conjunto de vértices $V(G)$ representa o conjunto de cidades e o seu conjunto de arestas $A(G)$ representa os possíveis caminhos entre pares de cidades. Suponhamos que as arestas tenham um custo associado as distâncias entre as cidades. Logo, dado um vértice v qualquer o problema consiste em encontrar uma rota que comece e acabe em v e passe por todos os vértices do grafo com custo mínimo. Em cada iteração da Etapa de Construção (Passos $3 - 7$) é incluída uma nova aresta na solução *Solution*. Este processo, é iniciado com a criação de uma lista de candidatos S, formada pelos componentes que podem ser adicionadas a solução parcial que está sendo construída (Passo 2). No Passo 3 são calculados, para todos os elementos de S, o valor da função objetivo caso o elemento em questão seja incorporado na solução parcial do problema. Em seguida, no Passo 4, é montado

um conjunto restrito de soluções RCL[1] com a seleção dos elementos de S que proporcionam os menores resultados da função objetivo. Esta escolha é baseada em um percentual α de tolerância em relação ao melhor resultado encontrado ($c_{\min}$). No Passo 5 é escolhido, de forma aleatória, um elemento $s \in RCL$ para ser incorporado na solução parcial *Solution* (Passo 6). Este fluxo é repetido até que *Solution* seja uma solução completa (Passo 7).

Algoritmo 4.2. ConstructGreedyRandomizedSolution

Dados $x \in \Re^n; \alpha \in (0, 1]$

Passo 1. $Solution \leftarrow \emptyset$

Passo 2. Monte uma lista S de componentes candidatos a entrar na solução

Passo 3. Calcule o custo $c(e)$ de incluir cada um destes elementos na solução do problema

Passo 4. $c_{\min} \leftarrow \min\{c(e) \mid e \in S\}$

$c_{\max} \leftarrow \max\{c(e) \mid e \in S\}$

$RCL \leftarrow \{e \in S \mid c(e) \leq c_{\min} + \alpha(c_{\max} - c_{\min})\}$

Passo 5. Escolha, aleatoriamente, um elemento $s \in RCL$

Passo 6. $Solution \leftarrow Solution \cup \{s\}$

Passo 7. Se *Solution* não for uma solução completa volte ao Passo 2

Passo 8. $x \leftarrow Solution$

Em todo o processo, o valor de α exerce um papel fundamental na qualidade das soluções encontradas. Em problemas de minimização, valores altos de α favorecem o aspecto aleatório do método enquanto valores mais baixos, a escolha gulosa, onde é dada a preferência para os elementos mais próximos de $c_{\min}$. O estudo do comportamento do GRASP em diversos problemas de combinatória favoreceu a compilação de boas estratégias na escolha do conjunto RCL. Em [22] são enumeradas algumas, descritas a seguir:

- valor de α fixo, próximo ao caso extremo da escolha gulosa;
- escolha aleatória de α utilizando uma distribuição uniforme;
- escolha aleatória de α utilizando uma distribuição crescente não uniforme;
- escolha periódica de α, a partir de um conjunto de valores finitos, de acordo com a qualidade das soluções obtidas (estratégia conhecida como GRASP Reativo).

[1] Do original, *Restricted Candidates List*

Qualquer que seja a estratégia utilizada na escolha do conjunto RCL, a solução x, gerada pela fase de construção, não é necessariamente ótima, mesma que localmente. Desta forma, a partir da solução encontrada nesta primeira fase, é iniciada uma etapa de refinamento em busca de melhores soluções na vizinhança de x. Esta etapa é mostrada no Algoritmo 4.3, que em seu passo único realiza uma busca na vizinhança de x, representada por $N(x)$, a procura de um ponto y que apresente menor valor da função objetivo. Em [22] são sugeridas duas formas distintas para a realização da busca em $N(x)$: a primeira, conhecida como *best improving*, consiste em avaliar toda a vinhança de x substituindo a solução atual pelo ponto analisado que obteve menor valor da função objetivo; a segunda, conhecida como *first improving*, consiste em retornar o primeiro ponto que obteve valor da função objetivo menor que a solução atual. Apesar de, na prática, ambas estratégias levarem a mesma solução final, a estratégia *first improving* foi, freqüentemente, mais eficiente em encontrá-la, conforme descrito em [22].

Algoritmo 4.3. LocalSearch

Dados $x \in \Re^n$

Passo 1. Enquanto x não for localmente ótimo faça

Encontre $y \in N(x)$ tal que $f(y) < f(x)$

$x \leftarrow y$

fim enquanto

O ponto x resultante é retornado pelo algoritmo como aproximação de um minimizador local do problema.

4.3 Aplicação do GRASP em problemas de otimização contínua

Com objetivo de extender a aplicação do GRASP a problemas de otimização contínua foi apresentado em [20] uma nova versão do método para resolução de problemas de programação não linear com restrições em caixa. Uma das características mais marcantes desta versão do GRASP é a não utilização de derivadas. Essa decisão visa dar uma menor complexidade nos cálculos do algoritmo ganhando, com isso, eficiência na resolução de alguns problemas de otimização. Como alternativa ao uso de derivadas para o cálculo de direção e passo que resolvem o problema de PNL são utilizados: uma constante $h > 0$, responsável pela determinação do passo do algoritmo, e um conjunto de direções pré-definidas D, utilizadas na fase de busca local. O controle do tamanho de h é realizado pela variável $NumIterNoImprov$ de forma tal que, a cada $MaxNumIterNoImprov$ iterações sem melhorar o valor de $f(x)$, o passo é diminuído em $\frac{h}{2}$. O conjunto de direções D define-se como

$$D(d) = \{d \in \Re^n \mid d_i \in \{-1, 0, 1\}\}. \tag{4.1}$$

O número de direções geradas corresponde a $3^n - 1$, inviabilizando a utilização desta regra para problemas com $n > 4$. Nestes casos, a cada iteração do GRASP, é executada apenas a fase de construção. Após K iterações do algoritmo, o ponto y resultante é considerado a aproximação do minimizador global.

Em cada iteração k do GRASP, é definido um novo ponto x_k. Caso $f(x_k)$ seja menor que a melhor solução encontrada até o momento ($f^* = f(y)$), os valores de f^* e y são atualizados com $f(x_k)$ e x_k, respectivamente. Em linhas gerais, as ações realizadas pelo algoritmo podem ser divididas em três etapas:

- Fase de construção, realizada pelo procedimento guloso *ConstructGreedyRandomizedSolution*, responsável pelas escolhas que levarão à solução global;
- Fase de busca local, realizado pelo procedimento *LocalSearch*, aplicado a problemas de até 4 variáveis, responsável por acelerar a identificação da solução. Observamos que, em nossa implementação do GRASP, o procedimento *LocalSearch* foi substituído pelo GENCAN, conforme será explicado mais detalhadamente no Capítulo 5;
- Conclusão da iteração, onde são atualizados os valores de f^*, y, h e $NumIterNoImprov$.

Algoritmo 4.4. Bloco Principal do GRASP

Dados $K \in \mathbb{N}; MaxNumIterNoImprov \in \mathbb{N}; \alpha \in (0, 1]$

Passo 0. $f^* \leftarrow +\infty$

$k \leftarrow 1$

$h \leftarrow 1$

$x_k \leftarrow \frac{1}{2}(l + u)$

Passo 1. Execute o procedimento $ConstructGreedyRandomizedSolution(x_k, n, h, l, u, \alpha)$

Passo 2. Se $n \leq 4$ então

execute o procedimento $LocalSearch(x_k, x_{localopt}, n, l, u, h)$

$x_k \leftarrow x_{localopt}$

fim se

Passo 4. Se $f^* > f(x_k)$ então

$f^* \leftarrow f(x_k)$

$y \leftarrow x_k$

$NumIterNoImprov \leftarrow 0$

senão

$NumIterNoImprov \leftarrow NumIterNoImprov + 1$

fim se

Passo 5. Se $NumIterNoImprov \geq MaxNumIterNoImprov$ então

$h \leftarrow \frac{h}{2}$

$NumIterNoImprov \leftarrow 0$

fim se

Passo 6. $k \leftarrow k + 1$

Se $k < K$ volte ao Passo 1

O ponto y é retornado como aproximação de um minimizador global do problema.

4.3.1 Fase de construção

A primeira etapa do GRASP consiste em encontrar um ponto x_k que seja uma boa aproximação do minimizador global. Isto é realizado através de sucessivas minimizações, uma para cada componente de x_k. O vetor S guarda cada um dos índices de x_k que ainda não foram minimizados. Para cada índice $i \in S$, são calculados

$$\begin{aligned} [z]_i &= argmin_{t=[l]_i+h\beta\leq[u]_i,\beta\in\mathbb{N}}\{F(t) \equiv f(y) \mid [y]_i = t \ \wedge \ [y]_j = [x_k]_j \ \forall \ j \neq i\}, \\ [g]_i &= F([z]_i) = f([x_k]_1, [x_k]_2, \ldots, [z]_i, \ldots, [x_k]_n), \end{aligned} \tag{4.2}$$

indicado com a execução do procedimento *Minimize* no Algoritmo 4.5. As variáveis $g_{\max}$ e $g_{\min}$ guardam, respectivamente, o maior e menor valor encontrados no vetor g. Após o cálculo de todos os índices de S, aqueles que obtiveram melhor resultado na minimização são colocados em um vetor de candidatos RCL seguindo o seguinte critério: $[g]_i \leq g_{\min} + \alpha(g_{\max} - g_{\min})$, onde α é uma constante pré-definida. Em seguida é escolhido aleatoriamente um índice $i \in RCL$ sendo $[x_k]_i$ atualizado com o valor de $[z]_i$. O objetivo desta escolha é evitar que o procedimento gere sempre o mesmo ponto, ficando preso em uma região onde não se encontra o minimizador global. Este índice é então retirado de S ($S = S \setminus \{i\}$). Todo o processo é refeito até que S esteja vazio.

Algoritmo 4.5. ConstructGreedyRandomizedSolution

Dados $x_k, l, u \in \Re^n, h \in \Re, \alpha \in (0, 1]$

Passo 0. $S \leftarrow \{1, 2, \ldots, n\}$

Passo 1. $g_{\min} \leftarrow +\infty$

$g_{\max} \leftarrow -\infty$

Para cada elemento $i \in S$ faça

$[g]_i \leftarrow Minimize(x_k, h, i, n, l, u, [z]_i)$

se $g_{\min} > [g]_i$ então $g_{\min} \leftarrow [g]_i$

se $g_{\max} < [g]_i$ então $g_{\max} \leftarrow [g]_i$

fim para

Passo 2. $RCL \leftarrow \emptyset$

para cada elemento $i \in S$ tal que $[g]_i \leq g_{\min} + \alpha(g_{\max} - g_{\min})$ faça $RCL \leftarrow RCL \cup \{i\}$

escolha um elemento $j \in RCL$ de forma aleatória

$[x_k]_j \leftarrow [z]_j$

Passo 3. $S \leftarrow S \setminus \{j\}$. Enquanto S não estiver vazio, volte ao Passo 1

Devolva o novo x_k.

4.3.2 Fase de busca local

Nesta etapa é realizado um refinamento da solução encontrada na fase de construção. Isto é realizado através da busca por melhores aproximações do minimizador global nas imediações do ponto x_k. Conforme mencionado em [20], pode ser utilizado para esta tarefa qualquer algoritmo de busca local. Em particular, foi utilizado um algoritmo simples que utiliza-se de direções constantes, indicadas no vetor D. Tomando $x_{localopt}$ como ponto inicial do procedimento, para cada direção $d \in D$ é verificado se $f(x_{localopt} + hd) < f(x_{localopt})$. Em caso positivo, fazemos $x_{localopt} \leftarrow x_{localopt} + hd$ e o processo é repetido para todas as direções de D. O algoritmo termina quando $\nexists d \in D$ tal que $f(x_{localopt} + hd) < f(x_{localopt})$.

Algoritmo 4.7. LocalSearch

Dados $x_k, l, u \in \Re^n$ e $h \in \Re$

Passo 0. Gere D conforme descrito na seção (4.1)

$D' \leftarrow D$

$x_{localopt} \leftarrow x_k$

Passo 1. Selecione uma direção $d \in D$

$D \leftarrow D \setminus \{d\}$

$\bar{x} \leftarrow x_{localopt} + hd$

se $\bar{x}$ é factível e $f(\bar{x}) < f(x_{localopt})$ então

$x_{localopt} \leftarrow \bar{x}$

$D \leftarrow D'$

fim se

Passo 2. Se D não estiver vazio então vá para o Passo 1

Devolva o novo $x_{localopt}$.

É válido observar que para problemas de até 4 variáveis, o GRASP assemelha-se a algoritmos que utilizam-se da estratégia de *Tunneling*, no sentido em que a Etapa de Construção do GRASP tenta melhorar a solução encontrada por uma Etapa de Refinamento anterior.

4.4 GRASP Contínuo

Através do estudo do comportamento do GRASP em problemas de PNL foram identificadas algumas deficiências, que contribuiram para uma queda na eficiência do método. Baseado nestas evidências, foram propostas em [14, 15] um conjunto de melhorias que resultaram em uma versão alternativa do método denominada GRASP Contínuo (C-GRASP).

O bloco principal do C-GRASP encontra-se descrito no Algoritmo 4.8. Nesta versão, a variável *MaxIterNoImprov* foi substituída por uma faixa de valores definida pelas constantes h_s e h_e, responsáveis por restringir o tamanho do passo h. Esta alteração foi realizada a partir da verificação de que valores muito pequenos de h proporcionavam um aumento considerável no tempo de execução da fase de construção sem que, no entanto, a solução atual fosse melhorada. Além disso, a redução do passo h passou a ser controlada pelos indicadores $Impr_c$ e $Impr_l$ responsáveis por sinalizar mudanças na aproximação do mínimo global y. Com isto, h passou a ser reduzido em toda iteração em que não houve alteração de y, ao invés de ser alterado a cada $MaxIterNoImprov$ iterações.

Algoritmo 4.8. Bloco Principal do C-GRASP

Dados $K \in \mathbb{N}; MaxNumIterNoImprov \in \mathbb{N}; \alpha \in (0, 1]$

Passo 0. $f^* \leftarrow +\infty$

$k \leftarrow 1$

Passo 1. Gere um ponto $x_k \in \Omega$ com distribuição uniforme

$h \leftarrow h_s$

Passo 2. $Impr_c \leftarrow FALSO$; $Impr_l \leftarrow FALSO$

$[x_k, Impr_c] \leftarrow ConstructGreedyRandomizedSolution(x_k, h, l, u, \alpha)$

Passo 3. $[x_k, Impr_l] \leftarrow LocalSearch(x_k, l, u, h, \rho_{lo})$

Passo 4. Se $f^* > f(x_k)$ então

$f^* \leftarrow f(x_k)$

$y \leftarrow x_k$

fim se

Passo 5. Se $Impr_c = FALSO$ e $Impr_l = FALSO$ então

$h \leftarrow \frac{h}{2}$

fim se

Passo 6. $k \leftarrow k + 1$

Passo 7. Se $k < K$ então

Se $h \geq h_e$ volte ao Passo 2

senão volte ao Passo 1

fim se

fim se

O ponto y é retornado como aproximação de um minimizador global de f.

A fase de construção encontra-se descrita no Algoritmo 4.9. A principal alteração desta fase foi a criação da variável $ReUse$ com o intuito de sinalizar alterações de valores nas componentes do ponto x_k. No Passo 1, o procedimento *Minimize* somente será executado se houver alguma mudança em x_k. Com isto, evita-se a realização de buscas locais desnecessárias. No Passo 3 é verificado se houve alteração no posicionamento de x_k. Em caso positivo, $ReUse$ é marcado como $VERDADEIRO$ para que g e z sejam atualizados a partir da nova posição de x_k.

Algoritmo 4.9. ConstructGreedyRandomizedSolution

Dados $x_k, l, u \in \Re^n, h \in \Re, \alpha \in (0, 1]$

Passo 0. $ReUse \leftarrow FALSO$

$S \leftarrow \{1, 2, \ldots, n\}$

Passo 1. $g_{\min} \leftarrow +\infty$

$g_{\max} \leftarrow -\infty$

Para cada elemento $i \in S$ faça

Se $ReUse = FALSO$ então

$[g]_i \leftarrow Minimize(x_k, h, i, n, l, u, [z]_i)$

fim se

se $g_{\min} > [g]_i$ então $g_{\min} \leftarrow [g]_i$

se $g_{\max} < [g]_i$ então $g_{\max} \leftarrow [g]_i$

fim para

Passo 2. $RCL \leftarrow \emptyset$

para cada elemento $i \in S$ tal que $[g]_i \leq g_{\min} + \alpha(g_{\max} - g_{\min})$ faça $RCL \leftarrow RCL \cup \{i\}$

Passo 3. Escolha um elemento $j \in RCL$ de forma aleatória

Passo 4. Se $[x_k]_j = [z]_j$ então

$ReUse \leftarrow VERDADEIRO$

senão

$[x_k]_j \leftarrow [z]_j$

$ReUse \leftarrow FALSO$

$Impr_c \leftarrow VERDADEIRO$

fim se

Passo 5. $S \leftarrow S \setminus \{j\}$. Se S não estiver vazio, volte ao Passo 1

Devolva x_k e $Impr_c$.

Na fase de busca local, a vizinhança de x_k é analisada a procura de pontos com menores valores da função objetivo. Se um ponto é encontrado, então x_k e $Impr_l$ são atualizados, conforme descrito no Algoritmo 4.10. Para determinação dos pontos da vizinhança de x_k que serão analisados, considere

$$S(x_k) = \{\bar{x} \in \Omega \mid \bar{x} = x_k + h\tau,\ \tau \in \mathbb{Z}^n\}, \tag{4.3}$$

um subconjunto dos pontos de Ω que estão a uma quantidade inteira de passos (de tamanho h) distantes de x_k. Seja

$$B_h(x_k) = \left\{ z \in \Omega \mid z = x_k + h\frac{(\bar{x} - x_k)}{\|\bar{x} - x_k\|}, \bar{x} \in S(x_k) \setminus x_k \right\} \tag{4.4}$$

a projeção dos pontos de $S(x_k) \setminus x_k$ em uma hiper-esfera com centro em x_k e com raio h.

Algoritmo 4.10. LocalSearch

Dados $x_k, l, u \in \Re^n, h \in \Re, \rho_{lo} \in (0,1]$

Passo 0. $NumGridPoints \leftarrow \prod_{i=1}^{n} \left\lceil \frac{(u_i - l_i)}{h} \right\rceil$

$MaxPointsToExamine \leftarrow \lceil \rho_{lo} NumGridPoints \rceil$

$NumPointsExamined \leftarrow 0$

Passo 1. Escolha aleatóriamente um elemento $z \in B_h(x_k)$

Passo 2. Se $z \in [l, u]$ e $f(z) < f(x_k)$ então

$x_k \leftarrow z$

$Impr_l \leftarrow VERDADEIRO$

fim se

Passo 3. $NumPointsExamined \leftarrow NumPointsExamined + 1$

Se $NumPointsExamined < MaxPointsToExamine$ volte ao Passo 1

Devolva x_k e $Impr_l$.

Em comparação com o Algoritmo 4.7, os pontos analisados não pertencem necessariamente ao grid $x_k + h\{-1, 0, 1\}^n$. Espera-se com isto, obter melhores aproximações dos mínimos locais. Além disto, o número de pontos a serem analisados é uma função que utiliza-se do parâmetro definido pelo usuário ρ_{lo} podendo ser adaptada de acordo com o problema a ser resolvido.

Capítulo 5

Experimentos computacionais

5.1 Introdução

Para a realização dos experimentos, implementamos em Fortran 77 os métodos *Multistart*, *Random Linkage*, GRASP e *Tunneling* utilizando a curva de Lissajous. Para a Fase de Refinamento (procedimento de minimização local P) utilizamos o algoritmo GENCAN, proposto em [5]. GENCAN é um algoritmo de restrições ativas utilizado para a resolução de problemas com restrições de caixa que, dentro das faces utiliza o método de Newton truncado com buscas lineares enquanto que, para sair da face, utiliza uma iteração do método do gradiente espectral projetado.

O objetivo deste capítulo é avaliar o desempenho dos métodos mencionados, frente a um conjunto de problemas clássicos de minimização em caixa. Os experimentos foram realizados sobre um conjunto de 52 problemas, propostos em [18, 1], criados a partir de um conjunto de 41 funções (vide Apêndice A para maiores informações sobre as funções utilizadas). Para todos os testes foi considerado como único critério de parada a realização de 2000 iterações, ou seja, 2000 minimizações locais. A execução dos experimentos foi realizada em ambiente Linux de 32 bits (Ubuntu 6.06), em um computador AMD $K8$ 1.8 GHZ com 1 GB de RAM, utilizando o compilador GNU Fortran (GCC) 3.4.6 com o parâmetros de compilação `-O4`.

Para avaliação conjunta dos métodos, convencionamos como iteração a realização de uma Etapa de Refinamento, ou seja, uma execução do GENCAN. Além disso, os algoritmos analisados utilizaram um mesmo conjunto de pontos iniciais de forma a garantir que o k-ésimo ponto inicial seria o mesmo para cada um dos algoritmos.

Os resultados dos testes foram divididos em seções de acordo com o método testado. Para cada uma das seções, apresentamos os resultados do método em questão e os comparamos com o método *Multistart*. Por último, comparamos, de forma sintética, o desempenho dos algoritmos que utilizaram o GENCAN na Etapa de Refinamento. Os resultados foram apresentados em tabelas compostas pelas seguintes colunas:

Prob: problema testado (os problemas encontram-se descritos no Apêndice A);

n: número de variáveis utilizado no teste;

$f(x^*)$: valor retornado pelo algoritmo como sendo a aproximação do mínimo global do problema testado;

it_{x^*}: número de iterações realizadas até encontrar $f(x^*)$;

t_{x^*}: tempo, em segundos, até encontrar $f(x^*)$;

f_{eval}: total de avaliações da função objetivo até encontrar o ponto x^*;

g_{eval}: total de avaliações do vetor gradiente até encontrar o ponto x^*;

$\sharp x_0$: total de pontos iniciais gerados até encontrar o ponto x^*.

A Tabela 5.1 mostra o desempenho do algoritmo *Multistart*. Como mencionado anteriormente, o método será avaliado nas seções seguintes comparando-o com os outros métodos.

		Multistart					
Prob	n	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
A.1	2	−1.142650	1	0.00	9	10	1
A.2	2	−377.608733	1	0.00	8	5	1
A.3	2	−186.730909	31	0.01	244	209	31
A.4a	2	−186.730909	143	0.06	1486	1104	143
A.4b	2	−186.730909	125	0.06	1348	996	125
A.5	5	0.000000	27	0.02	566	419	27
A.5	8	0.000000	27	0.03	644	471	27
A.5	10	0.000000	4	0.01	148	106	4
A.5	20	0.000000	50	0.30	3483	2098	50
A.5	30	0.000000	6	0.16	1190	701	6
A.6	4	−21.392482	214	0.28	7185	5438	214
A.7	2	−24.062498	74	0.02	529	493	74
A.8	7	−274.163160	3	0.00	33	29	3
A.9	2	−176.541793	1	0.00	8	7	1
A.10	2	−3.306868	1426	0.57	14797	11908	1426
A.11	3	−0.490260	2	0.00	21	17	2
A.12	2	0.007396	678	0.23	7456	4776	678
A.13	2	124.362182	1338	61.99	1024276	1025251	1338
A.14	4	0.000307	1223	24.46	503301	484418	1223
A.15	4	85822.201600	1	0.00	11	12	1
A.16	5	0.000000	26	0.01	285	219	26
A.17	2	0.000000	1	0.00	6	7	1
A.18	2	0.000000	1	0.00	5	6	1
A.19a	4	−10.153199	3	0.00	40	26	3
A.19b	4	−10.536409	3	0.00	51	29	3
A.20	4	0.000000	1	0.00	14	15	1
A.21	2	−1.031628	2	0.00	19	16	2
A.22	2	−186.730909	31	0.01	244	209	31
A.23	2	−78.332331	3	0.00	33	31	3
A.23	3	−117.498497	3	0.00	35	33	3
A.23	4	−156.664663	40	0.02	404	417	40
A.24	2	0.000000	1	0.00	6	6	1
A.25	2	−0.407461	1	0.00	13	14	1
A.26	2	−18.058696	1	0.00	9	10	1
A.27	2	−227.765750	1	0.00	14	15	1
A.28	2	−2429.414770	1	0.00	11	12	1
A.29	2	−2.000000	133	0.04	1272	982	133
A.30	2	0.397887	1	0.00	11	8	1
A.31	1	−3.372897	5	0.00	38	30	5
A.32	2	1.000000	18	0.02	310	227	18
A.33	1	7.000000	1	0.00	3	3	1
A.34	4	0.000000	1	0.00	30	21	1
A.35	2	0.000000	2	0.00	24	21	2
A.36	4	0.000000	1	0.00	41	27	1
A.37	10	0.000000	1	0.00	3	4	1
A.37	20	0.000000	1	0.00	7	8	1
A.37	30	0.000000	1009	1.42	4778	5787	1009
A.37	40	0.000000	148	0.26	651	799	148
A.38	10	0.000000	1	0.00	2	3	1
A.39	10	2.500000	1	0.00	11	12	1
A.40	10	0.000000	288	0.19	5618	2893	288
A.41	1	−5.534433	1	0.00	7	6	1

Tabela 5.1: Resultados obtidos a partir da versão implementada do *Multistart* para um conjunto de problemas propostos em [18, 1].

5.2 *Random Linkage*

Os testes do *Random Linkage* foram realizados considerando três valores para a constante σ, utilizada no cálculo da distância crítica r. Os valores utilizados, propostos em [18], foram $\sigma \in \{1, 2, 4\}$. A Tabela 5.2 mostra os resultados.

As três versões do *Random Linkage* e do *Multistart* encontraram soluções equivalentes em 48 dos 52 problemas testados. Considerando apenas os 4 problemas restantes, o método *Multistart* encontrou soluções melhores ou iguais as encontradas por alguma das variantes do *Random Linkage* em 3 problemas. O *Multistart* encontrou a melhor solução em 51 dos 52 problemas enquanto que *Random Linkage* com $\sigma = 1, 2, 4$ encontrou a melhor solução conhecida em 50, 50 e 49 problemas, respectivamente. Nesse sentido, podemos afirmar que o *Multistart* é levemente mais robusto.

Se consideramos agora os 48 problemas nos quais o *Multistart* e as três variantes do *Random Linkage* encontraram soluções de qualidade equivalente, podemos afirmar que as variantes do *Random Linkage* foram mais eficientes. Por exemplo, o algoritmo *Random Linkage* com $\sigma = 1$ foi mais rápido que o *Multistart* em 11 problemas, mais lento em 4 problemas e utilizou o mesmo tempo em 33 problemas. A comparação de tempos das outras duas variantes do *Random Linkage* com o *Multistart* são similares.

Ao analisar os 11 problemas nos quais o *Random Linkake* com $\sigma = 1$ foi mais rápido do que o *Multistart*, observamos que em 9 problemas os dois métodos acharam a solução partindo do mesmo ponto inicial. Logo, o ganho em eficiência do *Random Linkage* é uma consequência de ter descartado pontos iniciais que se encontravam em regiões de atração de minimizadores locais conhecidos.

		Random Linkage																	
		$\sigma = 1$						$\sigma = 2$						$\sigma = 4$					
Prob	n	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
A.1	2	−1.14265	1	0.00	10	10	1	−1.14265	1	0.00	10	10	1	−1.14265	1	0.00	10	10	1
A.2	2	−377.60873	1	0.00	9	5	1	−377.60873	1	0.00	9	5	1	−377.60873	1	0.00	9	5	1
A.3	2	−186.73091	19	0.00	184	125	31	−186.73091	17	0.00	164	109	31	−186.73091	11	0.00	119	69	31
A.4a	2	−186.73091	252	0.09	2603	1731	512	−186.73091	227	0.09	3040	1481	1337	−186.73091	101	0.04	2143	521	1463
A.4b	2	−186.73091	40	0.01	522	302	125	−186.73091	29	0.01	414	221	125	−186.73091	23	0.01	364	178	125
A.5	5	0.00000	44	0.04	1141	866	65	0.00000	53	0.05	1297	910	112	0.00000	36	0.04	964	638	152
A.5	8	0.00000	24	0.02	588	415	27	0.00000	33	0.03	840	563	58	0.00000	31	0.04	785	527	58
A.5	10	0.00000	4	0.01	152	106	4	0.00000	4	0.01	152	106	4	0.00000	4	0.01	152	106	4
A.5	20	0.00000	50	0.30	3533	2098	50	0.00000	49	0.29	3372	2010	50	0.00000	48	0.28	3232	1922	50
A.5	30	0.00000	6	0.16	1196	701	6	0.00000	5	0.12	932	554	6	0.00000	5	0.13	932	554	6
A.6	4	−21.50236	924	1.48	37607	23781	6915	−21.50236	924	1.48	37607	23781	6915	−21.50236	566	1.44	33301	14644	14299
A.7	2	−24.06250	37	0.01	323	235	74	−24.06250	30	0.01	270	189	74	−24.06250	23	0.01	216	140	74
A.8	7	−274.16316	3	0.00	36	29	3	−274.16316	3	0.00	36	29	3	−274.16316	3	0.00	36	29	3
A.9	2	−176.54179	1	0.00	9	7	1	−176.54179	1	0.00	9	7	1	−176.54179	1	0.00	9	7	1
A.10	2	−3.30687	514	0.21	5783	3904	1426	−3.30687	332	0.12	3965	2322	1426	−3.30687	1504	1.26	27177	9794	17322
A.11	3	−0.49026	2	0.00	23	17	2	−0.49026	2	0.00	23	17	2	−0.49026	2	0.00	23	17	2
A.12	2	0.00986	207	0.07	2716	1435	423	0.00986	1835	7.08	43696	11666	27922	0.00986	51	0.02	1946	354	1389
A.13	2	259.52386	255	60.59	1000645	1000563	266	259.52386	245	61.53	1000635	1000543	266	259.52386	230	66.27	1000620	1000513	266
A.14	4	0.00031	419	0.58	17945	9893	1859	0.00031	788	2.23	48628	18909	17819	0.00031	578	1.71	40787	13907	17819
A.15	4	85822.20160	1	0.00	12	12	1	85822.20160	1	0.00	12	12	1	85822.20160	1	0.00	12	12	1
A.16	5	0.00000	18	0.00	227	153	26	0.00000	14	0.00	187	124	26	0.00000	13	0.00	175	114	26
A.17	2	0.00000	1	0.00	7	7	1	0.00000	1	0.00	7	7	1	0.00000	1	0.00	7	7	1
A.18	2	0.00000	1	0.00	6	6	1	0.00000	1	0.00	6	6	1	0.00000	1	0.00	6	6	1
A.19a	4	−10.15320	3	0.00	43	26	3	−10.15320	3	0.00	43	26	3	−10.15320	3	0.00	43	26	3
A.19b	4	−10.53641	3	0.00	54	29	3	−10.53641	3	0.00	54	29	3	−10.53641	3	0.00	54	29	3
A.20	4	0.00000	1	0.00	15	15	1	0.00000	1	0.00	15	15	1	0.00000	1	0.00	15	15	1
A.21	2	−1.03163	2	0.00	21	16	2	−1.03163	2	0.00	16	13	3	−1.03163	2	0.00	16	13	3
A.22	2	−186.73091	19	0.00	184	125	31	−186.73091	17	0.00	164	109	31	−186.73091	11	0.00	119	69	31
A.23	2	−78.33233	3	0.00	36	31	3	−78.33233	3	0.00	36	31	3	−78.33233	3	0.00	36	31	3
A.23	3	−117.49850	3	0.00	38	33	3	−117.49850	3	0.00	38	33	3	−117.49850	3	0.00	38	33	3
A.23	4	−156.66466	32	0.00	352	327	40	−156.66466	35	0.02	410	361	72	−156.66466	57	0.04	2047	590	1485
A.24	2	0.00000	1	0.00	7	6	1	0.00000	1	0.00	7	6	1	0.00000	1	0.00	7	6	1
A.25	2	−0.40746	1	0.00	14	14	1	−0.40746	1	0.00	14	14	1	−0.40746	1	0.00	14	14	1
A.26	2	−18.05870	1	0.00	10	10	1	−18.05870	1	0.00	10	10	1	−18.05870	1	0.00	10	10	1
A.27	2	−227.76575	1	0.00	15	15	1	−227.76575	1	0.00	15	15	1	−227.76575	1	0.00	15	15	1
A.28	2	−2429.41477	1	0.00	12	12	1	−2429.41477	1	0.00	12	12	1	−2429.41477	1	0.00	12	12	1
A.29	2	−2.00000	47	0.01	560	335	133	−2.00000	31	0.01	436	234	133	−2.00000	20	0.01	337	156	133
A.30	2	0.39789	1	0.00	12	8	1	0.39789	1	0.00	12	8	1	0.39789	1	0.00	12	8	1
A.31	1	−3.37290	5	0.00	43	30	5	−3.37290	3	0.00	31	21	5	−3.37290	2	0.00	20	14	5
A.32	2	1.00000	14	0.01	264	223	18	1.00000	12	0.01	245	206	18	1.00000	27	0.02	717	363	307
A.33	1	7.00000	1	0.00	4	3	1	7.00000	1	0.00	4	3	1	7.00000	1	0.00	4	3	1
A.34	4	0.00000	1	0.00	31	21	1	0.00000	1	0.00	31	21	1	0.00000	1	0.00	31	21	1
A.35	2	0.00000	2	0.00	26	21	2	0.00000	2	0.00	26	21	2	0.00000	2	0.00	26	21	2
A.36	4	0.00000	1	0.00	42	27	1	0.00000	1	0.00	42	27	1	0.00000	1	0.00	42	27	1
A.37	10	0.00000	1	0.00	4	4	1	0.00000	1	0.00	4	4	1	0.00000	1	0.00	4	4	1
A.37	20	0.00000	1	0.00	8	8	1	0.00000	1	0.00	8	8	1	0.00000	1	0.00	8	8	1
A.37	30	0.00000	996	1.44	5729	5716	1009	0.00000	988	1.43	5695	5674	1009	0.00000	975	1.46	5634	5600	1009
A.37	40	0.00000	148	0.26	799	799	148	0.00000	148	0.26	799	799	148	0.00000	148	0.26	799	799	148
A.38	10	0.00000	1	0.00	3	3	1	0.00000	1	0.00	3	3	1	0.00000	1	0.00	3	3	1
A.39	10	2.50000	1	0.00	12	12	1	2.50000	1	0.00	12	12	1	2.50000	1	0.00	12	12	1
A.40	10	0.00000	275	0.18	5676	2725	330	0.00000	233	0.16	4929	2314	330	2.01332	848	0.66	17021	8177	2262
A.41	1	−5.53443	1	0.00	8	6	1	−5.53443	1	0.00	8	6	1	−5.53443	1	0.00	8	6	1

Tabela 5.2: Resultados obtidos a partir das implementações do *Random Linkage* para um conjunto de problemas propostos em [18, 1].

5.3 Tunneling

Os testes com o método de *Tunneling* foram realizados considerando 2 formas distintas para determinar o valor de t em (3.5), responsável pelo tamanho do passo na curva de Lissajous. A primeira forma, proposta em [23] e que referenciaremos como Lissajous [23], sugere a atualização de t conforme descrito em (3.6). O segundo passo, que referenciaremos como Lissajous(seq), incrementa, a cada iteração do procedimento de *Tunneling*, o passo em 0.02. A Tabela 5.3 mostra os resultados.

As duas versões do método de *Tunneling* e o *Multistart* encontraram soluções equivalentes em 50 dos 52 problemas testados. Nos 2 problemas restantes, o Lissajous(seq) encontrou soluções melhores que o *Multistart*. A versão Lissajous [23], por sua vez, saiu-se melhor que o *Multistart* em apenas um dos citados problemas. Nesse aspecto, podemos considerar o método de *Tunneling* mais robusto que o *Multistart*.

Analisando as duas variações testadas do método de *Tunneling*, o Lissajous(seq) encontrou mais rapidamente a solução esperada em 10 problemas contra 7 do Lissajous [23]. Nos problemas restantes o tempo utilizado pelas duas versões foi o mesmo.

Por último, observamos que o método de *Tunneling* foi particularmente eficiente para problemas pequenos (com até 5 variáveis) enquanto que, em problemas maiores, seu desempenho foi sensivelmente inferior ao *Multistart*. Como conseqüência deste fato, para os 50 problemas em que o método de *Tunneling* obteve soluções equivalentes ao *Multistart*, ambas as versões do método de *Tunneling* demoraram, em média, mais tempo para encontrar a solução que o *Multistart*.

		Lissajous(orig)								Lissajous(Seq)							
		Min. Local						Tunneling		Min. Local						Tunneling	
Prob	n	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$	it_{geral}	it_{tunn}	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$	it_{geral}	it_{tunn}
A.1	2	−1.14265	1	0.00	9	10	1	0	0	−1.14265	1	0.00	9	10	1	0	0
A.2	2	−377.60873	1	0.00	8	5	1	0	0	−377.60873	1	0.00	8	5	1	0	0
A.3	2	−186.73091	2	0.00	112	13	1	1	90	−186.73091	2	0.00	112	13	1	1	112
A.4a	2	−186.73091	25	0.02	14307	164	7	18	2103	−186.73091	22	0.02	14113	137	6	16	3946
A.4b	2	−186.73091	7	0.01	3590	42	2	5	1544	−186.73091	8	0.01	4910	47	2	6	2859
A.5	5	0.00000	3	0.00	2556	44	2	1	489	0.00000	14	0.04	14318	191	6	8	4076
A.5	8	0.00000	33	0.23	55184	578	27	6	2408	0.00000	36	0.25	58470	821	27	9	5396
A.5	10	0.00000	4	0.03	6148	106	4	0	0	0.00000	4	0.03	6148	106	4	0	0
A.5	20	0.00000	50	1.06	101483	2098	50	0	0	0.00000	50	1.06	101483	2098	50	0	0
A.5	30	0.00000	6	0.28	11190	701	6	0	0	0.00000	6	0.28	11190	701	6	0	0
A.6	4	−21.50236	277533	5720.00	116	198	40055	0	0	−21.50236	303035	5789.00	107	206	83360	0	0
A.7	2	−24.06250	5	0.00	1029	25	1	4	1006	−24.06250	4	0.00	1242	22	1	3	1217
A.8	7	−274.16316	3	0.00	4033	29	3	0	0	−274.16316	3	0.00	4033	29	3	0	0
A.9	2	−176.54179	1	0.00	8	7	1	0	0	−176.54179	1	0.00	8	7	1	0	0
A.10	2	−3.30687	43	0.06	35999	281	15	28	7697	−3.30687	118	0.16	92250	721	34	84	25524
A.11	3	−0.49026	2	0.00	102	12	1	1	1	−0.49026	2	0.00	103	12	1	1	90
A.12	2	0.00740	581	0.45	341896	3530	122	459	94920	0.00000	111086	1461.00	38	186	34986	0	0
A.13	2	124.36218	34	12.98	216037	189791	14	20	128	124.36218	6	0.02	1599	60	1	5	1533
A.14	4	0.00031	1269	33.24	2987308	484777	1223	46	39508	0.00031	1340	33.24	3052558	485506	1223	117	103747
A.15	4	85822.20160	1	0.00	11	12	1	0	0	85822.20160	1	0.00	11	12	1	0	0
A.16	5	0.00000	32	0.10	51421	249	26	6	1109	0.00000	34	0.11	54156	258	26	8	3836
A.17	2	0.00000	1	0.00	6	7	1	0	0	0.00000	1	0.00	6	7	1	0	0
A.18	2	0.00000	1	0.00	5	6	1	0	0	0.00000	1	0.00	5	6	1	0	0
A.19a	4	−10.15320	3	0.00	4040	26	3	0	0	−10.15320	3	0.00	4040	26	3	0	0
A.19b	4	−10.53641	3	0.00	4051	29	3	0	0	−10.53641	3	0.00	4051	29	3	0	0
A.20	4	0.00000	1	0.00	14	15	1	0	0	0.00000	1	0.00	14	15	1	0	0
A.21	2	−1.03163	2	0.00	100	12	1	1	88	−1.03163	2	0.00	83	11	1	1	73
A.22	2	−186.73091	2	0.00	112	13	1	1	90	−186.73091	2	0.00	134	13	1	1	112
A.23	2	−78.33233	3	0.00	4033	31	3	0	0	−78.33233	2	0.00	1614	17	1	1	1599
A.23	3	−117.49850	3	0.00	4035	33	3	0	0	−117.49850	2	0.00	1616	18	1	1	1599
A.23	4	−156.66466	40	0.12	78404	417	40	0	0	−156.66466	3	0.00	4027	28	2	1	1995
A.24	2	0.00000	1	0.00	6	6	1	0	0	0.00000	1	0.00	6	6	1	0	0
A.25	2	−0.40746	1	0.00	13	14	1	0	0	−0.40746	1	0.00	13	14	1	0	0
A.26	2	−18.05870	1	0.00	9	10	1	0	0	−18.05870	1	0.00	9	10	1	0	0
A.27	2	−227.76575	1	0.00	14	15	1	0	0	−227.76575	1	0.00	14	15	1	0	0
A.28	2	−2429.41477	1	0.00	11	12	1	0	0	−2429.41477	1	0.00	11	12	1	0	0
A.29	2	−2.00000	403	0.34	334650	2658	133	270	67498	−2.00000	11	0.01	7189	72	3	8	3099
A.30	2	0.39789	1	0.00	11	8	1	0	0	0.39789	1	0.00	11	8	1	0	0
A.31	1	−3.37290	2	0.00	11	11	1	1	1	−3.37290	2	0.00	13	11	1	1	2
A.32	2	1.00000	31	0.06	3830	331	15	16	10475	1.00000	36	0.07	38930	398	18	18	4511
A.33	1	7.00000	1	0.00	3	3	1	0	0	7.00000	1	0.00	3	3	1	0	0
A.34	4	0.00000	1	0.00	30	21	1	0	0	0.00000	1	0.00	30	21	1	0	0
A.35	2	0.00000	2	0.00	11	11	1	1	1	0.00000	2	0.00	31	16	1	1	13
A.36	4	0.00000	1	0.00	41	27	1	0	0	0.00000	1	0.00	41	27	1	0	0
A.37	10	0.00000	1	0.00	3	4	1	0	0	0.00000	1	0.00	3	4	1	0	0
A.37	20	0.00000	1	0.00	7	8	1	0	0	0.00000	1	0.00	7	8	1	0	0
A.37	30	0.00000	1009	18.02	2020778	5787	1009	0	0	0.00000	1009	18.02	2020778	5787	1009	0	0
A.37	40	0.00000	148	3.68	294651	799	148	0	0	0.00000	148	3.68	294651	799	148	0	0
A.38	10	0.00000	1	0.00	2	3	1	0	0	0.00000	1	0.00	2	3	1	0	0
A.39	10	2.50000	1	0.00	11	12	1	0	0	2.50000	1	0.00	11	12	1	0	0
A.40	10	0.00000	562	2.04	556838	5627	255	307	38487	0.00000	168	0.64	169556	1717	73	95	22099
A.41	1	−5.53443	1	0.00	7	6	1	0	0	−5.53443	1	0.00	7	6	1	0	0

Tabela 5.3: Resultados obtidos a partir das versões implementadas do método de *Tunneling*, utilizando a curva de Lissajous, para um conjunto de problemas propostos em [18, 1].

5.4 GRASP

Para a avaliação do GRASP foram utilizadas três versões distintas do método. A primeira, que referenciaremos apenas como GRASP, consistiu na implementação do algoritmo tal como descrito em [20]. Para a realização dos testes, definimos que o passo h, utilizado tanto na Etapa de Construção como de Refinamento do método fosse diminuido em sua metade a cada 20 iterações sem melhora na solução global calculada até aquele momento (parâmetro *MaxNumIterNoImprov*). Além disso, ao contrário dos demais métodos estudados, esta variação do GRASP utilizou como ponto inicial o ponto $((l_1 + u_1)/2, \ldots, (l_n + u_n)/2)$.

Na segunda, substituímos a Etapa Refinamento do GRASP pelo GENCAN. Esta versão, que referenciaremos como GRASP-GENCAN, utilizou, ao contrário do GRASP, pontos iniciais aleatórios. Além disso, foi criado o parâmetro *MaxNoFunctionImprov*, responsável por definir o número máximo de iterações, sem melhora no valor da solução global, antes que um novo ponto inicial seja adotado. Para realização dos testes, utilizamos como valores de *MaxNumIterNoImprov*, *MaxNoFunctionImprov* e α as constantes 2, 10 e 0.3, respectivamente.

A última versão, referenciada como C-GRASP-GENCAN, consistiu na implementação do algoritmo descrito em [14, 15], onde foram propostas melhorias no GRASP para a resolução de problemas de otimização contínua. Na versão implementada, utilizamos o GENCAN na Etapa de Refinamento do algoritmo e, assim como no GRASP-GENCAN, definimos o parâmetro α, utilizado na Etapa de Construção, com o valor de 0.3.

Na Tabela 5.4 verificamos que o GRASP, sem a utilização do GENCAN na Etapa de Refinamento, encontrou soluções melhores que o *Multistart* em 4 problemas, contra 28 do *Multistart*. Se considerarmos o tempo que o algoritmo levou para encontrar sua melhor solução o GRASP foi mais rápido em 12 situações contra 23 do *Multistart*.

Analisando apenas as versões do GRASP que utilizam o GENCAN na Etapa de Refinamento temos que, para os 52 testes realizados, as duas variações do GRASP encontraram a mesma solução que o *Multistart* em 49 problemas. Nos 3 problemas restantes, ambas as variações do GRASP encontraram uma solução melhor que o *Multistart*. Neste sentido, podemos afirmar que o GRASP é mais robusto que o *Multistart*.

Se considerarmos os 49 problemas em que as variações do GRASP utilizando o GENCAN e o *Multistart* encontraram soluções equivalentes, o GRASP-GENCAN foi mais rápido em 17 problemas, contra 5 do *Multistart*; enquanto que o C-GRASP-GENCAN, foi mais rápido em 18 problemas contra 3 do *Multistart*.

Comparando as duas versões do GRASP que utilizam o GENCAN verificamos que o C-GRASP-GENCAN foi mais eficiente que o GRASP-GENCAN em 12 problemas. Tal fato é explicado pelo sucesso do C-GRASP em identificar buscas lineares desnecessárias na Etapa de Construção e, com isto, reduzir o número de cálculos de funções. Isto mostra que as melhorias propostas para o C-GRASP-GENCAN aumentaram a eficiência do método.

		Multistart						GRASP				
Prob	n	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	$\sharp x_0$
A.1	2	−1.142650	1	0.00	9	10	1	−1.142647	84	0.01	20796	1
A.2	2	−377.608733	1	0.00	8	5	1	−377.584375	43	0.00	1578	1
A.3	2	−186.730909	31	0.01	244	209	31	−186.721815	129	0.07	86046	1
A.4a	2	−186.730909	143	0.06	1486	1104	143	−186.338941	168	0.35	329474	1
A.4b	2	−186.730909	125	0.06	1348	996	125	−186.729447	431	10.46	10182526	2
A.5	5	0.000000	27	0.02	566	419	27	0.000000	3	0.00	948	1
A.5	8	0.000000	27	0.03	644	471	27	0.000000	3	0.00	2271	1
A.5	10	0.000000	4	0.01	148	106	4	0.000000	3	0.01	3468	1
A.5	20	0.000000	50	0.30	3483	2098	50	0.000000	5	0.14	22055	1
A.5	30	0.000000	6	0.16	1190	701	6	0.000000	5	0.45	48830	1
A.6	4	−21.392482	214	0.28	7185	5438	214	−21.500000	42	0.03	16956	1
A.7	2	−24.062498	74	0.02	529	493	74	−24.062302	148	0.15	173972	1
A.8	7	−274.16316	3	0.00	33	29	3	−274.14845	109	0.25	148789	1
A.9	2	−176.541793	1	0.00	8	7	1	−176.525861	129	0.07	86049	1
A.10	2	−3.306868	1426	0.57	14797	11908	1426	−3.208125	233	0.24	270618	1
A.11	3	−0.490260	2	0.00	21	17	2	−0.490256	469	0.24	114652	3
A.12	2	0.007396	678	0.23	7456	4776	678	0.000000	1	0.00	3614	1
A.13	2	124.362182	1338	61.99	1024276	1025251	1338	124.362204	146	33.47	8353672	1
A.14	4	0.000307	1223	24.46	503301	484418	1223	0.001035	188	3399.86	1955334997	1
A.15	4	85822.2016	1	0.00	11	12	1	85826.7279	84	8.34	753037	1
A.16	5	0.000000	26	0.01	285	219	26	0.000000	1	0.00	316	1
A.17	2	0.000000	1	0.00	6	7	1	0.000000	2	0.00	110	1
A.18	2	0.000000	1	0.00	5	6	1	0.000000	1	0.00	29	1
A.19a	4	−10.153199	3	0.00	40	26	3	−10.153195	1	0.00	194	1
A.19b	4	−10.536409	3	0.00	51	29	3	−10,536283	1	0.00	280	1
A.20	4	0.000000	1	0.00	14	15	1	0.000000	1	0.00	153	1
A.21	2	−1.031628	2	0.00	19	16	2	−1.031551	147	0.02	34830	1
A.22	2	−186.730909	31	0.01	244	209	31	−186.721815	129	0.07	86046	1
A.23	2	−78.332331	3	0.00	33	31	3	−78.332076	103	0.04	80770	1
A.23	3	−117.498497	3	0.00	35	33	3	−117.498114	103	0.12	162319	1
A.23	4	−156.664663	40	0.02	404	417	40	−156.664152	103	0.28	274359	1
A.24	2	0.000000	1	0.00	6	6	1	0.000000	1	0.00	44	1
A.25	2	−0.407461	1	0.00	13	14	1	−0.407461	166	0.14	165374	1
A.26	2	−18.058696	1	0.00	9	10	1	−18.058686	125	0.03	42760	1
A.27	2	−227.76575	1	0.00	14	15	1	−227.765422	125	0.17	163390	1
A.28	2	−2429.41477	1	0.00	11	12	1	−2429.21468	63	0.02	19002	1
A.29	2	−2.000000	133	0.04	1272	982	133	−2.000000	1	0.00	44	1
A.30	2	0.397887	1	0.00	11	8	1	0.397922	128	0.09	168870	1
A.31	1	−3.372897	5	0.00	38	30	5	−3.372883	82	0.00	13172	1
A.32	2	1.000000	18	0.02	310	227	18	1.000000	32	0.00	32	1
A.33	1	7.000000	1	0.00	3	3	1	7.000000	1	0.00	12	1
A.34	4	0.000000	1	0.00	30	21	1	0.000000	1	0.00	316	1
A.35	2	0.000000	2	0.00	24	21	2	0.000000	22	0.00	668	1
A.36	4	0.000000	1	0.00	41	27	1	0.000000	1	0.00	133	1
A.37	10	0.000000	1	0.00	3	4	1	0.000000	4	0.00	6164	1
A.37	20	0.000000	1	0.00	7	8	1	0.000000	6	0.14	35286	1
A.37	30	0.000000	1009	1.42	4778	5787	1009	0.000000	5	0.47	65105	1
A.37	40	0.000000	148	0.26	651	799	148	0.000000	3	0.69	68883	1
A.38	10	0.000000	1	0.00	2	3	1	0.000000	3	0.00	2973	1
A.39	10	2.500000	1	0.00	11	12	1	2.500000	3	0.00	2973	1
A.40	10	0.000000	288	0.19	5618	2893	288	0.000000	3	0.00	5118	1
A.41	1	−5.534433	1	0.00	7	6	1	−5.534432	123	0.00	8527	1

Tabela 5.4: Resultados obtidos a partir da versão implementada do GRASP para um conjunto de problemas propostos em [18, 1].

		GRASP-GENCAN						C-GRASP-GENCAN					
Prob	n	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$	$f(x^*)$	it_{x^*}	t_{x^*}	f_{eval}	g_{eval}	$\sharp x_0$
A.1	2	−1.142650	1	0.00	69	7	1	−1.142650	1	0.00	72	7	1
A.2	2	−377.608733	1	0.00	23	7	1	−377.608733	1	0.00	26	7	1
A.3	2	−186.730909	1	0.00	69	6	1	−186.730909	1	0.00	72	6	1
A.4a	2	−186.730909	22	0.02	16417	107	2	−186.730909	10	0.00	1324	38	2
A.4b	2	−186.730909	8	0.00	907	48	1	−186.730909	6	0.00	655	26	1
A.5	5	0.000000	1	0.00	356	27	1	0.000000	1	0.00	305	27	1
A.5	8	0.000000	1	0.00	780	20	1	0.000000	1	0.00	596	27	1
A.5	10	0.000000	1	0.00	1183	21	1	0.000000	1	0.00	666	21	1
A.5	20	0.000000	1	0.04	4479	45	1	0.000000	1	0.02	2885	45	1
A.5	30	0.000000	1	0.10	9835	50	1	0.000000	1	0.07	6846	50	1
A.6	4	−21.502356	7	0.00	2483	20	1	−21.502356	5	0.00	1193	16	1
A.7	2	−24.062498	6	0.00	456	23	1	−24.062498	5	0.00	381	17	1
A.8	7	−274.163160	1	0.00	238	11	1	−274.163160	1	0.00	211	11	1
A.9	2	−176.541793	1	0.00	68	6	1	−176.541793	1	0.00	71	6	1
A.10	2	−3.306868	108	0.06	28984	559	10	−3.306868	464	0.11	11092	1624	74
A.11	3	−0.490260	32	0.02	1501	210	4	−0.490260	6	0.00	116	25	2
A.12	2	0.000000	12	0.10	192643	40	2	0.000000	11	0.03	63057	33	2
A.13	2	124.362182	1	0.01	3014	8	1	124.362182	1	0.01	3017	8	1
A.14	4	0.001036	64	64.47	38994713	4115	4	0.000941	338	10.28	6293840	1552	39
A.15	4	85822.201600	1	0.02	2019	10	1	85822.201600	1	0.02	1827	10	1
A.16	5	0.000000	12	0.02	19402	32	2	0.000000	27	0.01	9686	72	6
A.17	2	0.000000	1	0.00	43	5	1	0.000000	1	0.00	46	5	1
A.18	2	0.000000	1	0.00	19	5	1	0.000000	1	0.00	22	5	1
A.19a	4	−10.153199	1	0.00	134	10	1	−10.153199	1	0.00	132	10	1
A.19b	4	−10.536409	1	0.00	128	8	1	−10.536409	1	0.00	126	8	1
A.20	4	0.000000	1	0.00	83	14	1	0.000000	1	0.00	85	14	1
A.21	2	−1.031628	1	0.00	20	6	1	−1.031628	1	0.00	23	6	1
A.22	2	−186.730909	1	0.00	69	6	1	−186.730909	1	0.00	72	6	1
A.23	2	−78.332331	2	0.00	258	14	1	−78.332331	2	0.00	264	14	1
A.23	3	−117.498497	1	0.00	255	6	1	−117.498497	1	0.00	261	6	1
A.23	4	−156.664663	2	0.00	833	12	1	−156.664663	2	0.00	685	12	1
A.24	2	0.000000	1	0.00	35	3	1	0.000000	1	0.00	38	3	1
A.25	2	−0.407461	1	0.00	42	3	1	−0.407461	1	0.00	45	3	1
A.26	2	−18.058696	1	0.00	37	5	1	−18.058696	1	0.00	40	5	1
A.27	2	−227.765750	1	0.00	134	12	1	−227.765750	1	0.00	137	12	1
A.28	2	−2429.414770	1	0.00	130	8	1	−2429.414770	1	0.00	133	8	1
A.29	2	−2.000000	2	0.00	74	8	1	−2.000000	2	0.00	80	8	1
A.30	2	0.397887	1	0.00	126	4	1	0.397887	1	0.00	129	4	1
A.31	1	−3.372897	1	0.00	44	4	1	−3.372897	1	0.00	45	4	1
A.32	2	1.000000	23	0.01	2466	130	3	1.000000	11	0.00	482	29	3
A.33	1	7.000000	1	0.00	8	2	1	7.000000	1	0.00	9	2	1
A.34	4	0.000000	1	0.00	88	13	1	0.000000	1	0.00	90	13	1
A.35	2	0.000000	1	0.00	28	10	1	0.000000	1	0.00	31	10	1
A.36	4	0.000000	1	0.00	71	17	1	0.000000	1	0.00	88	23	1
A.37	10	0.000000	1	0.00	1543	4	1	0.000000	1	0.00	1395	4	1
A.37	20	0.000000	3	0.06	17647	10	1	0.000000	3	0.05	13579	10	1
A.37	30	0.000000	2	0.15	26045	7	1	0.000000	2	0.13	21407	7	1
A.37	40	0.000000	2	0.37	45925	7	1	0.000000	2	0.28	35356	7	1
A.38	10	0.000000	1	0.00	992	3	1	0.000000	1	0.00	781	3	1
A.39	10	2.500000	1	0.00	1000	11	1	2.500000	1	0.00	770	11	1
A.40	10	0.000000	3	0.01	5147	18	1	0.000000	3	0.00	3136	18	1
A.41	1	−5.534433	1	0.00	9	4	1	−5.534433	1	0.00	10	4	1

Tabela 5.5: Resultados obtidos a partir das versões implementadas do GRASP, utilizando o GENCAN na Fase de Refinamento, para um conjunto de problemas propostos em [18, 1].

5.5 Comparativo dos resultados

De um modo geral, as soluções encontradas pelos métodos foram muito próximas. Dos 52 problemas avaliados, os métodos encontraram soluções equivalentes em 47 problemas.

Para comparação do desempenho dos métodos utilizaremos gráficos de performance (*performance profiles*), propostos em [10]. Para isto, considere a seguinte formulação: seja o teste de n problemas aplicados a m métodos diferentes. Considere $t_{i,j}$ o tempo que o método j gastou para encontrar uma aproximação do minimizador global para o problema i. Visando guardar os melhores resultados para cada problema, considere $\tilde{t}_i = \min\{t_{i,j} \mid j = 1, \ldots, m\}$. Suponha $T_{k,j} = \{\text{número de testes em que } t_{i,j} \leq k\tilde{t}_i \mid i = 1, \ldots, n\}$, o gráfico de performance de um método j é representado da seguinte forma $g_j = \{(k, T_{k,j}/n) \mid k \geq 1\}$. Desta forma, por exemplo, o par ordenado $(5, 0.8)$ de um certo método j indica que o método conseguiu encontrar uma boa aproximação do minimizador global em 80% dos problemas se considerarmos como tempo máximo 5 vezes o melhor tempo encontrado para cada problema.

Utilizando esta formulação, os métodos foram comparados com relação ao número de iterações e o tempo necessário para encontrar uma boa aproximação de um minimizador global. Uma formulação alternativa, é apresentada no Apêndice C.

Na Figura (5.1) apresentamos o gráfico de desempenho dos métodos em relação ao número de iterações necessárias para a resolução dos problemas. Na Tabela 5.6 analisamos os dados do gráfico quanto a sua eficiência, ou seja, valores alcançados por cada método para $T_{1,j}$, e robustez, que verifica quantos problemas cada método conseguiu resolver independente do número de iterações. Os dados mostram um desempenho inicial muito superior do GRASP onde, em particular, o algoritmo C-GRA-GEN mostrou ser o algoritmo mais rápido em 88% dos problemas testados. Os demais métodos não chegaram a atingir a marca de 50%. Com relação a robustez, o método de *Tunneling*, em sua versão Lis Seq, foi o único a conseguir resolver todos os problemas testados.

	MST	RL(1)	RL(2)	RL(4)	LIS Orig	LIS Seq	GRA-GEN	C-GRA-GEN
Eficiência	0.42	0.42	0.42	0.42	0.44	0.46	0.80	0.88
Robustez	0.94	0.96	0.96	0.94	0.98	1.00	0.98	0.98

Tabela 5.6: Comparativo entre eficiência e robustez dos resultados apresentados no performance profile.

A Figura (5.2) mostra o gráfico de desempenho dos métodos em relação ao tempo gasto para a resolução dos problemas. Na Tabela 5.7 analisamos os dados do gráfico quanto a sua eficiência e robustez. A comparação de tempo mostra que, apesar do GRASP ter sido o método mais eficiente, a diferença de seu desempenho com os demais métodos caiu significativamente em relação a comparação anterior de número de iterações. Isto mostra que, apesar da Etapa de Construção do GRASP ter sido muito eficiente em encontrar pontos próximos a vizinhança de um minimizador global, utilizou para isto um tempo considerável de processamento. Em segundo lugar, em termos de desempenho, ficou o método *Random Linkage* tendo sido um dos métodos

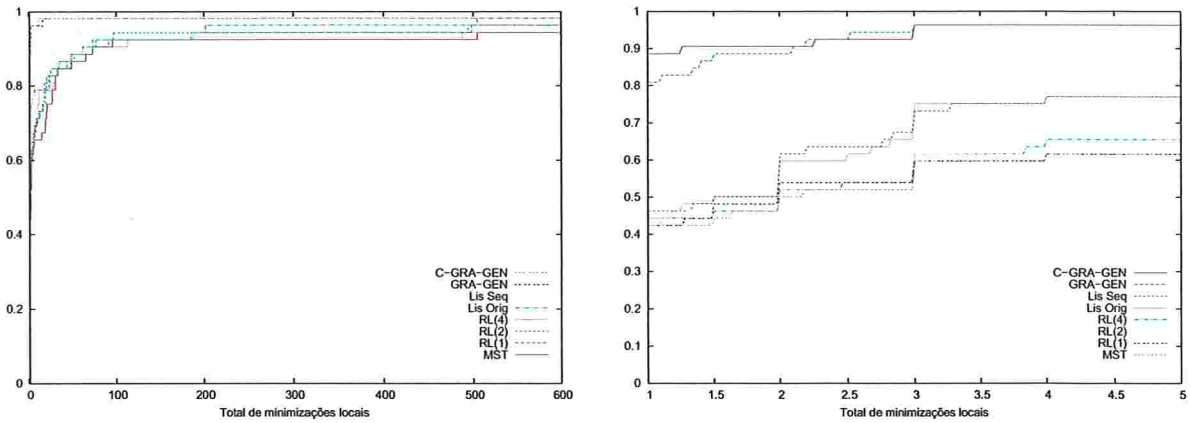


Figura 5.1: Gráfico de desempenho do número de minimizações locais realizadas pelos algoritmos para um conjunto de 52 problemas.

mais rápidos em 73% dos problemas analisados, para a sua versão com $\sigma = 1$. Por último, apesar da sua robustez, o método de *Tunneling* mostrou ser o menos eficiente dos métodos analisados. De forma análoga ao GRASP, o método de *Tunneling* também apresentou uma sensível queda de desempenho entre entre os gráficos de performance de iteração e tempo, explicado pelo alto tempo consumido por sua Etapa de Construção.

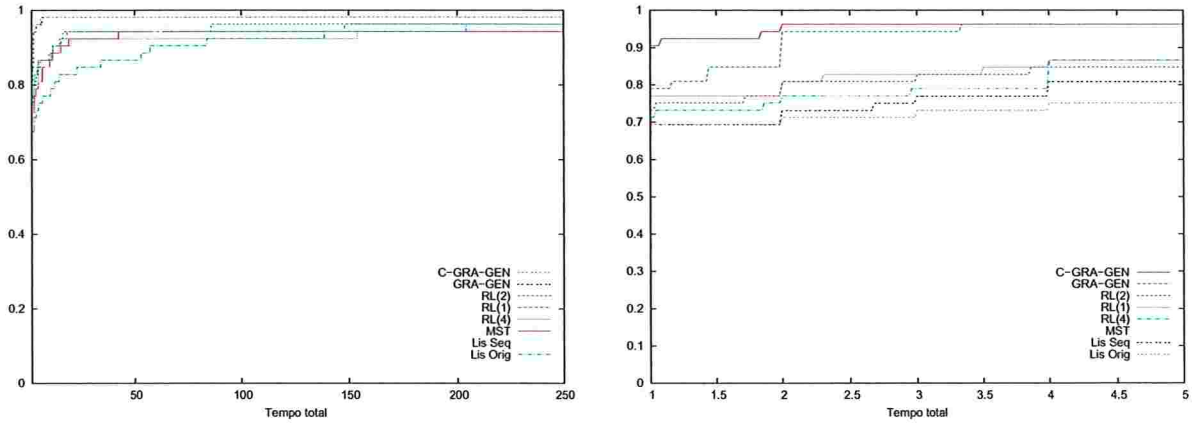


Figura 5.2: Gráfico de desempenho do tempo gasto pelos algoritmos para um conjunto de 52 problemas.

	MST	RL(1)	RL(2)	RL(4)	LIS Orig	LIS Seq	GRA-GEN	C-GRA-GEN
Eficiência	0.67	0.76	0.73	0.71	0.69	0.69	0.78	0.90
Robustez	0.94	0.96	0.96	0.94	0.98	1.00	0.98	0.98

Tabela 5.7: Comparativo entre eficiência e robustez dos resultados apresentados no performance profile.

Os resultados dos testes mostram que os algoritmos C-GRASP-GENCAN, GRASP-GENCAN, *Random Linkage* ($\sigma \in \{1, 2\}$) e *Multistart* foram os que apresentaram, na média, melhor desempenho para o conjunto de testes realizados. Visando avaliar o desempenho dos métodos relacionados a estes algoritmos com um número maior de variáveis, apresentamos no próximo capítulo o resultado de novos experimentos computacionais realizados em problemas de empacotamento. Dentro de cada método escolhemos o algoritmo que obteve melhor desempenho em comparação aos seus pares, em particular, utilizamos nos testes de empacotamento os algoritmos C-GRASP-GENCAN, *Random Linkage* ($\sigma = 1$) e *Multistart*.

Capítulo 6

Experimentos com problemas de empacotamento

6.1 Introdução

Os resultados obtidos no Capítulo 5 mostraram um melhor desempenho dos métodos *Multistart*, *Random Linkage* e GRASP para um conjunto de 52 problemas selecionados a partir literatura clássica de programação não linear. No entanto, a grande maioria dos testes realizados consistiu de problemas de pequeno porte. Por este motivo, analisamos neste capítulo, o desempenho destes métodos para problemas de empacotamento, caracterizados por possuírem um grande número de mínimos locais e apresentarem um número de variáveis sensivelmente maior que os problemas analisados anteriormente. Desta forma, foi possível avaliar o desempenho de suas etapas de construção em dimensões maiores e diante de um número grande de minimizadores locais.

Para a realização dos testes, adaptamos a versões dos métodos *Multistart*, *Random Linkage* ($\sigma = 2.0$) e C-GRASP, utilizados no Capítulo 5, para a resolução de problemas de empacotamento. O critério de parada para cada um dos testes realizados foi a execução de cada algoritmo por 15 minutos. Os experimentos foram realizados em ambiente Linux de 32 bits (Ubuntu 6.06), em um computador AMD $K8$ 1.8 GHZ com 1 GB de RAM, utilizando o compilador GNU Fortran (GCC) 3.4.6 com o parâmetros de compilação `-O4`.

6.2 Definição do problema

Para a realização dos experimentos, foram escolhidos os 21 cenários de testes indicados em [6] que apresentaram o maior número de círculos a serem empacotados. Todos os testes realizados consistiram em uma instância do problema descrito a seguir:

Sejam p^c círculos de raio r e uma caixa de dimensões $[0, d_1] \times [0, d_2]$. Deseja-se saber se é possível empacotar estes círculos de forma que a sua intersecção, para quaisquer par de círculos

p^i e p^j onde $i \neq j$ seja de no máximo um ponto, ou seja, que não haja sobreposições de círculos. Para dados p^c, r, d_1 e d_2, deseja-se encontrar os pontos (centros de círculos) $p^1, \ldots, p^c \in [r, d_1 - r][r, d_2 - r]$ que resolvem o problema:

$$\begin{array}{ll} \text{Minimizar} & \sum_{i \neq j} max(0, (2r)^2 - \|p^i - p^j\|_2^2)^2 \\ \text{sujeita a} & \\ & r \leq p_1^i \leq d_1 - r, \text{ e} \\ & r \leq p_2^i \leq d_2 - r, \text{ para } i = 1, \ldots, c. \end{array} \tag{6.1}$$

Os pontos $p^1, \ldots, p^c$ são os centros dos círculos sendo que p_1^i e p_2^i representam as coordenadas x e y de $p^i \in \Re^2$, $i = 1, \ldots, c$. Se o valor da função objetivo é ZERO então foi encontrado um ponto que é minimizador global de (6.1).

Para cada um dos testes realizados foram indicadas as dimensões da caixa e o raio do círculo a ser empacotado. A partir destas informações, cada algoritmo procurou encontrar, dentro do prazo de 15 minutos, o número máximo de círculos que cabiam na caixa.

Os resultados dos testes foram divididos em seções de acordo com o método testado. Por último, comparamos, de forma sintética, o desempenho dos métodos testados. Os resultados foram apresentados em tabelas compostas pelas seguintes colunas:

Problema: problema testado;

Dimensão: dimensões da caixa onde serão empacotados os círculos;

Raio: raio dos círculos a serem empacotados;

Núm círculos: número de círculos empacotados;

it_{x^*}: número de iterações gastas até encontrar um empacotamento perfeito para o maior número de círculos avaliado;

it_x: número de iterações gastas até encontrar o maior número de círculos que cabem na caixa;

t_{x^*}: tempo, em segundos, até encontrar it_{x^*};

t_*: tempo, em segundos, até encontrar it_x;

f_{eval}: total de avaliações da função objetivo até encontrar x^*;

g_{eval}: total de avaliações do vetor gradiente até encontrar x^*;

Qtd x_0: total de pontos iniciais gerados até encontrar x^*.

6.3 Multistart

O desempenho do método *Multistart* foi semelhante ao resultado apresentado em [6], indicativo de que a implementação do algoritmo está correta. Dos 21 testes realizados, o método obteve sucesso em encontrar o maior número conhecido de círculos empacotados para 21 problemas. No problema 1.4, no entanto, o algoritmo conseguiu empacotar 85 círculos contra 86 de [6]. Esta diferença explica-se pelo tempo levado para empacotar 86 círculos, que em [6] foi de 37108.39 segundos.

Problema	Dimensão	Raio	Núm círculos	it_{x^*}	it_x	$t(x^*)$	$t(*)$	f_{eval}	g_{eval}	Qtd x_0
1	1200 × 800	102	22	3	25	0.03	0.07	474	172	3
2	1200 × 800	101	23	7	32	0.06	0.12	1018	390	7
3	471 × 196	14	126	1477	1700	519.68	620.71	596352	189417	1477
1.1	160 × 80	6	91	7	102	2.61	16.34	5897	1179	7
1.2	100 × 200	8	84	4151	4304	349.19	365.87	900405	326938	4151
1.3	120 × 240	10	74	2646	2723	248.06	252.31	711265	239230	2646
1.4	100 × 80	5	85	1222	1357	124.94	138.09	310276	108958	1222
1.5	120 × 80	6	68	286	358	41.19	43.78	126200	34013	286
1.6	120 × 100	6	87	739	864	109.08	126.03	244146	78094	739
1.7	80 × 80	5	68	16600	16794	816.72	828.60	3097585	1161434	16600
1.8	100 × 100	6	71	10	96	1.53	5.82	5506	1296	10
1.9	120 × 120	7	75	44	126	10.21	15.37	31518	6196	44
2.1	160 × 80	10	32	5	37	0.08	0.24	976	336	5
2.2	100 × 200	13	29	6	36	0.08	0.15	1237	406	6
2.3	120 × 240	15	32	1	33	0.04	0.23	364	115	1
2.4	100 × 80	8	32	1977	2016	16.82	17.02	225078	87915	1977
2.5	120 × 80	9	30	137	181	1.23	1.53	17982	6436	137
2.6	120 × 100	10	30	6	36	0.08	0.21	1302	393	6
2.7	80 × 80	7	32	7	40	0.18	0.33	2125	653	7
2.8	100 × 100	9	30	3	33	0.06	0.14	991	242	3
2.9	120 × 120	11	30	3	33	0.12	0.22	1602	341	3

Tabela 6.1: Resultados obtidos a partir da versão implementada do *Multistart*, para o problema de empacotamento definido em (6.1).

6.4 Random Linkage

O método *Random Linkage* conseguiu empacotar a mesma quantidade de círculos que o *Multistart* em 20 problemas testados. Destes, o método conseguiu melhorar o desempenho do *Multistart* em 5 problemas. Como aspecto negativo, o método obteve desempenho muito inferior que o *Multistart* em 4 problemas. Isto ocorreu porque o método descartou pontos iniciais que resolveriam o problema de empacotamento.

6.5 GRASP Contínuo

Ao contrário do verificado no Capítulo 5, o C-GRASP obteve desempenho médio inferior aos demais métodos testados na resolução dos problemas de empacotamento. Em 9 dos problemas,

Problema	Dimensão	Raio	Núm círculos	it_{x^*}	it_x	$t(x^*)$	$t(*)$	f_{eval}	g_{eval}	Qtd x_0
1	1200 × 800	102	22	3	25	0.03	0.08	477	172	3
2	1200 × 800	101	23	7	32	0.07	0.12	1025	390	7
3	471 × 196	14	126	1926	2217	721.40	888.14	954106	246903	193350
1.1	160 × 80	6	91	22	115	9.20	17.93	20869	3761	1549
1.2	100 × 200	8	84	1745	1883	177.01	190.26	555582	13762	175209
1.3	120 × 240	10	74	4347	4422	429.45	433.28	1636167	396251	237105
1.4	100 × 80	5	85	298	402	35.79	47.45	108280	26951	28970
1.5	120 × 80	6	68	58	130	7.79	10.22	34307	7058	4987
1.6	120 × 100	6	87	98	210	14.18	37.44	38709	9993	9134
1.7	80 × 80	5	67	823	897	58.99	62.84	278193	67533	82424
1.8	100 × 100	6	71	94	170	12.54	15.67	45832	10657	8851
1.9	120 × 120	7	75	173	258	22.82	28.43	6728	1374	506
2.1	160 × 80	10	32	5	37	0.08	0.24	981	336	5
2.2	100 × 200	13	29	6	36	0.08	0.17	1243	406	6
2.3	120 × 240	15	32	1	33	0.04	0.26	365	115	1
2.4	100 × 80	8	32	1788	1827	17.42	17.69	206454	79596	1977
2.5	120 × 80	9	30	128	172	1.21	1.51	17089	6020	137
2.6	120 × 100	10	30	6	36	0.08	0.21	1308	393	6
2.7	80 × 80	7	32	7	40	0.18	0.34	2132	653	7
2.8	100 × 100	9	30	3	33	0.06	0.15	994	242	3
2.9	120 × 120	11	30	3	33	0.12	0.22	1605	341	3

Tabela 6.2: Resultados obtidos a partir da versão implementada do *Random Linkage*, com a constante $\sigma = 1$, para o problema de empacotamento definido em (6.1).

não conseguiu empacotar a mesma quantidade que os demais métodos e, nos testes restantes, o C-GRASP conseguiu empacotar a quantidade esperada de círculos mais rapidamente apenas nos problemas 1.7 e 2.4. A perda de eficiência do algoritmo explica-se pela demora na execução de sua Etapa de Construção com o aumento do número de círculos a serem empacotados. No problema 1.1, por exemplo, foram gastos na Etapa de Construção, com passo $h = 1.0$, 54.44 segundos contra 0.17 da Etapa de Refinamento para empacotar 83 círculos.

Problema	Dimensão	Raio	Núm círculos	it_{x^*}	it_x	$t(x^*)$	$t(*)$	f_{eval}	g_{eval}	Qtd x_0
1	1200 x 800	102	22	5	27	4.78	15.07	1001240	114	1
2	1200 x 800	101	23	68	91	81.83	94.41	15693143	1197	12
3	471 x 196	14	72	1	73	75.77	900.00	1756298	3	1
1.1	160 x 80	6	83	1	84	51.16	914.66	918933	97	1
1.2	100 x 200	8	78	8	86	127.39	876.46	2564407	523	2
1.3	120 x 240	10	72	12	84	170.32	791.56	3979477	160	3
1.4	100 x 80	5	83	13	102	157.72	852.75	2782226	550	3
1.5	120 x 80	6	67	8	82	48.23	318.70	1268398	736	2
1.6	120 x 100	6	84	1	85	49.96	916.41	870709	105	1
1.7	80 x 80	5	68	28	104	127.39	373.97	3078903	530	5
1.8	100 x 100	6	70	11	81	82.64	383.77	2046718	344	3
1.9	120 x 120	7	74	13	87	144.49	651.28	2897795	712	3
2.1	160 x 80	10	32	45	82	21.83	32.09	2188754	511	8
2.2	100 x 200	13	29	10	43	4.33	11.04	551503	246	2
2.3	120 x 240	15	32	19	57	12.86	27.34	1342143	265	4
2.4	100 x 80	8	32	14	85	4.84	21.39	496848	224	3
2.5	120 x 80	9	30	28	88	9.26	21.90	1059352	251	6
2.6	120 x 100	10	30	153	183	47.36	52.44	5611915	1326	28
2.7	80 x 80	7	32	2	34	1.03	6.18	103000	198	1
2.8	100 x 100	9	30	1	31	.81	5.74	92682	68	1
2.9	120 x 120	11	30	2	38	1.41	9.06	167541	95	1

Tabela 6.3: Resultados obtidos a partir da versão implementada do C-GRASP para o problema de empacotamento definido em (6.1).

6.6 Comparativo dos resultados

Analisando o resultados dos testes, observamos que o *Multitart* foi o único método a empacotar o número de círculos esperado em todos os problemas testados. Na Tabela 6.4 verificamos que, dentre os 12 problemas nos quais o *Multistart* e o C-GRASP empacotaram o mesmo número de círculos, o *Multistart* encontrou a solução esperada em um número menor de iterações em 7 problemas contra 5 do C-GRASP. Neste critério, o desempenho do *Random Linkage* foi melhor em 6 problemas contra 5 do *Multistart*. Verificamos ainda, na Tabela 6.5, que o C-GRASP encontrou a solução esperada mais rapidamente apenas no teste 1.7. O motivo desta divergência entre o seu desempenho no número de iterações com o tempo de processamento deve-se ao tempo gasto pela Etapa de Construção do método em encontrar um boa aproximação do minimizador global. O *Random Linkage*, por sua vez, saiu-se melhor em 6 problemas contra 6 do *Multistart*. Nos demais problemas o desempenho de tempo dos dois métodos foi semelhante. Por último, apresentamos nas Figuras 6.1-6.21 a solução final devolvida por cada método para cada um dos problemas testados.

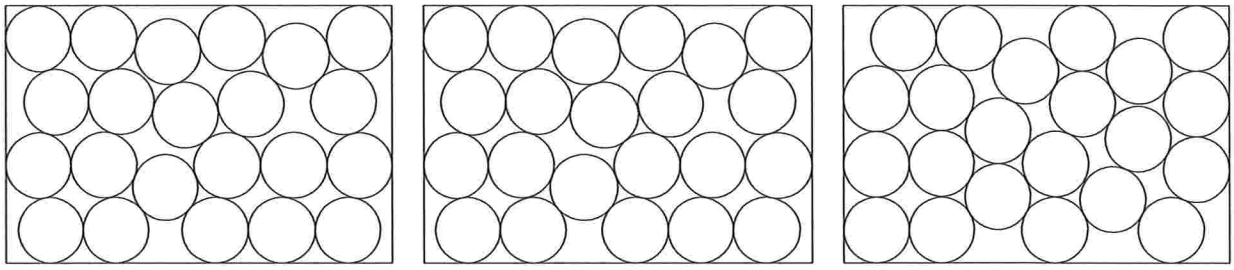
A comparação dos resultados mostra um desempenho ligeiramente superior do *Multistart* em relação ao *Random Linkage*. Em particular, o *Random Linkage* saiu-se melhor em problemas que o *Multistart* teve dificuldades em encontrar um empacotamento. Isto mostra que o *Random Linkage* obteve êxito, em vários dos problemas, em eliminar pontos iniciais desnecessários.

	$it(x^*)$			$it(*)$		
Problema	Multistart	R.Linkage	C-GRASP	Multistart	R.Linkage	C-GRASP
1	3	3	5	25	25	27
2	7	7	68	32	32	91
3	1477	1926	–	1700	2217	–
1.1	7	22	–	102	115	–
1.2	4151	1745	–	4304	1883	–
1.3	2646	4347	–	2723	4422	–
1.4	1222	298	–	1357	402	–
1.5	286	58	–	358	130	–
1.6	739	98	–	864	210	–
1.7	16600	–	28	16794	–	104
1.8	10	94	–	96	170	–
1.9	44	173	–	126	258	–
2.1	5	5	45	37	37	82
2.2	6	6	10	36	36	43
2.3	1	1	19	33	33	57
2.4	1977	1788	14	2016	1827	85
2.5	137	128	28	181	172	88
2.6	6	6	153	36	36	183
2.7	7	7	2	40	40	34
2.8	3	3	1	33	33	31
2.9	3	3	2	33	33	38

Tabela 6.4: Comparativo do número de iterações realizadas por cada um dos métodos na resolução do problema de empacotamento definido em (6.1).

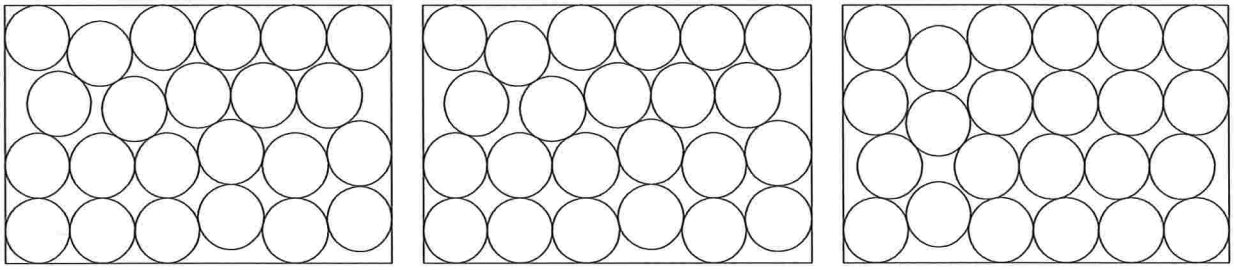
	$t(x^*)$			$t(*)$		
Problema	Multistart	R.Linkage	C-GRASP	Multistart	R.Linkage	C-GRASP
1	0.03	0.03	4.78	0.07	0.08	15.07
2	0.06	0.07	81.83	0.12	0.12	94.41
3	519.68	721.40	–	620.71	888.14	–
1.1	2.61	9.20	–	16.34	17.93	–
1.2	349.19	177.01	–	365.87	190.26	–
1.3	248.06	429.45	–	252.31	433.28	–
1.4	124.94	35.79	–	138.09	47.45	–
1.5	41.19	7.79	–	43.78	10.22	–
1.6	109.08	14.18	–	126.03	37.44	–
1.7	816.72	–	127.39	828.60	–	373.97
1.8	1.53	12.54	–	5.82	15.67	–
1.9	10.21	22.82	–	15.37	28.43	–
2.1	0.08	0.08	21.83	0.24	0.24	32.09
2.2	0.08	0.08	4.33	0.15	0.17	11.04
2.3	0.04	0.04	12.86	0.23	0.26	27.34
2.4	16.82	17.42	4.84	17.02	17.69	21.39
2.5	1.23	1.21	9.26	1.53	1.51	21.90
2.6	0.08	0.08	47.36	0.21	0.21	52.44
2.7	0.18	0.18	1.03	0.33	0.34	6.18
2.8	0.06	0.06	0.81	0.14	0.15	5.74
2.9	0.12	0.12	1.41	0.22	0.22	9.06

Tabela 6.5: Comparativo do tempo demorado por cada um dos métodos na resolução do problema de empacotamento definido em (6.1).



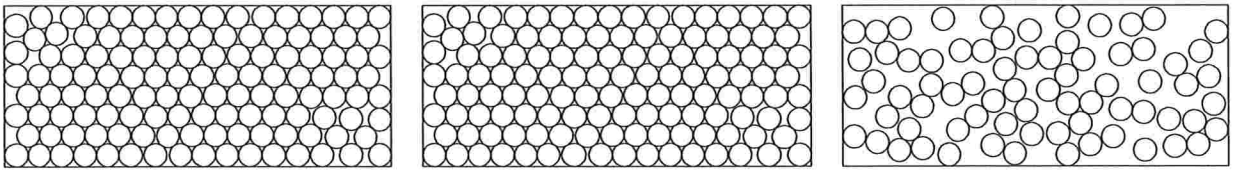
Multistart *Random Linkage* C-GRASP

Figura 6.1: Resultado final encontrado pelos métodos para o problema 1.



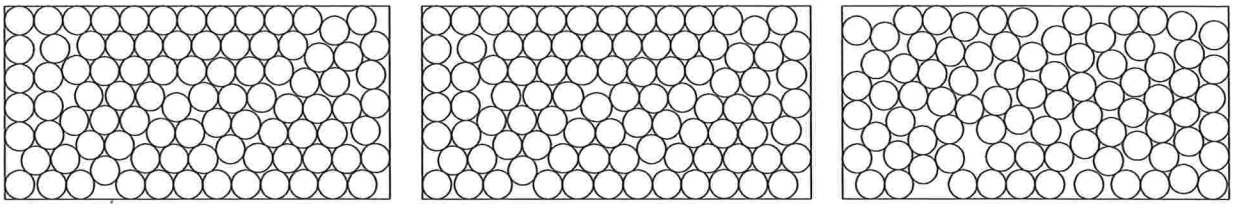
Multistart *Random Linkage* C-GRASP

Figura 6.2: Resultado final encontrado pelos métodos para o problema 2.



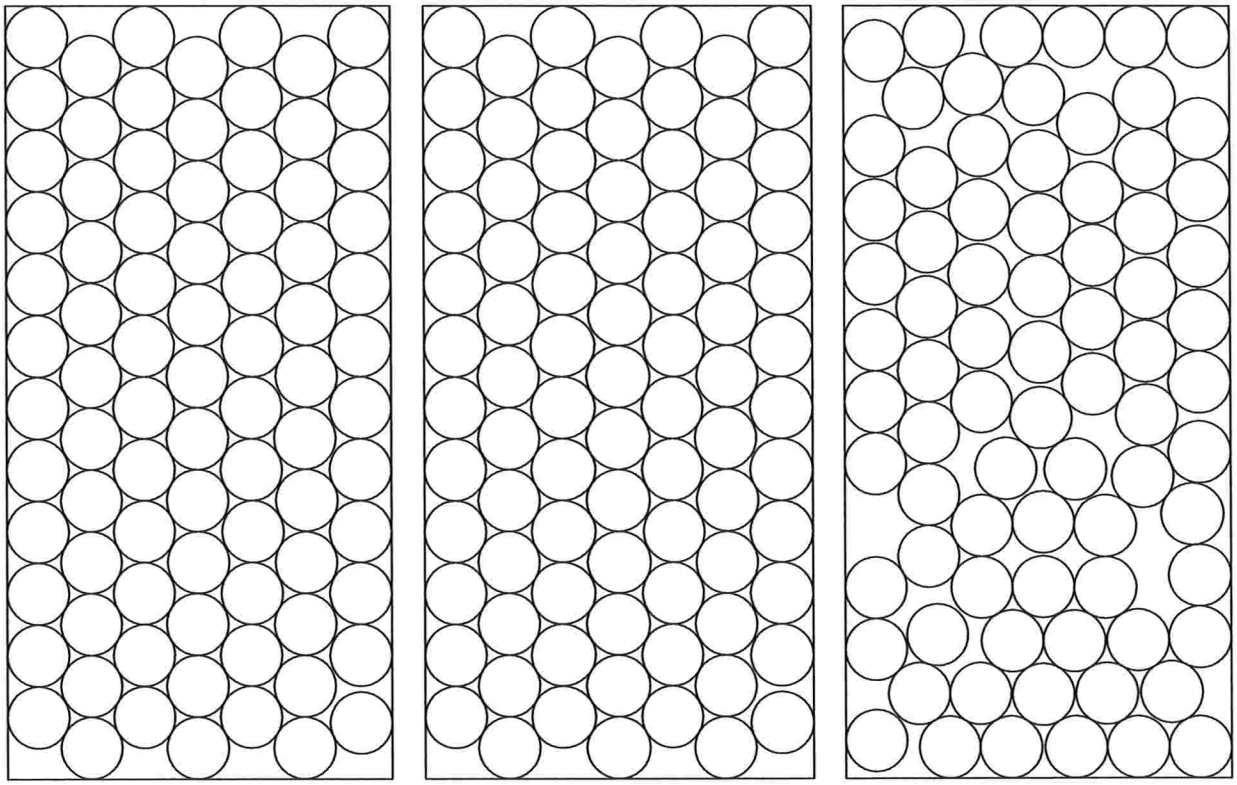
Multistart *Random Linkage* C-GRASP

Figura 6.3: Resultado final encontrado pelos métodos para o problema 3.



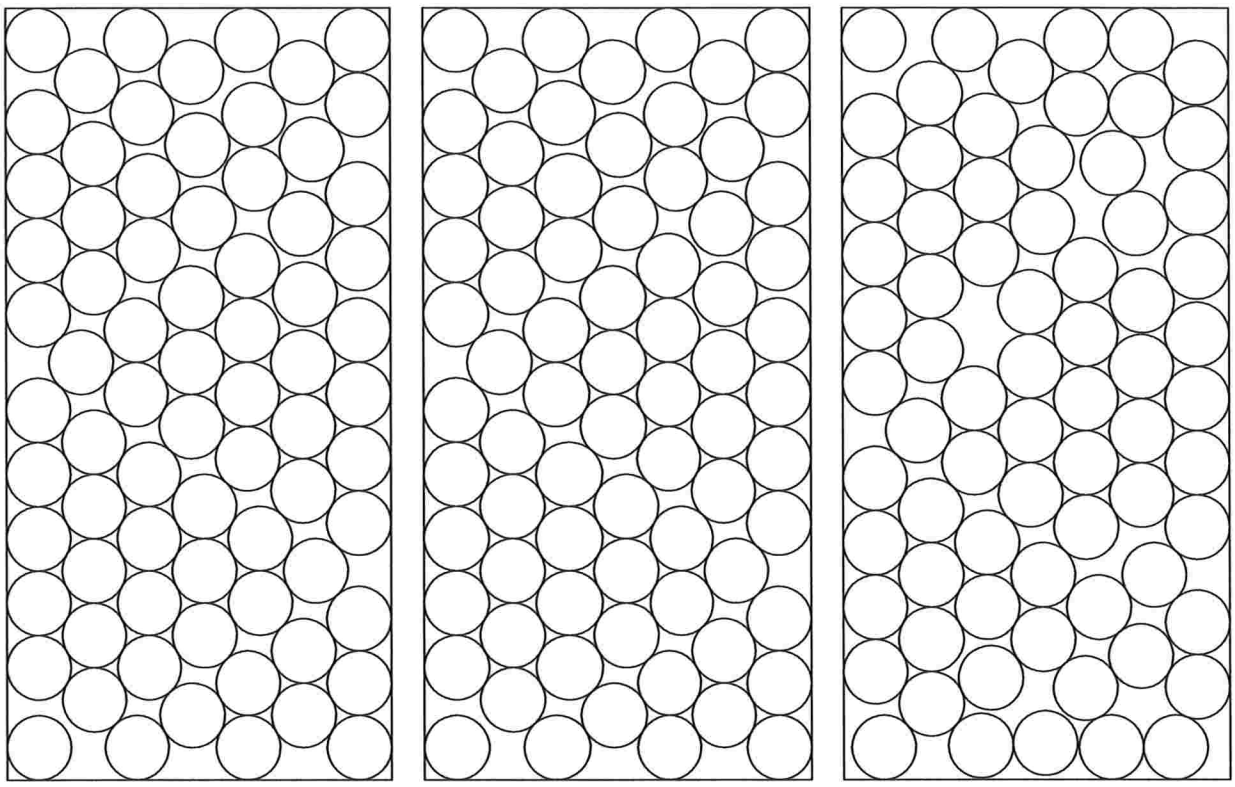
Multistart *Random Linkage* C-GRASP

Figura 6.4: Resultado final encontrado pelos métodos para o problema 1.1.



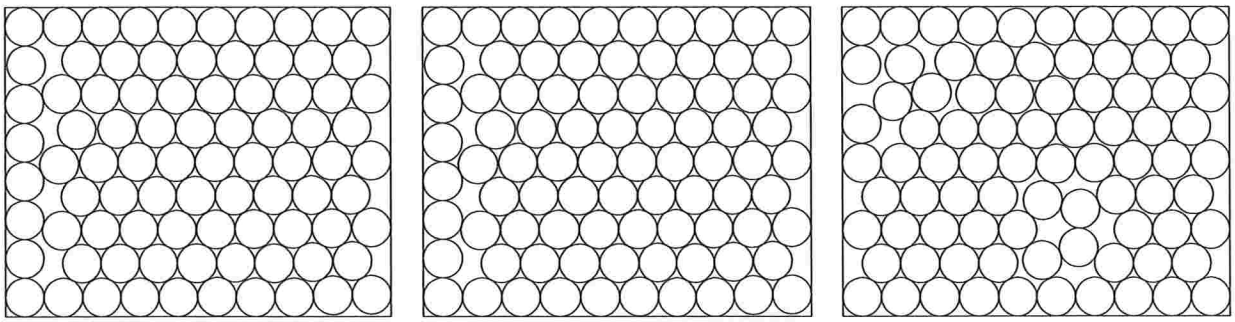
Multistart *Random Linkage* C-GRASP

Figura 6.5: Resultado final encontrado pelos métodos para o problema 1.2.



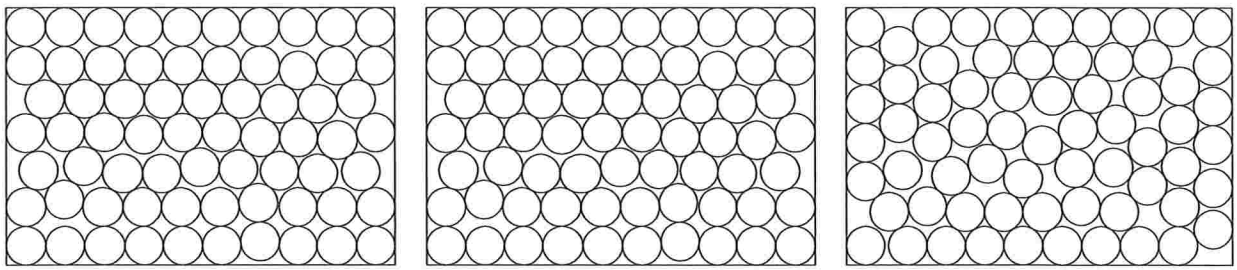
Multistart *Random Linkage* C-GRASP

Figura 6.6: Resultado final encontrado pelos métodos para o problema 1.3.



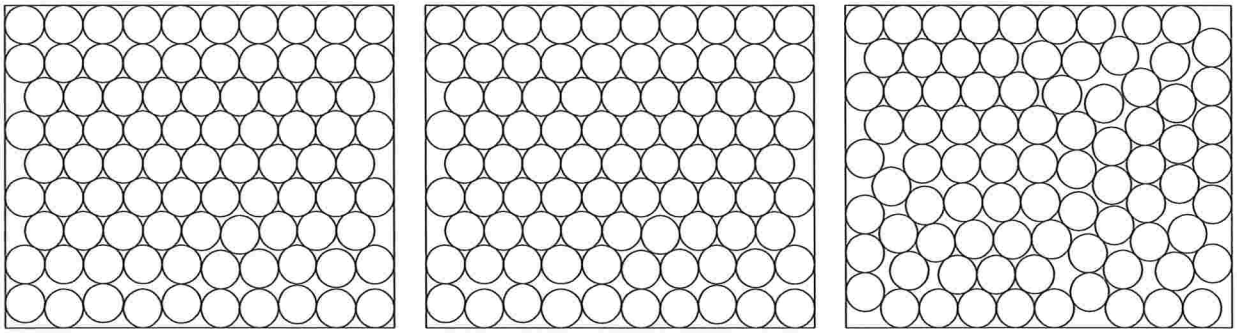
Multistart *Random Linkage* C-GRASP

Figura 6.7: Resultado final encontrado pelos métodos para o problema 1.4.



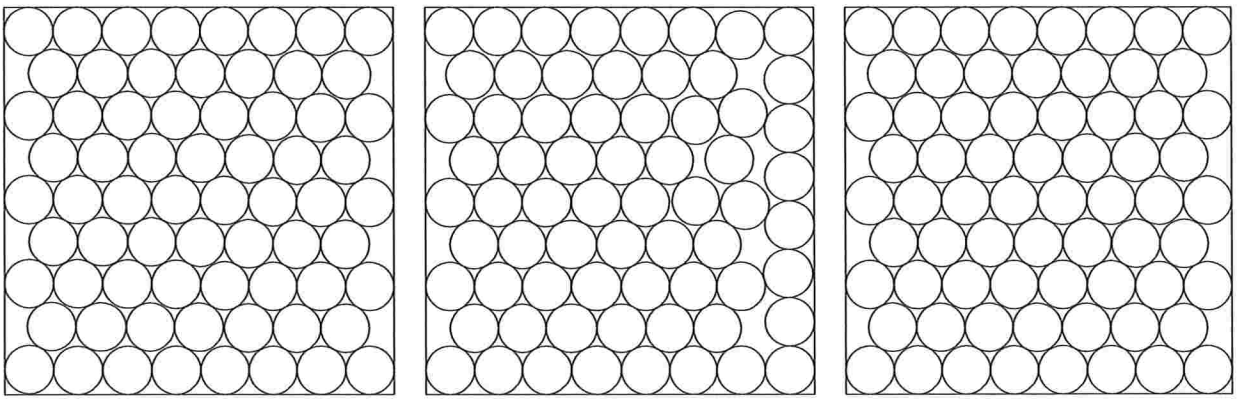
Multistart *Random Linkage* C-GRASP

Figura 6.8: Resultado final encontrado pelos métodos para o problema 1.5.



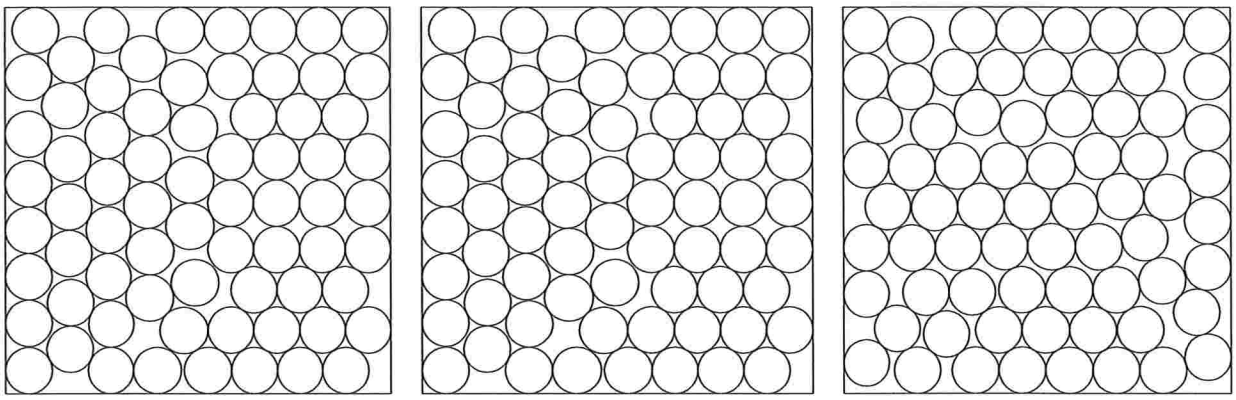
Multistart *Random Linkage* C-GRASP

Figura 6.9: Resultado final encontrado pelos métodos para o problema 1.6.



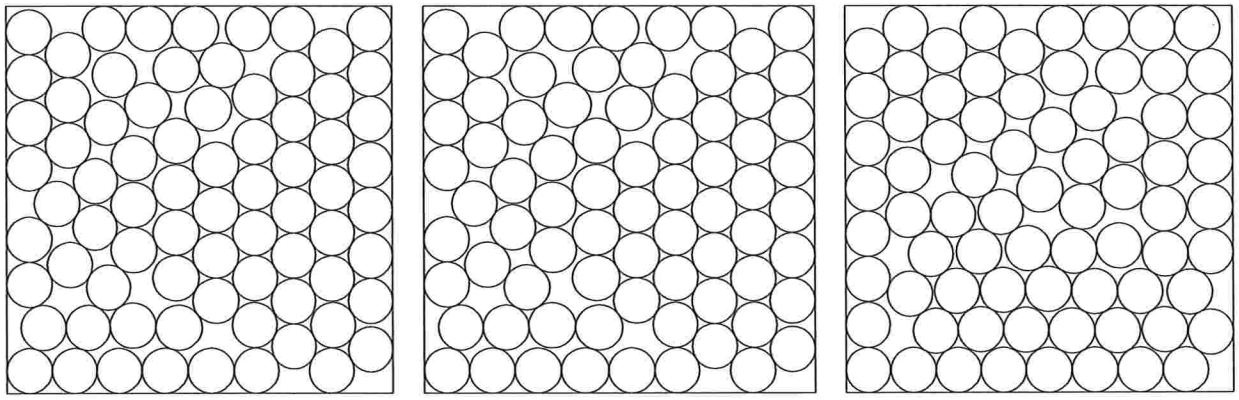
Multistart *Random Linkage* C-GRASP

Figura 6.10: Resultado final encontrado pelos métodos para o problema 1.7.



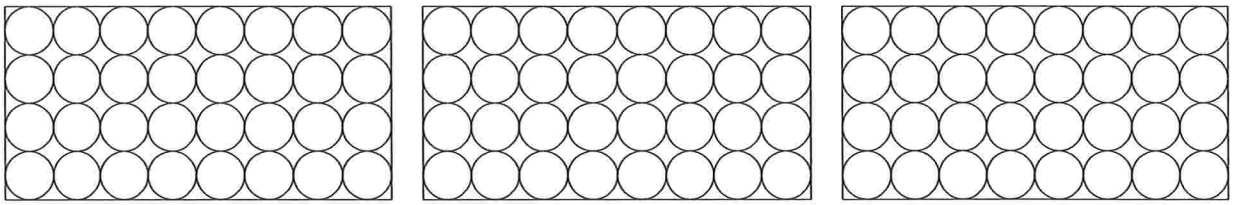
Multistart *Random Linkage* C-GRASP

Figura 6.11: Resultado final encontrado pelos métodos para o problema 1.8.



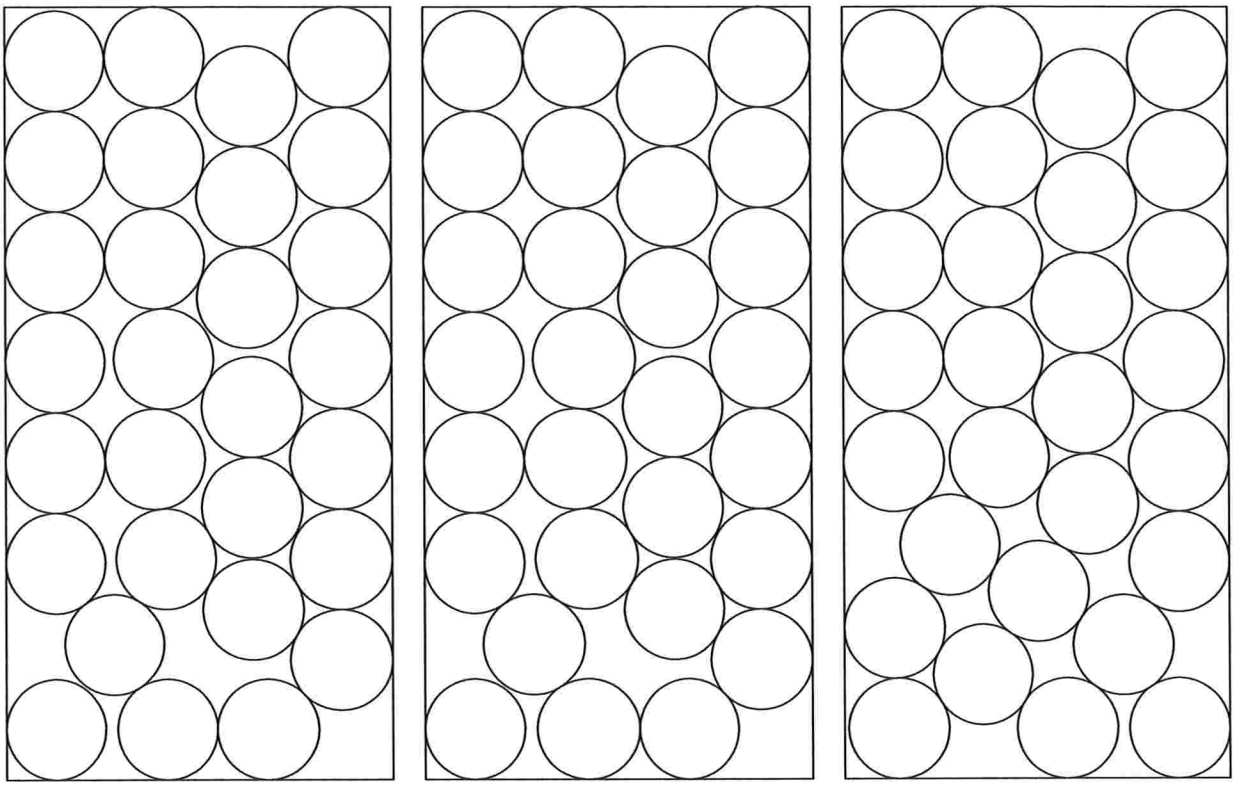
Multistart *Random Linkage* C-GRASP

Figura 6.12: Resultado final encontrado pelos métodos para o problema 1.9.



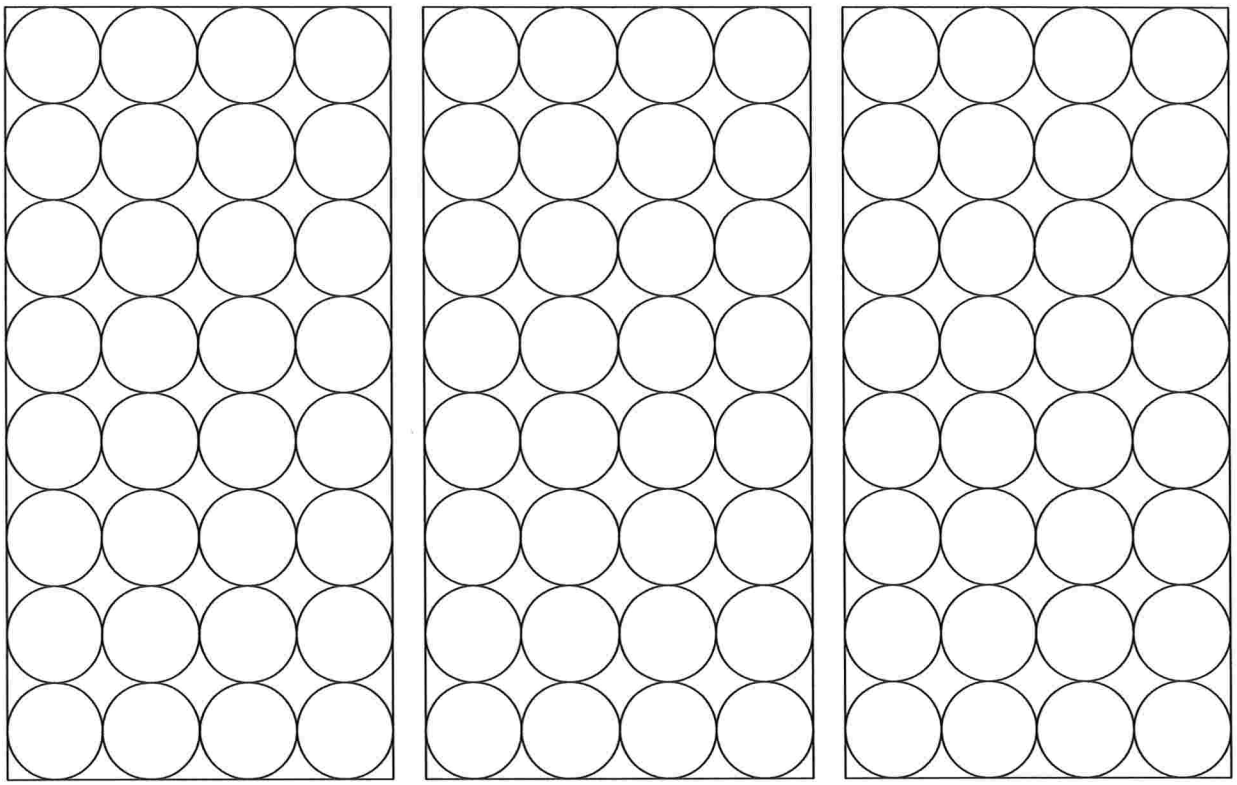
Multistart *Random Linkage* C-GRASP

Figura 6.13: Resultado final encontrado pelos métodos para o problema 2.1.



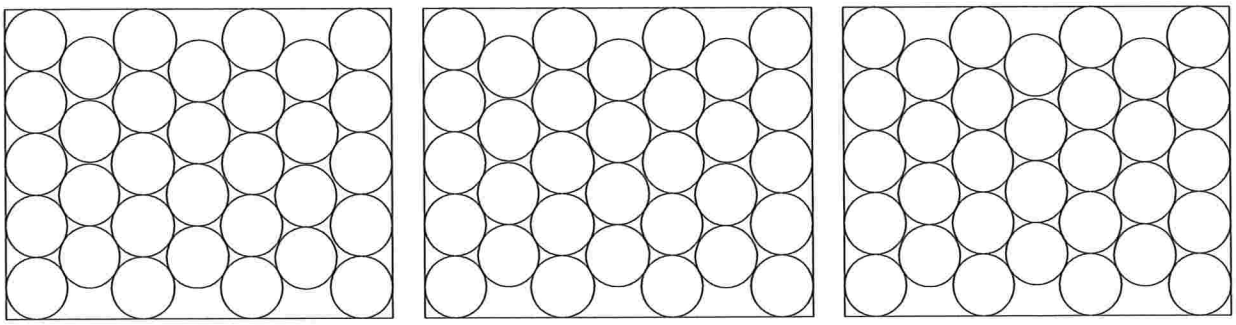
Multistart *Random Linkage* C-GRASP

Figura 6.14: Resultado final encontrado pelos métodos para o problema 2.2.



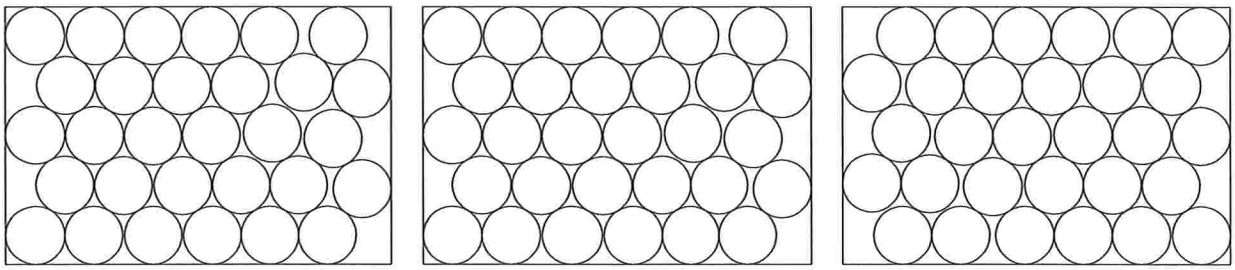
Multistart *Random Linkage* C-GRASP

Figura 6.15: Resultado final encontrado pelos métodos para o problema 2.3.



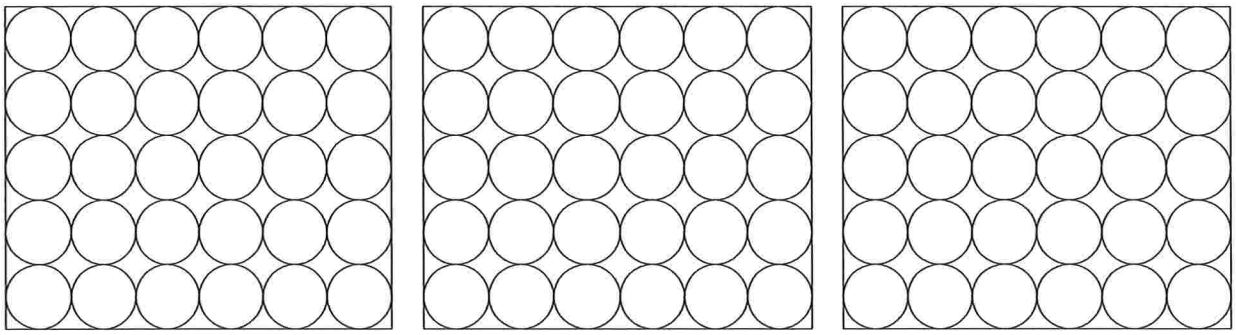
Multistart *Random Linkage* C-GRASP

Figura 6.16: Resultado final encontrado pelos métodos para o problema 2.4.



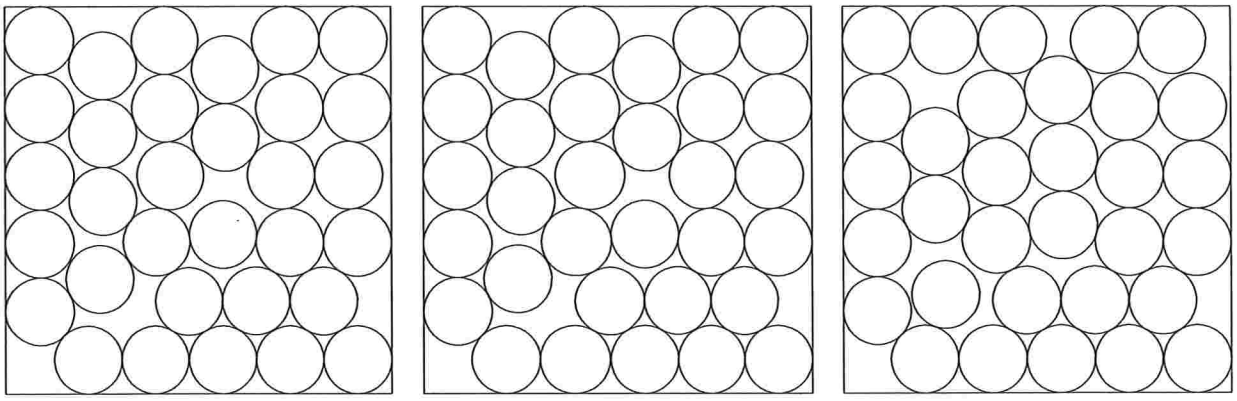
Multistart *Random Linkage* C-GRASP

Figura 6.17: Resultado final encontrado pelos métodos para o problema 2.5.



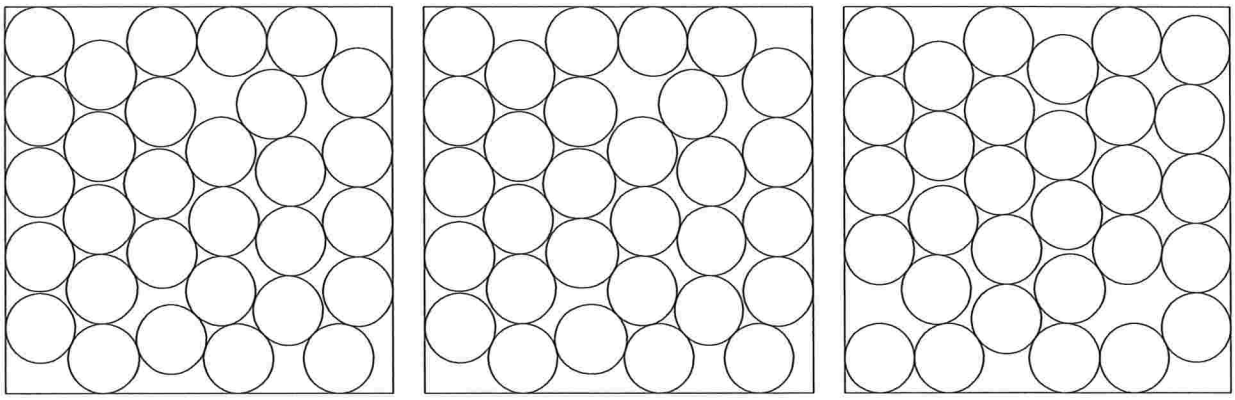
Multistart *Random Linkage* C-GRASP

Figura 6.18: Resultado final encontrado pelos métodos para o problema 2.6.



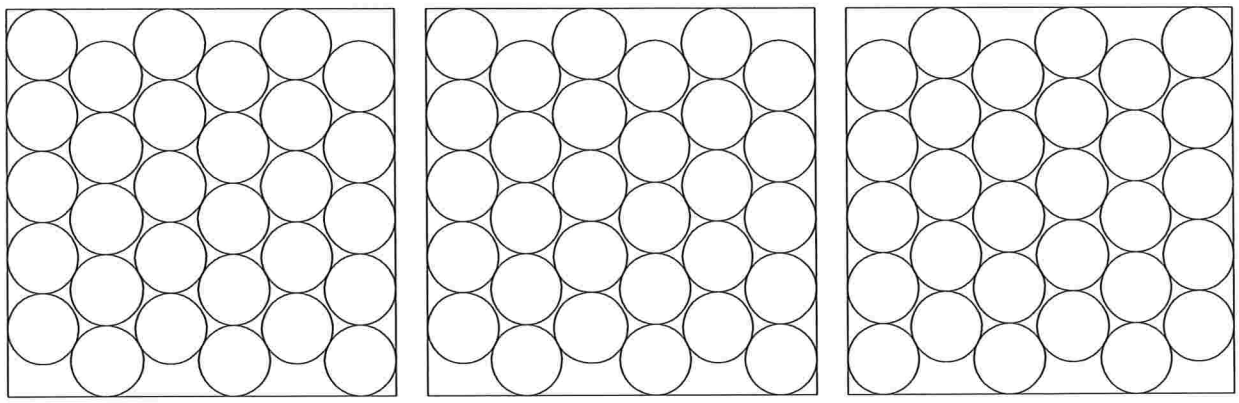
Multistart *Random Linkage* C-GRASP

Figura 6.19: Resultado final encontrado pelos métodos para o problema 2.7.



Multistart *Random Linkage* C-GRASP

Figura 6.20: Resultado final encontrado pelos métodos para o problema 2.8.



Multistart *Random Linkage* C-GRASP

Figura 6.21: Resultado final encontrado pelos métodos para o problema 2.9.

Capítulo 7

Conclusões

Os experimentos realizados sobre os algoritmos estocásticos estudados nos permitiram avaliar em situações práticas diferenciadas os pontos fortes e fracos de cada um dos métodos. Em particular, a utilização comum do GENCAN na Etapa de Refinamento garantiu que o diferencial de cada método ocorresse na Etapa de Construção. Neste aspecto, utilizando o método *Multistart* como referencial destacamos os desempenho dos métodos para cada conjunto de problemas.

O método *Random Linkage* mostrou-se muito bom na resolução de problemas tanto de pequeno como de grande porte. Nos testes do Capítulo 5 foi o segundo método mais eficiente e nos testes do Capítulo 6 teve um desempenho muito bom, chegando a ser método mais eficiente para 6 problemas de empacotamento.

O método de *Tunneling* utilizando a curva de *Lissajous* mostrou-se eficiente para problemas com um número reduzido de variáveis aplicados a uma região factível pequena. Isto pode ser verificado pelo seu desempenho nos testes A.3, A.4, A.7, A.10, A.13, A.22, A.23, A.29, A.6, A.12. Por outro lado, analisando a função A.5 verificamos que o desempenho do método de *Tunneling* foi melhor que o *Multistart* apenas no teste com menor número de variáveis ($n = 5$) tendo o seu desempenho piorado com o aumento deste número ($n \in \{8, 10, 20, 30\}$).

Por último, analisamos o desempenho do C-GRASP. Tendo sido o método mais eficiente nos testes realizados no Capítulo 5 não conseguiu repetir sua performance nos testes de empacotamento (Capítulo 6). Este resultado é explicado pelo tempo consumido pela sua Etapa de Construção que aumenta sensivelmente com o incremento do número de variáveis do problema.

A partir dos resultados apresentados pelos experimentos, não foi possível definir um método único na resolução dos problemas avaliados neste texto. Não obstante, destacamos a eficiência do C-GRASP na resolução dos problemas do Capítulo 5, sensivelmente superior aos demais métodos analisados e o bom desempenho do *Random Linkage* em ambos os conjuntos de testes realizados. Por último, ressaltamos o desempenho levemente superior do *Multistart* nos testes com problemas de empacotamento.

Apêndice A

Funções utilizadas nos experimentos

As funções abaixo foram retiradas de [18]:

1. **Função *quartic***

$$f(x) = \frac{x_1^4}{4} - x_1^2 + \frac{x_1}{10} + \frac{x_2^2}{10}, \tag{A.1}$$

onde $x \in \Omega = [-10, 10]^n$ e $f(x^*) = -0.352386$.

2. **Função *six hump***

$$f(x) = (4 - 2.1x_1^2 + x_1^3)x_1^2 + x_1x_2 + (4x_2^2 + 4)x_2^2, \tag{A.2}$$

onde $-3 \leq x_1 \leq 3$, $-2 \leq x_2 \leq 2$ e $f(x^*) = -1.0316285$.

3. **Função de Shubert**

$$f(x) = \prod_{i=1}^{n} \sum_{j=1}^{5} (jcos((j+1)x_i + j)), \tag{A.3}$$

onde $n = 2$, $x \in \Omega = [-10, 10]^n$ e $f(x^*) = -186.73091$.

4. **Função de Shubert Penalizada**

$$\begin{aligned} f(x) = & \prod_{i=1}^{n} \sum_{j=1}^{5} (jcos((j+1)x_i + j)) + \\ & + \beta((x_1 + 1.42513)^2 + (x_2 + 0.80032)^2), \end{aligned} \tag{A.4}$$

onde $n = 2$, $x \in \Omega = [-10, 10]^n$ e $f(x^*) = -186.73091$. Para a realização dos testes foi considerado $\beta \in \{0.5, 1.0\}$ referenciados como problemas A.4a e A.4b, respectivamente.

5. **Função de Piccioni**

$$f(x) = 10sen^2(\pi x_1) - \sum_{i=1}^{n-1} \left[(x_i - 1)^2(1 + 10sen^2(\pi x_{i+1}))\right] - (x_n - 1)^2, \tag{A.5}$$

onde $x \in \Omega = [-10, 10]^n$ e $f(x^*) = 0.0$. Foram realizados testes para $n \in \{5, 8, 10, 20, 30\}$.

As funções abaixo foram retiradas de [1]:

1. **Função de Levy**

$$\begin{aligned} f(x) =& sen^2(3\pi x_1) + \sum_{i=1}^{n-1} \left[(x_i - 1)^2(1 + sen^2(3\pi x_{i+1}))\right] + \\ &+ (x_n - 1)(1 + sen^2(2\pi x_n)), \end{aligned} \tag{A.6}$$

onde $n = 4$ e $x \in \Omega = [-10, 10]^n$. A função contém 7100 mínimos locais e $f(x^*) = -11.504403$.

2. **Função de Shubert**

$$f(x) = -\sum_{i=1}^{n}\sum_{j=1}^{5} j sen((j+1)x_i + j), \tag{A.7}$$

onde $n = 2$ e $x \in \Omega = [-10, 10]^n$. A função contém 400 mínimos locais e $f(x^*) = -24.062499$.

3.

$$f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i), \tag{A.8}$$

onde $n = 7$ e $x \in \Omega = [-5, 2]^n$. A função contém 2^n mínimos locais e $f(x^*) = -274.16316$.

4. **Função de Hansen**

$$f(x) = \sum_{i=1}^{5}(i cos((i-1)x_1 + i) \sum_{j=1}^{5}(j cos((j+1)x_2 + j), \tag{A.9}$$

onde $n = 2$ e $x \in \Omega = [-10, 10]^n$. A função contém 760 mínimos locais e $f(x^*) = -176.541793$.

5.

$$\begin{aligned} f(x) =& e^{sen(50x_1)} + sen(60e^{x_2}) + sen(70 sen(x_1)) + sen(sen(80x_2)) - \\ &- sen(10(x_1 + x_2)) + \frac{(x_1^2 + x_2^2)}{4}, \end{aligned} \tag{A.10}$$

onde $x_1,\ x_2 \in \Omega = [-1, 1]^n$. A função contém 2400 mínimos locais e $f(x^*) = -3.306868647475$.

6. **Função de Hartman**

$$f(x) = -\sum_{i=1}^{4} c_i e^{-\sum_{j=1}^{n} A_{ij}(x_j - P_{ij})^2} \tag{A.11}$$

onde $x \in \Omega = [0,1]^n$. Para $n = 3$ e, dados os seguintes valores

$$A = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}, \quad P = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4378 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix} \quad \text{e} \quad c = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \end{pmatrix},$$

o valor do mínimo global corresponde a $f(x^*) = -3.861305797098$ [1].

7. **Função de Griewank**

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \tag{A.12}$$

onde $n = 2$ e $x \in \Omega = [-500, 700]^n$. A função contém mais de 500 mínimos locais e $f(x^*) = 0.0$.

8.

$$f(x) = \sum_{i=1}^{m} [2 + 2i - (e^{ix_1} + e^{ix_2})]^2, \tag{A.13}$$

onde $x_1,\ x_2 \in \Omega = [-1000, 5]^n$. Para $m = 10$, $f(x^*) = 124.3621823719$.

9.

$$f(x) = \sum_{i=1}^{11} \left[y_i - \frac{x_1 u_i (u_i + x_2))}{(u_i(u_i + x_3) + x_4)} \right]^2, \tag{A.14}$$

onde $x \in \Omega = [-1000, 1000]^n$. Para $n = 4$ e $u_i = \frac{1}{b_i}$, $b = (0.25, 0.5, 1, 2, 4, 6, 8, 10, 12, 14, 16)^T$ e $y = (0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342d0, 0.0323, 0.0235, 0.0246)^T$, o mínimo global esperado é de $f(x^*) = 0.000307486$.

10.

$$f(x) = \sum_{i=1}^{m} \left[(x_1 + t_i x_2 - e^{t_i})^2 + (x_3 + x_4 sen(t_i) - cos(t_i))^2 \right]^2, \tag{A.15}$$

onde $x \in \Omega = [-100, 100]^n$. Para $m = 20$ e $t_i = \frac{i}{5}$, o mínimo global esperado é de $f(x^*) = 85822.20171974$.

11.

$$f(x) = \frac{1}{400} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \tag{A.16}$$

onde $n = 5$ e $x \in \Omega = [-10, 10]^n$ e $f(x^*) = 0.0$.

As funções abaixo foram retiradas de [20]:

1.

$$f(x) = (x_1^2 + x^2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{A.17}$$

onde $x \in \Omega = [-6, 6]^n$ e $f(x^*) = 0.0$.

2.

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{A.18}$$

onde $x \in \Omega = [-2, 2]^n$ e $f(x^*) = 0.0$.

3.

$$f(x) = -\sum_{i=1}^{m}[(x - a_i)^T(x - a_i) + c_i]^{-1} \tag{A.19}$$

onde $n = 4$ e $x \in \Omega = [0, 10]^n$. Para $m = 5$ o valor da função objetivo no mínimo global é $f(x^*) = -10.1531957$ (A.19a); para $m = 10$, $f(x^*) = -10.5362836$ (A.19b). Para tanto, foram utilizados os seguintes valores para os vetores a_i e as constantes c_i:

i	a_i				c_i	i	a_i				c_i
1	4.0	4.0	4.0	4.0	0.1	6	2.0	9.0	2.0	9.0	0.6
2	1.0	1.0	1.0	1.0	0.2	7	5.0	5.0	3.0	3.0	0.3
3	8.0	8.0	8.0	8.0	0.2	8	8.0	1.0	8.0	1.0	0.7
4	6.0	6.0	6.0	6.0	0.4	9	6.0	2.0	6.0	2.0	0.5
5	3.0	7.0	3.0	7.0	0.4	10	7.0	3.6	7.0	3.6	0.5

4.

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4, \tag{A.20}$$

onde $x \in \Omega = [-3, 3]^n$ e $f(x^*) = 0.0$.

5.

$$f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2, \tag{A.21}$$

onde $-3 \leq x_1 \leq 3$, $-2 \leq x_2 \leq 2$ e $f(x^*) = -1.031628$.

6.

$$f(x) = \left[\sum_{i=1}^{5} icos((i+1)x_1 + i)\right]\left[\sum_{i=1}^{5} icos((i+1)x_2 + i)\right], \tag{A.22}$$

onde $x \in \Omega = [-10, 10]^n$ e $f(x^*) = -186.7309$.

7.

$$f(x) = \frac{1}{2}\sum_{i=1}^{n}(x_i^4 - 16x_i^2 + 5x_i), \tag{A.23}$$

onde $x \in \Omega = [-20, 20]^n$. Para $n = 2$, $f(x^*) = -78.332331$; $n = 3$, $f(x^*) = -117.4984$; $n = 4$, $f(x^*) = -156.66466$.

8.

$$f(x) = 0.5x_1^2 + 0.5(1 - cos(2x_1)) + x_2^2, \tag{A.24}$$

onde $x \in \Omega = [-5, 5]^n$ e $f(x^*) = 0.0$.

9.

$$f(x) = 10x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-1}(x_1^2 + x_2^2)^4, \tag{A.25}$$

onde $x \in \Omega = [-5, 5]^n$ e $f(x^*) = -0.407461$.

10.

$$f(x) = 10^2x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-2}(x_1^2 + x_2^2)^4, \tag{A.26}$$

onde $x \in \Omega = [-5, 5]^n$ e $f(x^*) = -18.058697$.

11.

$$f(x) = 10^3x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-3}(x_1^2 + x_2^2)^4, \tag{A.27}$$

onde $x \in \Omega = [-20, 20]^n$ e $f(x^*) = -227.765747$.

12.

$$f(x) = 10^4x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-4}(x_1^2 + x_2^2)^4, \tag{A.28}$$

onde $x \in \Omega = [-20, 20]^n$ e $f(x^*) = -2429.414749$.

13.

$$f(x) = x_1^2 + x_2^2 - cos(18x_1) - cos(18x_2), \tag{A.29}$$

onde $x \in \Omega = [-5, 5]^n$ e $f(x^*) = -2.0$.

14.

$$f(x) = \left(x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right) cos(x_1) + 10, \tag{A.30}$$

onde $x \in \Omega = [-20, 20]^n$ e $f(x^*) = 0.397887$.

15.

$$f(x) = -\left[\sum_{i=1}^{5} sen((i+1)x_1 + i\right], \tag{A.31}$$

onde $x \in \Omega = [-20, 20]^n$ e $f(x^*) = -3.372897$.

16.

$$f(x) = e^{(0.5(x_1^2+x_2^2-25))^2} + sen(4x_1 - 3x_2)^4 + 0.5(2x_1 + x_2 - 10)^2, \tag{A.32}$$

onde $x \in \Omega = [0, 6]^n$ e $f(x^*) = 1.0$.

17.

$$f(x) = x_1^6 - 15x_1^4 + 27x_1^2 + 250, \tag{A.33}$$

onde $x \in \Omega = [-5, 5]^n$ e $f(x^*) = 7.0$.

18.

$$\begin{aligned} f(x) =& 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + \\ &+ 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1), \end{aligned} \tag{A.34}$$

onde $x \in \Omega = [-3, 3]^n$ e $f(x^*) = 0.0$.

19.

$$f(x) = (1.5 - x_1(1 - x_2))^2 + (2.25 - x_1(1 - x_2^2))^2 + (2.625 - x_1(1 - x_2^3))^2, \tag{A.35}$$

onde $x \in \Omega = [0, 5]^n$ e $f(x^*) = 0.0$.

20.

$$f(x) = \sum_{i=1}^{10} (e^{-0.2i} + 2e^{-0.4i} - x_1 e^{-0.2x_2 i} - x_3 e^{-0.2x_4 i})^2, \tag{A.36}$$

onde $x \in \Omega = [-20, 7]^n$ e $f(x^*) = 0.0$.

21.

$$f(x) = \left[\sum_{i=1}^{n} \frac{x_i^2}{2^{i-1}} \right] + \left[\sum_{i=2}^{n} \frac{x_i x_{i-1}}{2^i} \right], \tag{A.37}$$

onde $x \in \Omega = [-20, 7]^n$ e $f(x^*) = 0.0$. Foram realizados testes para $n \in \{10, 20, 30, 40\}$.

22.

$$f(x) = \sum_{i=1}^{10} x_i^2, \tag{A.38}$$

onde $x \in \Omega = [-10, 7]^n$ e $f(x^*) = 0.0$.

23.

$$f(x) = \sum_{i=1}^{10} \lfloor x_i^2 + 0.5 \rfloor^2, \tag{A.39}$$

onde $x \in \Omega = [-10, 7]^n$ e $f(x^*) = 2.5$.

24.

$$f(x) = -20e^{-0.2\sqrt{0.1 \sum_{i=1}^{10} x_i^2}} - e^{0.1 \sum_{i=1}^{10} cos(2\pi x_i)} + 20 + e \tag{A.40}$$

onde $x \in \Omega = [-10, 20]^n$ e $f(x^*) = 0.0$.

25.

$$f(x) = sen(x_1) + sen\left(\frac{10x_1}{3}\right) + log_{10}(x_1) - 0.84x_1 \tag{A.41}$$

onde $x \in \Omega = [0.1, 6]^n$ e $f(x^*) = -5.534$.

Apêndice B

Fluxogramas dos algoritmos implementados

B.1 Multistart

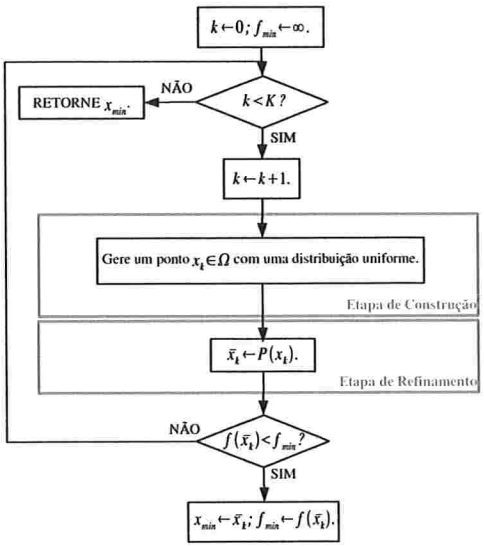


Figura B.1: Fluxograma do algoritmo *Multistart*.

B.2 Random Linkage

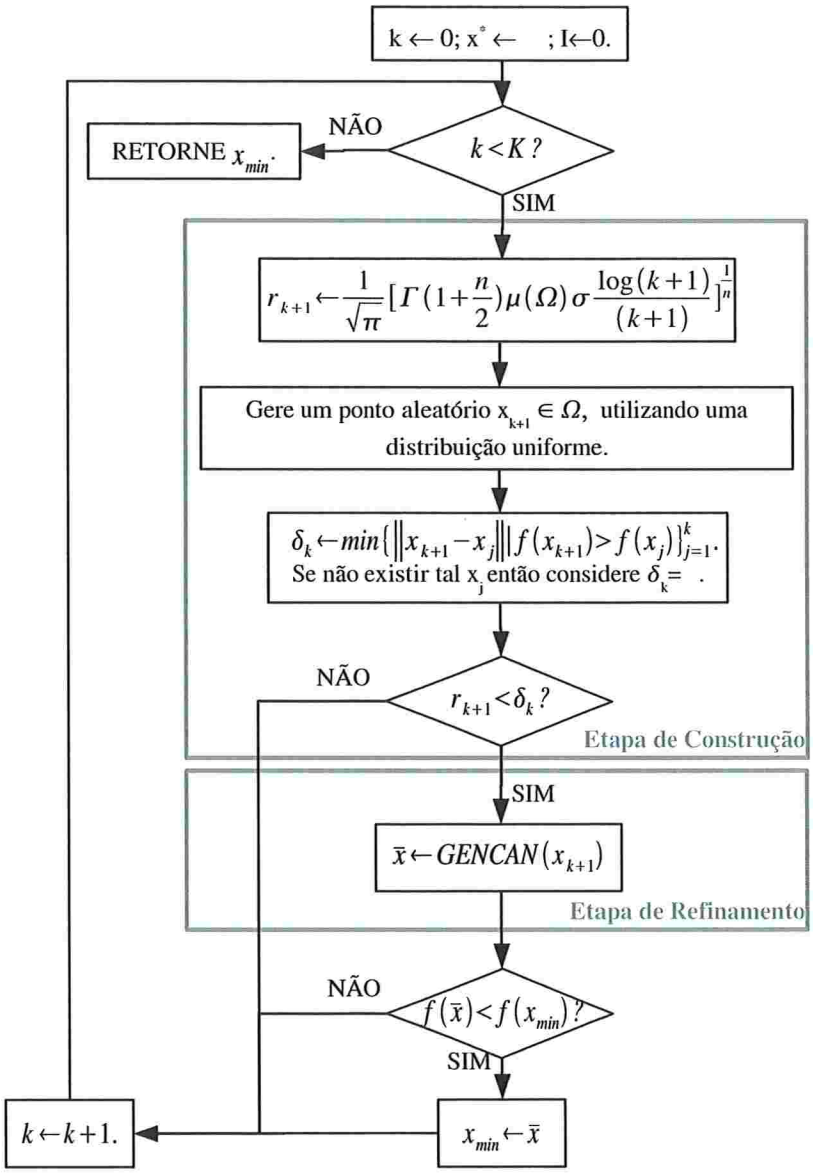


Figura B.2: Fluxograma do algoritmo *Random Linkage*.

B.3 *Tunneling*

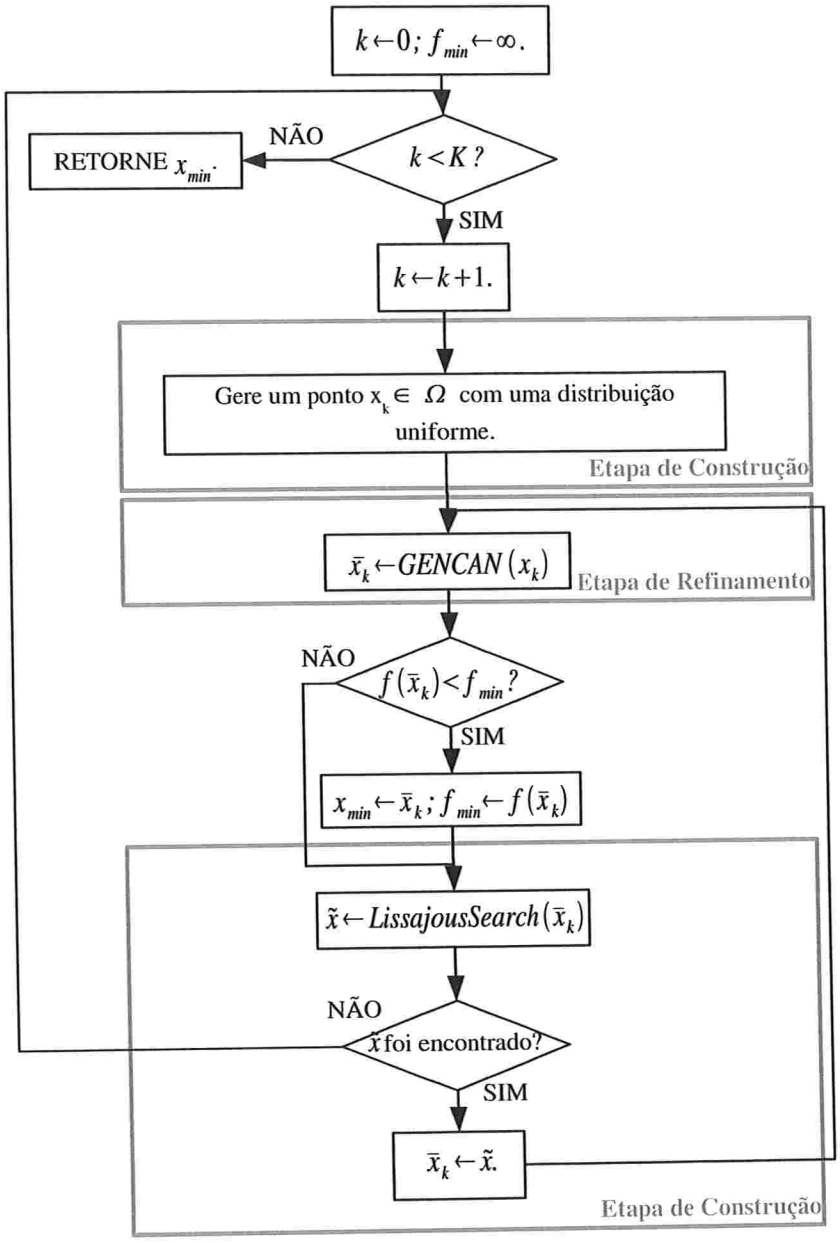


Figura B.3: Fluxograma geral do algoritmo de *Tunneling* utilizando a curva de *Lissajous*.

Procedimento *LissajousSearch*:

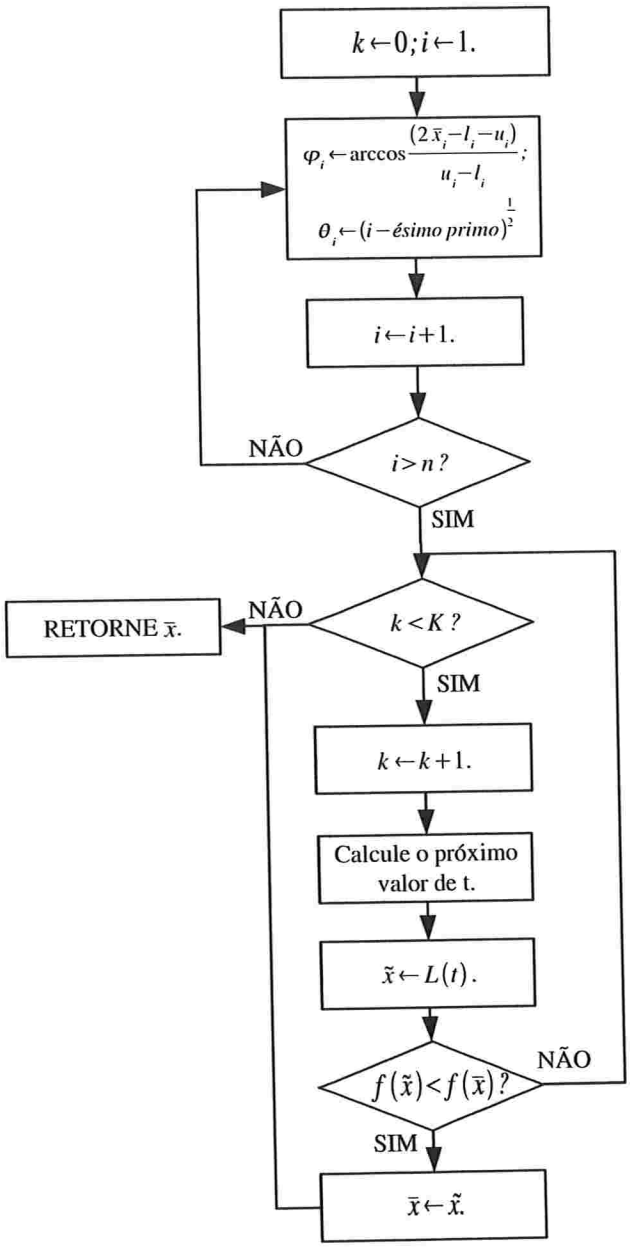


Figura B.4: Detalhe da Fase de *Tunneling* utilizando a curva de *Lissajous*.

B.4 GRASP

B.4.1 GRASP

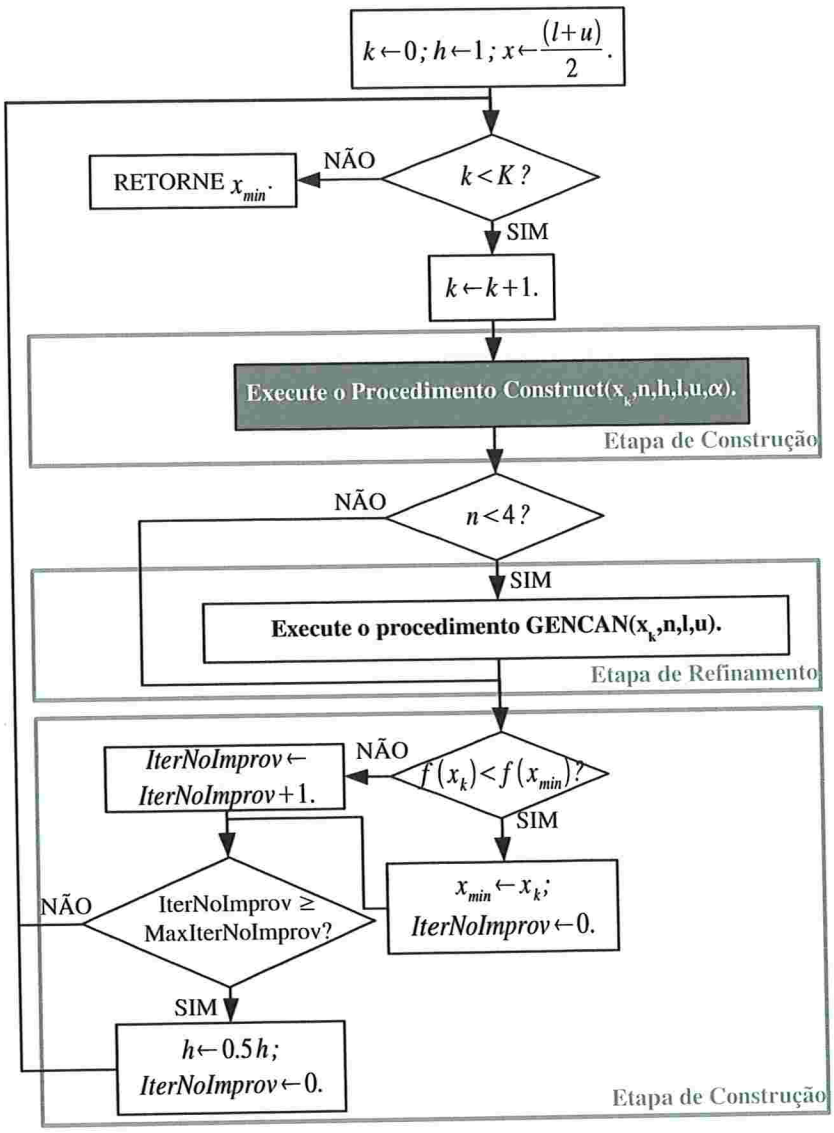


Figura B.5: Fluxograma geral do algoritmo *GRASP*.

Procedimento Construct:

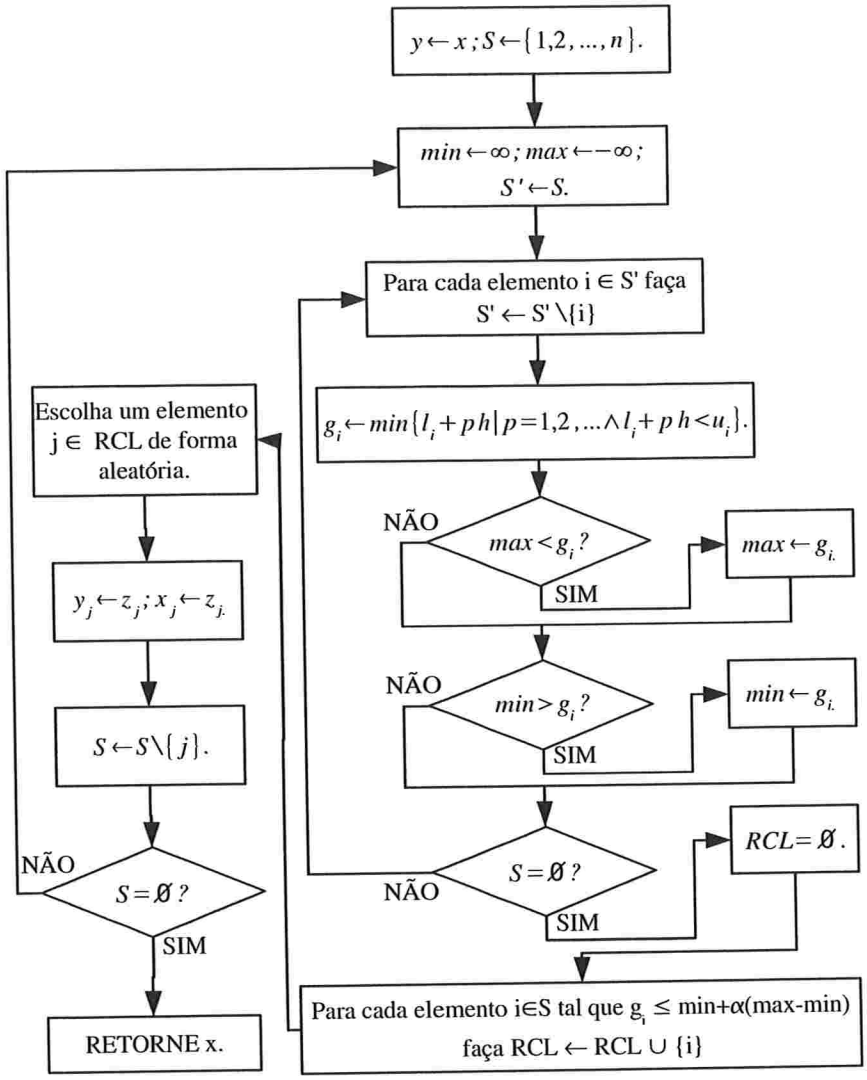


Figura B.6: Fluxograma da Etapa de Construção do *GRASP*.

B.4.2 GRASP contínuo

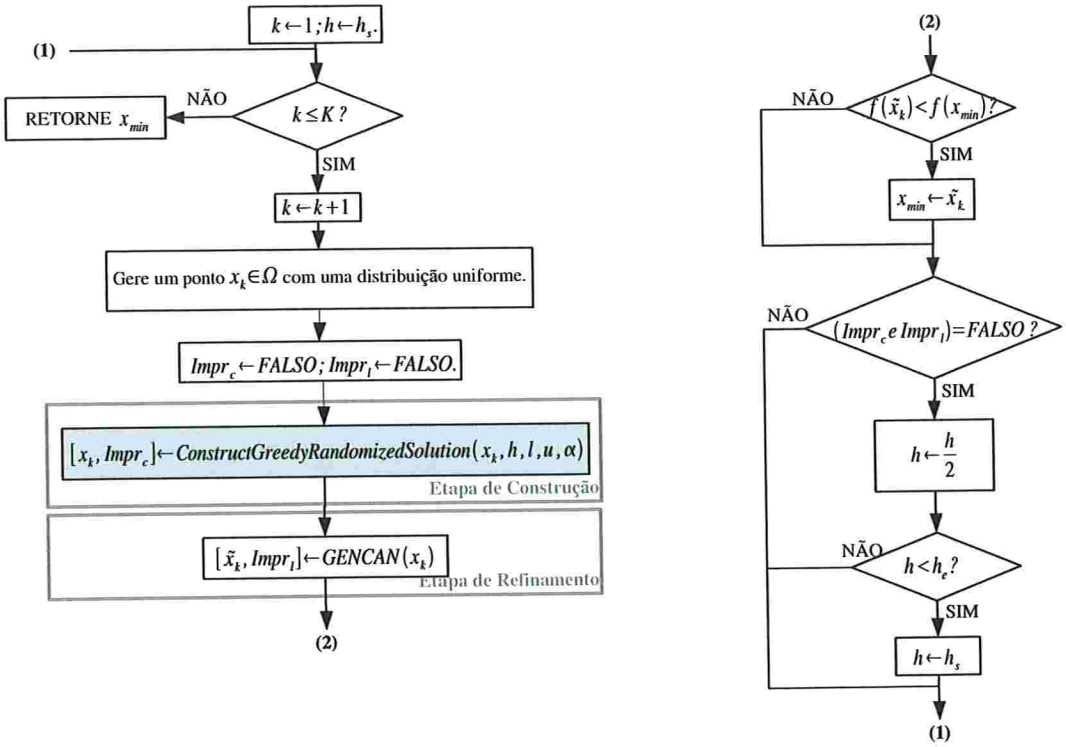


Figura B.7: Fluxograma geral do algoritmo GRASP contínuo.

Procedimento *ConstructGreedyRandomizedSolution*:

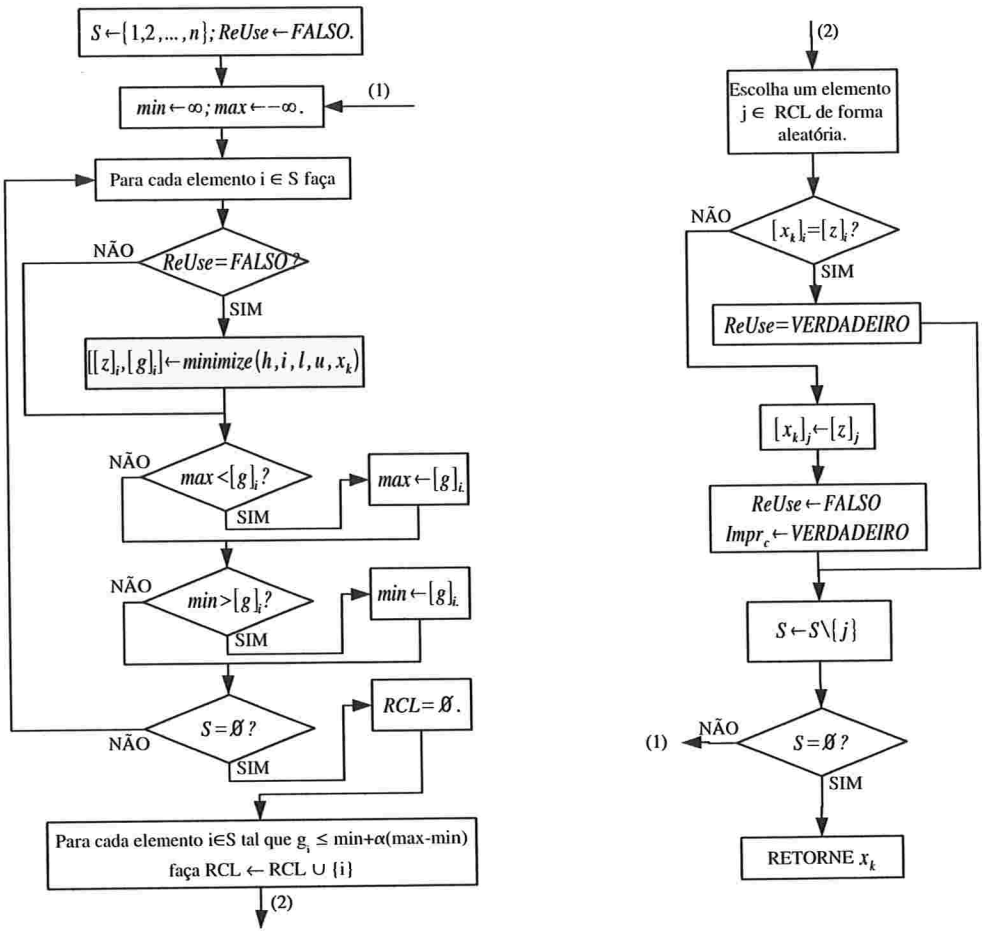


Figura B.8: Detalhes da Etapa de Construção do GRASP contínuo.

Procedimento *Minimize*:

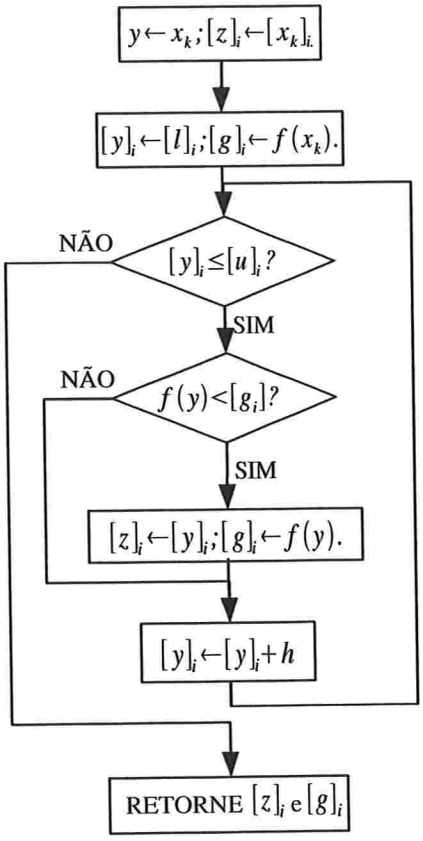


Figura B.9: Detalhes do procedimento *Minimize*.

Apêndice C

Comparação alternativa dos testes realizados no Capítulo 5

C.1 Comparativo dos resultados

De um modo geral, as soluções encontradas pelos métodos foram muito próximas. Dos 52 problemas avaliados, os métodos encontraram soluções equivalentes em 47 problemas.

Na Tabela C.1 comparamos o número de iterações utilizadas pelos métodos na resolução de cada um dos problemas. Os dados mostrados na tabela indicam a relação do número de iterações atingido por cada método com o melhor resultado encontrado para aquele problema. Desta forma, os resultados com valor igual a 1.00 refletem os métodos que encontraram a solução esperada no menor número de iterações. Os resultados da tabela mostram um desempenho muito superior das duas versões do GRASP em comparação aos demais métodos. Verificamos ainda que todos os métodos analisados tiveram desempenho médio superior ao *Multistart*.

Na Tabela C.2 fizemos comparação semelhante com o tempo gasto pelos métodos para encontrar a solução desejada. Visando evitar divisão por zero, consideramos os tempos encontrados pelos métodos da seguinte forma $max\{0.01, t_{x^*}\}$. Os resultados mostram um desempenho superior do C-GRASP-GENCAN seguido pelo GRASP-GENCAN.

Associando os resultados das duas tabelas apresentadas identificamos, por exemplo, que o *Multistart* precisou, em média, de 15.6 vezes mais iterações para encontrar a mesma solução que o C-GRASP-GENCAN, mas realizou este cálculo, em média, em apenas 1.8 vezes a mais de tempo que o C-GRASP-GENCAN. Estes dados indicam que, apesar da eficiência da Etapa de Contrução do GRASP em encontrar pontos mais próximos a vizinhança de um mínimizador global, este processo consome um tempo considerável para ser realizado.

Os resultados dos testes, também apresentados de forma sintética na Tabela C.3, mostram que os algoritmos C-GRASP-GENCAN, GRASP-GENCAN, *Random Linkage* ($\sigma \in \{1, 2\}$) e *Multistart* foram os que apresentaram, na média, melhor desempenho para o conjunto de testes realizados.

Problema	n	MST	RL(1)	RL(2)	RL(4)	LIS Orig	LIS Seq	GRA-GEN	C-GRA-GEN
A.1	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.2	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.3	2	31.00	19.00	17.00	11.00	2.00	2.00	1.00	1.00
A.4a	2	14.30	25.20	22.70	10.10	2.50	2.20	2.20	1.00
A.4b	2	20.83	6.67	4.83	3.83	1.17	1.33	1.33	1.00
A.5	5	27.00	44.00	53.00	36.00	3.00	14.00	1.00	1.00
A.5	8	27.00	24.00	33.00	31.00	33.00	36.00	1.00	1.00
A.5	10	4.00	4.00	4.00	4.00	4.00	4.00	1.00	1.00
A.5	20	50.00	50.00	49.00	48.00	50.00	50.00	1.00	1.00
A.5	30	6.00	6.00	5.00	5.00	6.00	6.00	1.00	1.00
A.6	4	–	–	–	–	–	–	–	–
A.7	2	18.50	9.25	7.50	5.75	1.25	1.00	1.50	1.25
A.8	7	3.00	3.00	3.00	3.00	3.00	3.00	1.00	1.00
A.9	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.10	2	33.16	11.95	7.72	34.98	1.00	2.74	2.51	10.79
A.11	3	1.00	1.00	1.00	1.00	1.00	1.00	16.00	3.00
A.12	2	–	–	–	–	–	–	–	–
A.13	2	–	–	–	–	–	–	–	–
A.14	4	–	–	–	–	–	–	–	–
A.15	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.16	5	2.17	1.50	1.17	1.08	2.67	2.83	1.00	2.25
A.17	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.18	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.19a	4	3.00	3.00	3.00	3.00	3.00	3.00	1.00	1.00
A.19b	4	3.00	1.00	3.00	3.00	3.00	3.00	1.00	1.00
A.20	4	1.00	2.00	1.00	1.00	1.00	1.00	1.00	1.00
A.21	2	2.00	2.00	2.00	2.00	2.00	2.00	1.00	1.00
A.22	2	31.00	19.00	17.00	11.00	2.00	2.00	1.00	1.00
A.23	2	1.50	1.50	1.50	1.50	1.50	1.00	1.00	1.00
A.23	3	3.00	3.00	3.00	3.00	3.00	2.00	1.00	1.00
A.23	4	20.00	16.00	17.50	28.50	20.00	1.50	1.00	1.00
A.24	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.25	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.26	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.27	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.28	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.29	2	66.50	23.50	15.50	10.00	201.50	5.50	1.00	1.00
A.30	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.31	1	5.00	5.00	3.00	2.00	2.00	2.00	1.00	1.00
A.32	2	1.64	1.27	1.09	2.45	2.82	3.27	2.09	1.00
A.33	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.34	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.35	2	2.00	2.00	2.00	2.00	2.00	2.00	1.00	1.00
A.36	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.37	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.37	20	1.00	1.00	1.00	1.00	1.00	1.00	3.00	3.00
A.37	30	504.50	498.00	494.00	487.50	504.50	504.50	1.00	1.00
A.37	40	74.00	74.00	74.00	74.00	74.00	74.00	1.00	1.00
A.38	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.39	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.40	10	–	–	–	–	–	–	–	–
A.41	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Média		20.77	18.66	18.44	17.99	20.27	16.02	1.46	1.33

Tabela C.1: Comparativo do número de iterações realizadas por cada método para um conjunto de problemas propostos em [18, 1]. Os números acima indicam o número de vezes de iterações que um método precisou para encontrar uma boa aproximação do minimizador global, em comparação ao método mais eficiente.

Problema	n	MST	RL(1)	RL(2)	RL(4)	LIS Orig	LIS Seq	GRA-GEN	C-GRA-GEN
A.1	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.2	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.3	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.4a	2	6.00	10.00	9.00	4.00	2.00	2.00	2.00	1.00
A.4b	2	6.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.5	5	2.00	4.00	5.00	4.00	1.00	4.00	1.00	1.00
A.5	8	3.00	2.00	3.00	4.00	23.00	25.00	1.00	1.00
A.5	10	1.00	1.00	1.00	1.00	3.00	3.00	1.00	1.00
A.5	20	15.00	15.00	14.50	14.00	53.00	53.00	2.00	1.00
A.5	30	2.29	2.29	1.71	1.86	4.00	4.00	1.43	1.00
A.6	4	–	–	–	–	–	–	–	–
A.7	2	2.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.8	7	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.9	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.10	2	9.50	3.50	2.00	20.17	1.00	2.67	1.00	1.83
A.11	3	1.00	1.00	1.00	1.00	1.00	1.00	2.00	1.00
A.12	2	–	–	–	–	–	–	–	–
A.13	2	–	–	–	–	–	–	–	–
A.14	4	–	–	–	–	–	–	–	–
A.15	4	1.00	1.00	1.00	1.00	1.00	1.00	2.00	2.00
A.16	5	1.00	1.00	1.00	1.00	10.00	11.00	2.00	1.00
A.17	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.18	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.19a	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.19b	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.20	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.21	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.22	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.23	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.23	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.23	4	2.00	2.00	2.00	4.00	12.00	1.00	1.00	1.00
A.24	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.25	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.26	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.27	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.28	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.29	2	4.00	1.00	1.00	1.00	34.00	1.00	1.00	1.00
A.30	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.31	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.32	2	2.00	1.00	1.00	2.00	6.00	7.00	1.00	1.00
A.33	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.34	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.35	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.36	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.37	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.37	20	1.00	1.00	1.00	1.00	1.00	1.00	6.00	5.00
A.37	30	10.92	11.23	11.00	11.23	138.62	138.62	1.15	1.00
A.37	40	1.00	1.00	1.04	1.04	14.15	14.15	1.42	1.08
A.38	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.39	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
A.40	10	–	–	–	–	–	–	–	–
A.41	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Média		2.12	1.89	1.86	2.20	7.14	6.39	1.23	1.13

Tabela C.2: Comparativo do tempo gasto por cada método para um conjunto de problemas propostos em [18, 1]. Os números acima indicam quantas vezes um método demorou, em comparação aos demais, para encontrar uma boa aproximação do minimizador global.

Problema	n	MST	RL(1)	RL(2)	RL(4)	LIS Orig	LIS Seq	GRA-GEN	C-GRA-GEN
Iterações relativas		20.77	18.66	18.44	17.99	20.27	16.02	1.46	1.33
Tempos relativos		2.12	1.89	1.86	2.20	7.14	6.39	1.23	1.13

Tabela C.3: Sintético com as médias apresentadas nas Tabelas C.1 e C.2.

Referências Bibliográficas

[1] I. G. Akrotirianakis, C. A. Floudas. Computational Experience with a new class of convex underestimators: box-constrained NLP problems, *Journal of Global Optimization* 29, pp. 249-264, 2004.

[2] R. Andreani, E. G. Birgin, J. M. Martínez, M. L. Schuverdt. Augmented Lagrangian methods under the Constant Positive Linear Dependence constraint qualification, por aparecer em *Mathematical Programming*, 2006. (DOI: 10.1007/s10107-006-0077-1)

[3] R. Andreani, E. G. Birgin, J. M. Martínez, M. L. Schuverdt. On Augmented Lagrangian methods with general lower-level constraints, Technical Report MCDO-050304, Department of Applied Mathematics, UNICAMP, Brazil, 2005.

[4] G. de Barra. *Introduction to Measure Theory*, Van Nostrand Reinhold Company Ltd, Great Britain, 1974.

[5] E. G. Birgin, J. M. Martínez. Large-scale active-set box-constrained optimization method with spectral projected gradients, *Computational Optimization and Applications* 23, pp. 101-125, 2002.

[6] E. G. Birgin, J. M. Martínez, D. P. Ronconi, Optimizing the packing of cylinders into a rectangular container: a nonlinear approach, *European Journal of Operational Research* 160, pp. 19-33, 2005.

[7] C. G. E. Boender, H. E. Romeijn. Stochastic methods em *Handbook of Global Optimization*, pp. 829-869, Kluwer Academic Publishers, Holanda, 1995.

[8] V. Chvátal. A greedy heuristic for the set convering problem, *Mathematics of Operations Research* 4, pp. 233-235, 1979.

[9] L. C. W. Dixon, E. Spedicato, G. P. Szegö. *Nonlinear optimisation, theory and algorithms*, Birkhauser Boston, 1980.

[10] E. D. Dolan, J. J. Moré. Benchmarking optimization software with performance profiles, *Mathematical Programming* 91, pp. 201-213, 2002.

[11] T. A. Feo, M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem, *Operations Research Letters* 8, pp. 67-71, 1989.

[12] T. A. Feo, M. G. C. Resende. Greedy randomized adaptive search procedures, *Journal of Global Optimization* 6, pp.109-133, 1995.

[13] H. L. Guidorizzi. *Um curso de Cálculo* - Volume 2 (5a. edição), LTC Editora, Brasil, 2004.

[14] M. J. Hirsch, C. N. Meneses, P. M. Pardalos, M. G. C. Resende. Global optimization by continuous GRASP, Technical Report TD-6MPUV9. AT&T Labs Research, 2006.

[15] M. J. Hirsch, P. M. Pardalos, M. G. C. Resende. Speeding up continuous GRASP, Technical Report TD-6U2P2H, AT&T Shannon Laboratory, 2006.

[16] A. V. Levy, S. Gomez. The Tunneling method applied to global optimization, em *Numerical Optimization 1984* (Ed. P. T. Bogus), Philadelphia, SIAM, pp. 213-244, 1985.

[17] A. V. Levy, A. Montalvo. The Tunneling algorithm for the global minimization of functions, *SIAM Journal on Scientific Computing* 6, pp. 15-29, 1985.

[18] M. Locatelli, F. Schoen. Numerical experience with Random Linkage algorithms for global optimisation, Technical Report 15-98, Dip. di Sistemi e Informatica, Università di Firenze, Italy, 1998.

[19] M. Locatelli, F. Schoen. Random Linkage: A family of acceptance/rejection algorithms for global optimization, *Mathematical Programming* 85, pp. 379-396, 1999.

[20] C. N. Meneses, P. M. Pardalos, M. G. C. Resende. GRASP for nonlinear optimization, Technical Report TD-6DUTRG, *AT&T Labs Research*, 2005.

[21] P. M. Pardalos, Fabio Schoen. Recent advances and ternds in global optimization: Deterministic and stochastic methods, *Proceedings of the Sixth International Conference on Foundations of Computer-Aided Process Design*, Austin, TX, pp. 119-131, 2004.

[22] M. G. C. Resende, C. C. Ribeiro. Greedy randomized adaptive search procedures em *Handbook of Metaheuristics* (eds. F. Glover e G. Kochenberger), Kluwer, pp. 219-250, 2002.

[23] M. Salvatierra. *Otimização global usando trajetórias densas e aplicações*, tese de Doutorado, Departamento de Matemática Aplicada, IMECC, UNICAMP, 2005.

[24] A. H. G. Rinnooy-Kan, G. Timmer. Stochastic global optimization methods. part I: Clustering methods, *Mathematical Programming* 39, pp. 27-56, 1987.

[25] A. H. G. Rinnooy-Kan, G. Timmer. Stochastic global optimization methods. part II: Multi level methods, *Mathematical Programming* 39, pp. 57-78, 1987.