Universidade Federal da Paraíba

Centro de Informática

Programa de Pós-Graduação em Informática

Novas Abordagens Sequencial e Paralela da meta-heurística C-GRASP Aplicadas à Otimização Global Contínua

Lisieux Marie Marinho dos Santos Andrade

João Pessoa - Paraíba

Agosto, 2013

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Novas Abordagens Sequencial e Paralela da meta-heurística C-GRASP Aplicadas à Otimização Global Contínua

Lisieux Marie Marinho dos Santos Andrade

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Computação Distribuída

Lucídio dos Anjos Formiga Cabral
(Orientador)

João Pessoa, Paraíba, Brasil

A553n Andrade, Lisieux Marie Marinho dos Santos.
Novas abordagens sequencial e paralela da meta-heurística C-GRASP aplicadas à Otimização Global Contínua / Lisieux Marie Marinho dos Santos Andrade.-- João Pessoa, 2013.
99f. : il.
Orientador: Lucídio dos Anjos Formiga Cabral
Dissertação (Mestrado) – UFPB/CI
1. Informática. 2. Ciência da computação. 3. Otimização Global Contínua. 4. C-GRASP. 5. C-GGVNS. 6. CUDA.

UFPB/BC CDU: 004(043)

A minha família, meu alicerce e porto seguro.

Agradecimentos

Todas as conquistas alcançadas não seriam possíveis sem o amparo amigo, o estímulo e o companheirismo que tanto recebi na minha trajetória acadêmica, e que por isto venho a agradecer...

A Deus por proporcionar meu encontro com pessoas maravilhosas e por me permitir vivenciar momentos enriquecedores.

A minha Mãe e ao meu Pai (*in memoriam*), por promover o melhor ambiente familiar regado de carinho, atenção, amparo, amor, respeito, companherísmo e sobretudo confiança. A minha melhor amiga, minha irmã querida, que me apoia em todos os momentos e a quem tenho como exemplo, junto a nossos Pais.

Ao meu amado Hilber, pela cumplicidade, companheirismo, por todos estes anos de crescimento, compreendendo minhas faltas, apoiando e incentivando meus planos.

Ao meu querido orientador, Professor Lucídio dos Anjos Formiga Cabral, pelo tempo dedicado desde minha graduação, em reuniões primorosas, tornando problemas nebulosos em simples desafios. Acreditando sempre no meu potencial e apresentando novos horizontes.

Aos Professores Andrei de Araújo Formiga e Roberto Quirino do Nascimento, pela atenção em sala de aula e principalmente em reuniões e contribuições fundamentais para o desenvolvimento desta pesquisa.

A Thayse Christine Souza Dias, por meus primeiros passos na área, por toda a amizade, apoio, atenção e carinho.

A Thatyana Carla Souza Dias e Luiz Mauricio Fraga Martins, pela confiança em meu trabalho e pela amizade construída.

Aos Professores Gilberto Farias de Sousa Filho e Tiago Maritan Ugulino de Araújo pelo acompanhamento inicial desta pesquisa.

Aos meus colegas de curso e da UFPB Virtual pelo apoio, partilha de conhecimentos e aflições neste período.

A Leandro Figueiredo e Raphael Xavier pelas longas horas em laboratório aperfeiçoando este trabalho.

A todos vocês, meus sinceros agradecimentos.

Resumo

O presente trabalho aborda o Problema de Otimização Global Contínua, em sua forma de minimização, através de duas abordagens para o procedimento *Continuous Greedy Randomized Adaptive Search Procedure* (C-GRASP). A elaboração do primeiro método, sequencial e híbrido, parte da deficiência presente nas abordagens atuais, em promover boa exploração no espaço de vizinhança. Sendo constituída da combinação de duas meta-heurísticas, C-GRASP padrão e *Continuous General Variable Neighborhood Search* (C-GVNS). Como estratégia para a realização de trocas sistemática de estruturas de vizinhanças, mostrou-se eficiente aos testes computacionais realizados. O segundo procedimento elaborado parte do grande consumo de tempo ao utilizar funções com alta dimensão, pelo procedimento de construção do método C-GRASP padrão. Como os problemas de otimização possuem crescimento elevado de dimensionalidade, é desejável ter versões paralelas do método de otimização para lidar com os problemas maiores. Desta forma, para o novo procedimento elaborado foi empregado a plataforma de computação paralela *Compute Unified Device Architecture* (CUDA), que, conforme verificado nos experimentos realizados, promoveu promissora aceleração quanto ao tempo de processamento.

Palavras-chave: Otimização Global Contínua, *Continuous Greedy Randomized Adaptive Search Procedure*, C-GGVNS, *Compute Unified Device Architecture*.

Abstract

The present work deals with the Continuous Global Optimization Problem, in its minimization form, by testing two approaches for the Continuous Greedy Randomized Adaptive Search Procedure (C-GRASP). The development of the first method - sequential and hybrid - comes from the deficiency of current approaches to provide a good neighborhood space exploration. Being constructed from the combination of two meta-heuristics, standard C-GRASP and Continuous General Variable Neighborhood Search (C-GVNS), as a strategy to achieving symmetric trades of neighborhood structures, it performed efficiently in the computational tests that were taken. The second procedure arises from the large consume of time when using high dimension functions with the standard C-GRASP construction procedure. As the optimization problems have a high dimensionality increase, it's preferable to have two parallel versions of the optimization method in order to handle bigger problems. Thus, for this new procedure developed, it was used the Compute Unified Device Architecture (CUDA), which provided promising acceleration regarding the processing time, based on the experiments performed.

Keywords: Continuous Global Optimization, Continuous Greedy Randomized Adaptive Search Procedure, C-GGVNS, Compute Unified Device Architecture.

Conteúdo

Lista de Símbolos

APS : *Adaptive Pattern Search*

C-GRASP : *Continuous Greedy Randomized Adaptive Search Procedure*

C-GVNS : *Continuous General Variable Neighborhood Search*

CUDA :*Compute Unified Device Architecture*

EC-GRASP : *Enhanced Continuous Greedy Randomized Adaptive Search Procedure*

GRASP : *Greedy Randomized Adaptive Search Procedure*

ILS : *Iterated Local Search*

LCR : Lista de Candidatos Restrita

POGC : Programa de Otimização Global Contínuo

SA : *Simulated Annealing*

VNS : *Variable Neighborhood Search*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O surgimento de problemas recorrentes impõe aos indivíduos observadores o instinto da percepção e da busca por soluções viáveis, caracterizando assim os objetivos e fazendo surgir a concepção do problema como um sistema. Dessa forma, a administração de transporte, a dinâmica molecular, o alinhamento de proteínas, o sequenciamento de produção, entre outros, são exemplos de problemas reais que requerem processos ágeis, capazes de trabalhar com suas complexidades e utilizar técnicas otimizadas para obtenção de soluções viáveis. Alinhado a esta necessidade, o desenvolvimento de métodos de otimização, que produzam soluções com qualidade e custo de produção reduzido, tem obtido crescimento elevado.

A Otimização é uma área da computação e da matemática, que tem por objetivo encontrar valores adequados, que minimizem ou maximizem os valores da função objetivo definida sobre um domínio. Desta maneira, sua classificação pode ser dada como: otimização discreta e otimização contínua. A diferença encontra-se justamente na natureza dos problemas tratados, problemas de otimização discreta possuem um conjunto finito de soluções, já problemas de otimização contínua possuem soluções de domínio contínuo.

A dificuldade existente no tratamento de tais problemas de otimização encontra-se na escolha do método adequado, uma vez que, estes, em sua maioria sendo convencionais ou exatos, configuram-se ineficientes pois não proporcionam uma convergência adequada para o encontro de uma boa solução. Outros métodos que utilizam cálculo do gradiente da função para definir direções adequadas de busca promovem um elevado esforço computacional, devido a algumas funções não serem diferenciáveis ou não contínuas.

Dentro deste cenário, pesquisadores tem intensificado o desenvolvimento de métodos

diversificados baseados principalmente na simulação de fenômenos da natureza, bem como ferramentas de inteligência artificial denominados meta-heurísticas, que por muito tempo foi aplicado à problemas de natureza discreta e que está ganhando aplicações no domínio contínuo.

Algumas meta-heurísticas que se destacam na resolução de problemas contínuos, são: *Greedy Randomized Adaptive Search Procedure* (GRASP), *Tabu Search* (Busca Tabu) e *Simulated Annealing* (SA), *Variable Neighborhood Search* (VNS), que não utilizam calculo de gradiente, e que apesar de possuirem uma convergência lenta quando comparados a métodos clássicos de programação não linear, são eficientes por escaparem de valores aparentemente satisfatórios (ótimos locais), comumente presentes em problemas desta natureza.

Tendo em vista estes aspectos, a comunidade científica tem intensificado o estudo de métodos híbridos com o intuito de minimizar o esforço computacional promovido pela convergência lenta. Desta forma, o presente trabalho, apresenta o C-GGVNS, uma combinação de duas meta-heurísticas, a *Greedy Randomized Adaptive Search Procedure* (C-GRASP) e a *General Variable Neighborhood Search* (GVNS), com finalidade de melhorar os resultados já existentes na literatura. E também a elaboração da primeira versão paralela de parte do procedimento C-GRASP, utilizando a plataforma *Compute Unified Device Architecture* (CUDA), sendo este um primeiro passo importante no caminho de uma versão totalmente paralela do C-GRASP, capaz de o encontrar eficientemente ótimas soluções.

1.1 Objetivos

1.1.1 Objetivo Geral

Realizar a construção de métodos heurísticos para a resolução de Problema de Otimização Global Contínua (POGC).

1.1.2 Objetivos Específicos

- Implementar a hibridização C-GGVNS aplicada à POGCs;
- Avaliar o impacto da hibridização no nível de esforço computacional, medido em termos de número de chamadas, ao procedimento de avaliação da função objetivo, du-

rante a busca pela solução ótima e em média do valor da função objetivo, quando estabelecido um número máximo de iterações;

- Implementar a paralelização da fase de construção do procedimento C-GRASP, através do uso da plataforma CUDA;
- Verificar o desempenho do procedimento C-GRASP com construção paralela em nível de tempo de processamento.

1.2 Estrutura do Trabalho

Esta dissertação está dividida em seis capítulos. O capítulo 2 apresenta os conceitos básicos sobre otimização combinatória, otimização contínua, métodos heurísticos e meta-heurísticas, com a descrição mais detalhada de alguns conceitos.

No terceiro capítulo, será realizada uma revisão sobre os principais trabalhos descritos na literatura relacionados ao presente estudo. Em seguida, o quarto capítulo apresenta o método proposto, os problemas testes e os parâmetros de avaliação que serão utilizados.

O capítulo 5, descreve os experimentos e as métricas utilizadas para a validação do método, além de apresentar e discutir os resultados alcançados. Por fim, no capítulo 6 serão tecidas as considerações finais e descritas algumas propostas de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

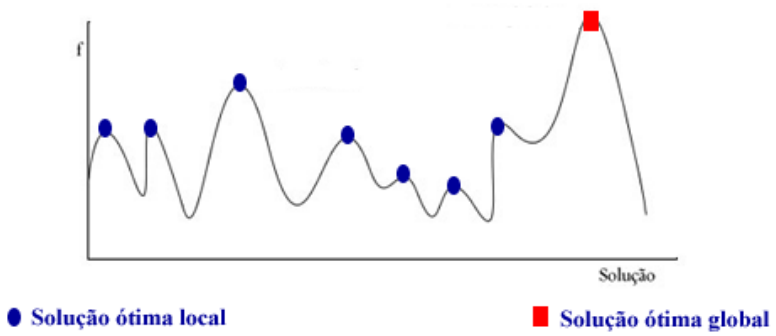
Este capítulo apresenta a base teórica que sustenta conceitualmente o tema em estudo. Para tanto, aborda e descreve os conceitos básicos da otimização contínua e das meta-heurísticas.

2.1 Otimização

A Otimização é definida como o procedimento pelo qual procura-se determinar as melhores soluções para problemas em sua maioria do mundo real (YANG, 2008). Habitualmente, tal determinação é realizada através de métodos de pesquisa que não dependem da experimentação envolvem a manipulação física do objeto em estudo. Em tais condições, as representações dos problemas e de seus comportamentos ocorrem através de modelos que assumem a forma de equações matemáticas, estruturadas com um conjunto S de variáveis s, também chamadas de soluções, e uma função definida $f : S \rightarrow \mathbb{R}$, conhecida como função objetivo, que deve ser minimizada ou maximizada de acordo com o objetivo do problema.

Considera-se uma solução ótima aquela que minimiza (ou maximiza) a função objetivo do modelo matemático. Há duas maneiras de classificar uma solução: ótima local e ótima global (Figura 2.1). Uma solução é considerada ótima local quando em seu espaço de vizinhança não existir nenhuma solução que melhore a função objetivo. Já a solução ótima global corresponde àquela que, em todo o conjunto de soluções possíveis, fornece à função objetivo o melhor valor (YANG, 2008).

Figura 2.1: Solução ótima local e global



Muitos problemas de otimização são classificados como NP-Difícil[1], ou seja, são intratáveis do ponto de vista computacional, e não existe algoritmo que os solucionem em tempo polinomial (PAPADIMITRIOU; STEIGLITZ, 1998). Dada esta limitação, surgiram os Algoritmos de Aproximação, que realizam de forma promissora a obtenção de soluções aproximadas ou quase ótimas, sendo decomposta em duas grandes classes: Heurísticas e Meta-heurísticas, que serão abordadas respectivamente nas seções 2.3 e 2.4.

Os problemas de Otimização, sejam eles NP-difícies ou não, podem possuir naturezas diferentes, apresentando-se como discretos ou contínuos. Problemas discretos, como o próprio nome revela, abordam variáveis discretas em seu espaço de soluções, possuindo normalmente valores inteiros. Em contraste, problemas de natureza contínua possuem variáveis que assumem quaisquer valores desde que respeitem as restrições descritas no problema.

2.2 Otimização Contínua

Um problema de Otimização Global no espaço contínuo (POGC) possui grande aplicações práticas no campo da Física Computacional, da minimização de energia em moléculas, da seleção de carteiras equilibradas entre um conjunto de possíveis ativos com o objetivo de maximizar os ganhos, entre outras aplicações (ANDRÉASSON; EVGRAFOV; PATRIKSSON, 2007). O objetivo é otimizar os parâmetros dos problemas, representados matematicamente por um vetor (x), onde a função objetivo $f(x)$ manipula os objetivos do problema, incorporando algumas restrições, conforme a equação 2.1.

[1] Também conhecido na literatura como NP-hard.

$$global \min_{x \in S} f(x) \tag{2.1}$$

$$\text{S.a: } g_i(x) \leq 0 \qquad \text{para i = 1,...,m} \tag{2.2}$$

$$h_i(x) = 0 \qquad \text{para i=1,...,l} \tag{2.3}$$

Sendo $f(x)$, $g_i(x)$ e $h_i(x)$ presentes nas equações 2.2 e 2.3, cuja a função objetivo é de minimização. Tais funções são contínuas pertencentes ao $\mathbb{R}^n$,e o conjunto S (o espaço de soluções) é um hiper-retângulo discretizado que define o limite superior e inferior das variáveis em que o vetor x é uma solução viável do problema que satisfaz todas as restrições. O objetivo é encontrar x^*, tal que $f(x) \geq f(x^*)$, para qualquer solução viável x (MLADENOVIć et al., 2008).

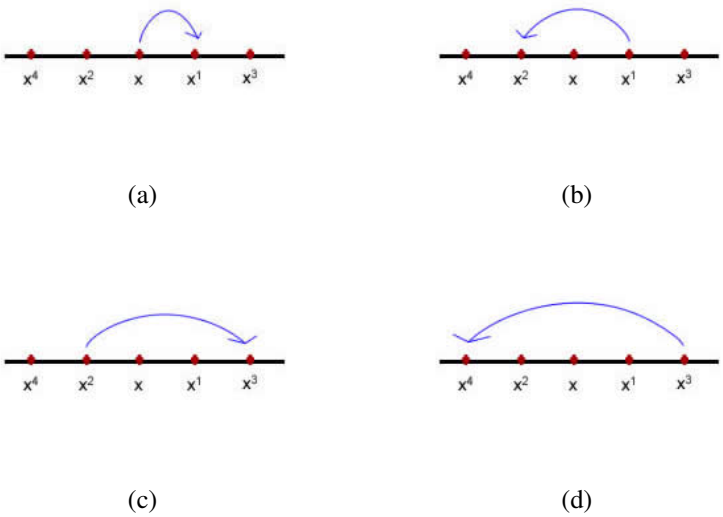
As funções $g_i(x)$ e $h_i(x)$ são denominadas, respectivamente, de restrições de desigualdade e igualdade e, quando presentes, definem o problema como Problema de Otimização Global com Restrições. Neste trabalho serão abordados apenas problemas irrestritos, ou seja, sem restrições.

Desta forma, a seguir serão descritos alguns procedimentos clássicos e algumas metaheurísticas para a resolução de problemas desta natureza.

2.2.1 Busca Linear

O procedimento de busca linear, em uma única dimensão, inicia dado um ponto t sobre o eixo $\mathbb{R}^n$, e conforme uma probabilidade de distribuição, a partir de um ponto qualquer x é realizado um caminho contínuo em $\mathbb{R}^n$ em busca de t (BECK, 1964). O procedimento de busca por t pode seguir diversas estratégias, como, por exemplo, partindo do ponto x, em uma direção positiva ou negativa, verifica-se se o ponto encontrado x^1 é t (Figura 2.2a); caso não seja volta-se, em sentido oposto, para explorar a outra metade em direção x^2 (Figura 2.2b). Caso não seja encontrado o ponto t, retorna-se novamente para explorar x^3 (Figura 2.2c) e assim sucessivamente.

Figura 2.2: Representação do procedimento de Busca Linear



A aplicação do procedimento de busca linear no contexto da otimização global contínua tem por objetivo encontrar uma solução ótima no domínio do problema, que consequentemente melhore a função objetivo utilizada, através de uma estratégia de pesquisa. Tal estratégia em sua maioria é de gênero iterativo, que ao escolher uma direção busca, melhorar a função objetivo. Dentro deste cenário, nesta subseção será apresentado o método iterativo de busca linear denominado proporção áurea.

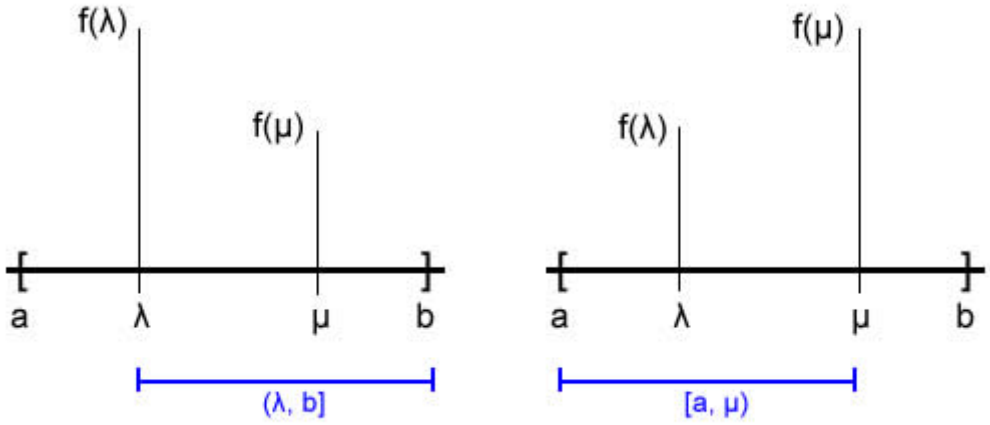
Método da Proporção Áurea

A proporção áurea, também conhecida como proporção de ouro, refere-se à um método iterativo que fundamenta-se no intervalo da incerteza. O Intervalo baseia-se na busca por encontrar o melhor valor para $f(x)$ no intervalo desconhecido $[a, b]$; a este intervalo chamamos de intervalo da incerteza. O procedimento de busca consiste em reduzir o intervalo, à medida que não for encontrado o valor mínimo, sendo assim, como a função f é quase convexa, então o intervalo da incerteza poderá ser reduzido a dois pontos, sejam eles λ e μ (BAZARAA; SHERALI; SHETTY, 2006).

Segundo Bazaraa, Sherali e Shetty (2006), uma função é quase convexa no intervalo $[a, b]$, se tendo λ e $\mu \in [a, b]$, tal que $\lambda < \mu$, logo se $f(\lambda) > f(\mu)$ então $f(z) \geq f(\mu)\ \forall$ $z \in [a, \lambda)$, conforme a ilustração na Figura 2.3a; ou se $f(\lambda) \leq f(\mu)$ então $f(z) \geq f(\lambda)\ \forall$

$z \in (\mu, b]$, como também pode ser observado na Figura 2.3b.

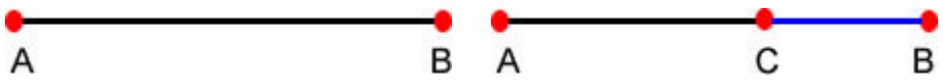
Figura 2.3: Representação do intervalo da incerteza. (a) Intervalo reduzido à $(\mu, b]$. (b) Intervalo reduzido à $[a, \lambda)$



(a) (b)

Com isto, o objetivo do método de proporção áurea é reduzir o intervalo da incerteza, subdividindo o espaço de busca baseado na razão áurea. A razão áurea é definida de modo que se tendo um segmento AB (Figura 2.4a) e existindo um ponto C que divide AB (Figura 2.4b), em média e extrema razão(Equação 2.4), será obtido o valor equivalente à 1,618 chamado de ϕ (DUNLAP, 1997).

Figura 2.4: Seção Áurea



(a) (b)

$$\frac{AB}{AC} = \frac{AC}{CB} = 1,618 = \phi \tag{2.4}$$

Dado tais recursos, o procedimento inicia escolhendo, através da razão áurea, os dois pontos λ e $\mu \in [a, b]$ (Equações 2.5 e 2.6).

$$\lambda = a + (1 - (\phi - 1))(b - a) \text{ e} \tag{2.5}$$

$$\mu = a + (\phi - 1)(b - a) \tag{2.6}$$

A cada iteração é verificado se os valores de λ e μ foram reduzidos até o comprimento final (t) previamente estabelecido. Posteriormente, a função objetivo é consultada em cada ponto. Caso $f(\lambda) > f(\mu)$, então é calculado um novo valor para μ e o intervalo de incerteza passará a ser $[\lambda, b]$ (CHAPRA; CANALE, 2008). Caso o valor de $f(\lambda) \leq f(\mu)$, o valor de λ é renovado e o intervalo de incerteza será reduzido para $[a, \mu]$, conforme pode ser observado na Figura 2.5.

Figura 2.5: Pseudo código Seção Áurea

procedimento SecaoAurea ($f(.)$, a, b, t)

1. $\phi \leftarrow 0,618$;
2. $\lambda \leftarrow a + (1 - \phi)(b - a)$;
3. $\mu \leftarrow a + \phi(b - a)$;
4. **enquanto**($|b - a| > t$) **faça**
5. **se** ($f(\lambda) > f(\mu)$) **então**
6. $a \leftarrow lambda$;
7. $\lambda \leftarrow \mu$;
8. $\mu = a + \phi(b - a)$;
9. **fim se**;
10. **senão**
11. $b \leftarrow \mu$;
12. $\mu \leftarrow \lambda$;
13. $\lambda = a + (1 - \phi)(b - a)$;
14. **fim senão**;
15. **fim enquanto**;
16. $\lambda \leftarrow (b + a)/2$

fim SecaoAurea.

2.3 Heurística

Heurísticas são procedimentos intuitivos eficientes, concebidos para solucionar problemas com custo computacional aceitável (AGUILERA; ROLI; SAMPELS, 2008). Propondo-se a reduzir a complexidade sem garantir, porém a otimalidade de seus resultados, buscando sempre valores próximos ao ótimo e utilizando, para tanto, operações simples de julgamento (TONETTO et al., 2006).

Existem vários procedimentos heurísticos aplicados à otimização combinatória, como *Simulated Annealing* (KIRKPATRICK, 1984), *Busca Tabu* (GLOVER; MARTI, 2006), *Va-*

riable Neighborhood Search (MLADENOVIć; HANSEN, 1997), entre outros.

Um dos procedimentos comumente utilizados é o método de busca local, onde o objetivo é procurar no espaço de soluções S de um dado problema, uma solução s^*, vizinha à solução corrente s, porém com propriedades de uma solução localmente ótima.

Apesar do grande avanço das heurísticas ao longo dos anos, Chaves (2009) destaca que os métodos heurísticos possuem certa fragilidade na execução de seus procedimentos, residindo na dificuldade de escapar de soluções ótimas locais, por utilizar o conceito de tentativa e erro. Outra dificuldade destacada é o caráter específico destes métodos que não possibilita sua total eficiência em uma classe ampla de problemas.

Dentro deste cenário, foram desenvolvidos algoritmos flexíveis que compartilham dos mesmos objetivos que os procedimentos heurísticos, porém sem a utilização excessiva de recursos computacionais, caracterizando assim as meta-heurísticas.

2.4 Meta-heurísticas

Introduzida por Glover (1986), o desenvolvimento das meta-heurísticas surgiu da combinação de métodos básicos de heurísticas, com o objetivo de resolver problemas de otimização combinatória, ou seja, cujo conjunto de soluções é composto por um conjunto finito de valores discretos, assim como os métodos heurísticos (ALBA, 2005).

As estratégias de busca utilizadas pelas meta-heurísticas, segundo Alba (2005), podem ser classificadas por quatro comportamentos distintos:

- Inspirados nos processos naturais, como a simulação do comportamento humano, acreditando na possibilidade de evoluir uma população de soluções, que possuem características complementares, com o objetivo de permitir maior diversificação no espaço de busca.

- Uso de memória auxiliar de forma dinâmica, extraindo informações no decorrer da busca.

- Utilização de uma estratégia determinística, em que a solução inicial, através de buscas locais, encaminham o algoritmo para a solução final.

- Execução iterativa e gulosa, em que a solução é construída de forma gradativa, analisando a variável de decisão do problema.

Uma vez que meta-heurísticas faz um uso parcimonioso de recursos computacionais em suas operações, torna-se possível alcançar classes maiores de problemas. Isto por ter permitido um papel destacado em diversas aplicações nos últimos vinte anos (TALBI, 2009). Elas podem ser construídas da junção de duas ou mais heurísticas, que permite a elevação do número de procedimentos existentes.

A seguir, serão apresentadas três meta-heurísticas: *Greedy Randomized Adaptative Search Procedure* (GRASP), *Iterated Local Search*(ILS) e *Variable Neighborhood Search* (VNS).

2.4.1 *Greedy Randomized Adaptative Search Procedure* (GRASP)

O procedimento *Greedy Randomized Adaptive Search Procedure* (GRASP), proposto por Feo e Resende (1995), é um método *multistart* para problemas combinatórios, cuja iteração consiste de dois processos: fase de construção e fase de busca local.

A fase de construção é realizada de forma pseudo-randômica (pseudo-aleatória), sendo um processo iterativo que analisa cada elemento gerado, através de uma função adaptativa gulosa, responsável por selecionar cada elemento que fará parte da solução corrente. Este método possibilita a geração de várias soluções GRASP diferentes a cada iteração (RESENDE; RIBEIRO, 2003). A função adaptativa gulosa é definida em $g : C \rightarrow \mathbb{R}$, onde C, corresponde a uma lista de elementos que possuem características que beneficiam a função objetivo do problema tratado. A definição da função como gulosa é dada por esta escolher apenas elementos favoráveis à solução, utilizando para isso uma variável de controle α, chamada de fator de gulosidade, que varia no intervalo [0,1]. Quando $\alpha = 0$ a função gera soluções puramente gulosas, já quando $\alpha = 1$, gera soluções puramente aleatórias.

Após a fase de construção, as soluções obtidas ainda não são consideradas ótimas locais, portanto devem ser melhoradas e passam passam pela fase de busca local. Neste método, trabalha-se de maneira iterativa em busca de uma melhor solução em sua vizinhança, sendo sua eficiência dependente das construções de cada solução. O método é finalizado após ser percorrido todo o espaço de vizinhança da solução atual, conforme pode ser observado na

Figura 2.6.

Figura 2.6: Pseudo código GRASP

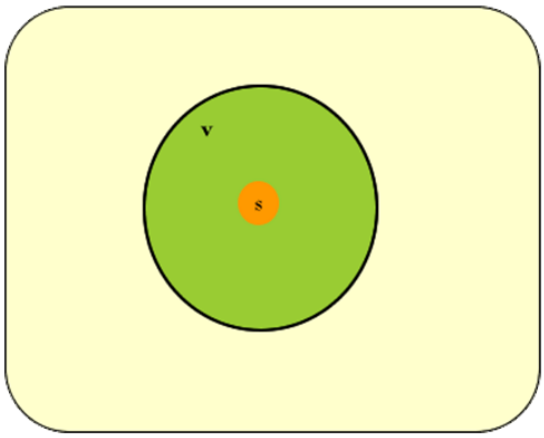
procedimento GRASP ($GRASPMax$, s)

1. **para** $i \leftarrow 1...GRASPMax$ **faça**
2. ConstrucaoSolucao(s);
3. BuscaLocal(s);
4. AtualizandoSolucao(s, s^*);
5. **fim-para**
6. returna(s^*);

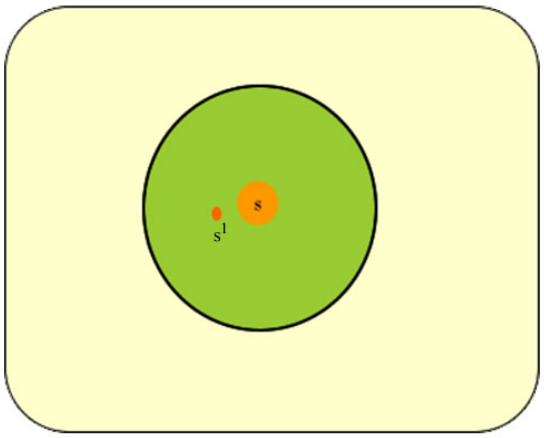
fim `GRASP`.

A representação do processo é ilustrado na Figura 2.7, em que partindo de uma solução inicial s (Figura 2.7a), pertencente ao espaço de soluções S, é realizado um procedimento de busca local no espaço de vizinhança v, que busca encontrar soluções favoráveis à resolução do problema, preenchendo assim a Lista de Candidatos Restrita (LCR), que, ao final da iteração, é conhecida a solução s^1 (Figura 2.7b).

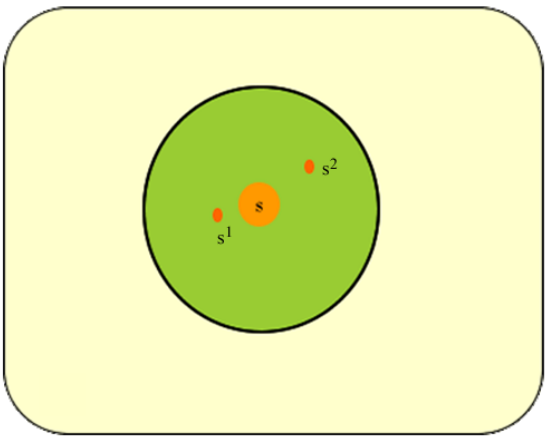
Figura 2.7: Representação do Procedimento GRASP, em quatro etapas, nas exploração do espaço de vizinhança v



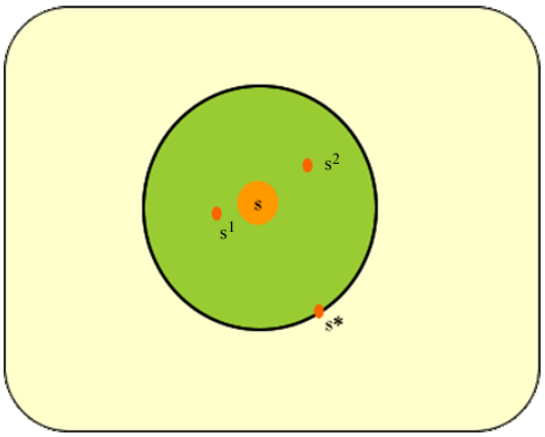
(a) Geração da solução inicial s



(b) Solução s^1, vizinha de s, encontrada após a primeira iteração



(c) Solução s^2, vizinha de s^1, encontrada após o reinicio do procedimento



(d) Solução s^*, vizinha de s^2, sendo ótima para todas as iterações GRASP

Com isto, o procedimento é reiniciado na próxima iteração, obtendo a solução s^2 (Figura 2.7c). O processo é realizado $GRASPMax$ vezes até que seja encontrada a solução s^* (Figura 2.7d), considerada a melhor solução obtida em todas as iterações. Pode-se observar que o procedimento realiza a exploração apenas de um espaço de vizinhança v inicial, em todas as iterações em busca de uma solução que favoreça à resolução do problema tratado.

O método GRASP é denominado adaptativo, por contribuições advindas da escolha dos

elementos a cada iteração, portanto segundo Souza, o GRASP "conjuga bons aspectos dos algoritmos puramente gulosos, como aqueles dos procedimentos aleatórios de construção de soluções".

2.4.2 *Iterated Local Seach* (ILS)

Este método foi proposto por Stützle (1998), possuindo a função de gerar novas soluções de partida para o procedimento de busca local, através de perturbações na solução ótima local.

Conforme Lourenço, Martin e Stützle (2003), sua estrutura é composta por quatro fases: a geração de solução inicial, busca local, perturbação e o critério de aceitação.

A geração de solução inicial pode ser realizada aleatoriamente ou com a utilização de uma heurística eficiente; a fase de busca local é um algoritmo de avaliação, que busca encontrar uma solução vizinha à solução corrente, a qual melhore a função objetivo; já a fase de perturbação é constituída por pequenos movimentos na solução corrente, porém tais movimentos deverão ser fortes o suficiente para explorar o espaço de soluções e ao mesmo tempo suaves para não alterar as características da atual solução. Por fim, o critério de aceitação é responsável por verificar se houve melhora no procedimento, analisando o valor da função objetivo.

A seguir é descrito o pseudo código ILS.

Figura 2.8: Pseudo código ILS

procedimento ILS ($MaxIterILS$, s)

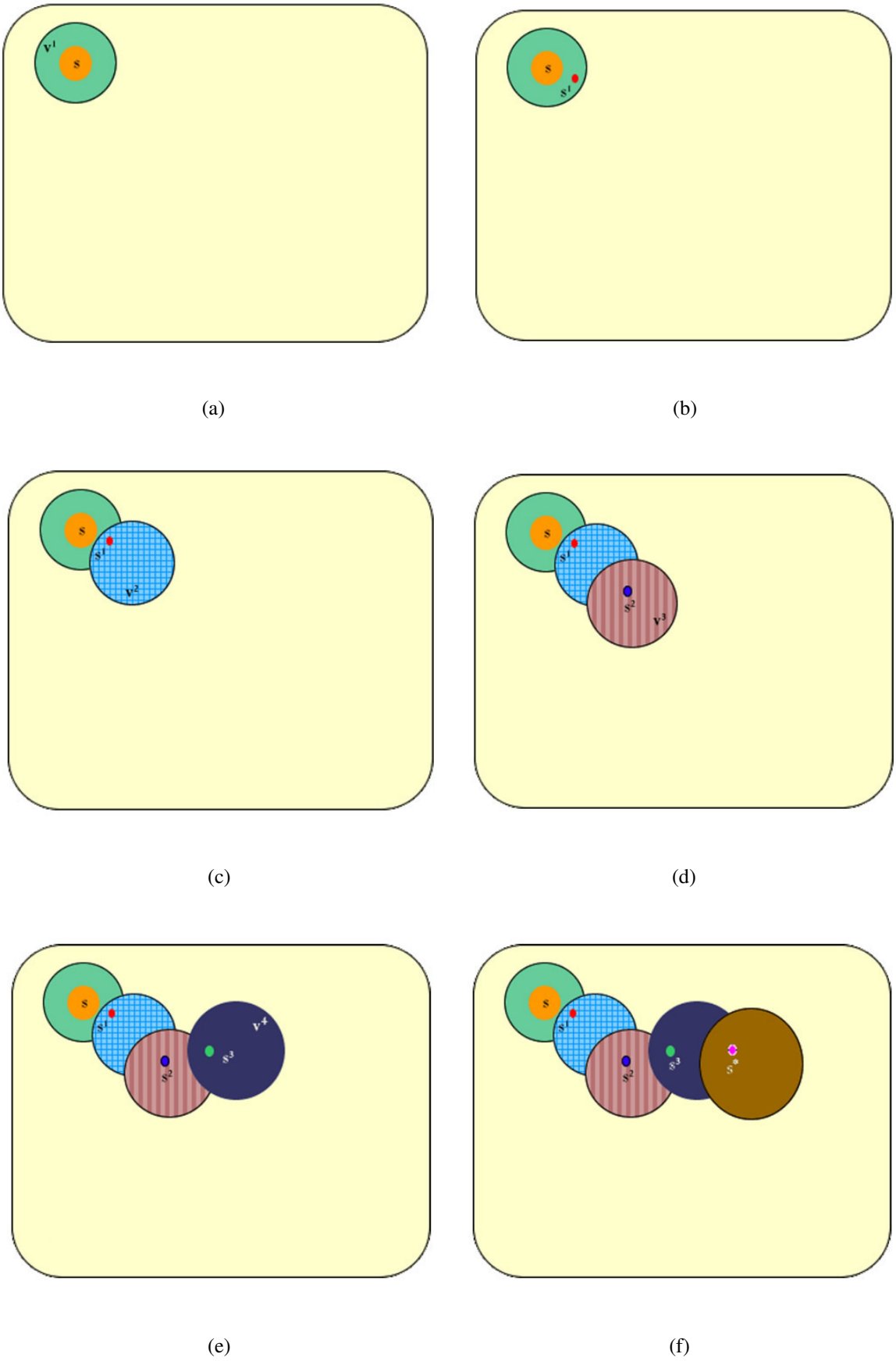
1. $s \leftarrow$ConstruçãoDaSoluçãoInicial();
2. $s^* \leftarrow$BuscaLocal(s);
3. **enquanto** $i < MaxIterILS$ **faça**
4. $s^1 \leftarrow$Pertubacao(s^*);
5. $s^2 \leftarrow$BuscaLocal(s^1);
6. **se** ($f(s^2) < f(s^*)$) **faça**
7. $s^* \leftarrow s^2$;
8. $f(s^*) \leftarrow f(s^2)$;
9. **fim-se**
10. **fim-enquanto**
11. **retorna**($s*$);

fim ILS.

O critério de aceitação é uma componente utilizada não apenas para a intensificação da pertubação da solução, possuindo também a responsabilidade de qualificar a diversificação utilizada no procedimento.

A Figura 2.9, apresenta o comportamento do procedimento ILS dado um conjunto de soluções S. A partir da solução inicial s (Figura 2.9a), é realizado um procedimento de busca local em seu espaço de vizinhança, obtendo uma solução vizinha que sofrerá perturbações para ser submetida ao procedimento de busca local. Caso esta satisfaça o critério de aceitação, é obtida a solução s^1 (Figura 2.9b), que reinicia o processo até ser encontrada a solução s^* (Figura 2.9f), que em seu espaço de vizinhança não possui solução.

Figura 2.9: Representação do procedimento ILS na exploração do espaço de soluções S



2.4.3 *Variable Neighborhood Search* (VNS)

Introduzida por Mladenović e Hansen (1997), diferencia-se das demais meta-heurísticas por não utilizar uma busca em um único espaço de vizinhança. Sua estratégia consiste em realizar variações sistemáticas da vizinhança, explorando assim regiões distintas.

Tendo um conjunto finito de estruturas de vizinhanças N_k com $k = 1...n$, sendo $s^k(s)$ a solução na k-ésima vizinhança de s. O procedimento inicia com $k = 1$ e até que k seja diferente de n, procurando uma solução aleatória s^1, pertencente a k-ésima vizinhança de x ($s^1 \in N_k(s)$), que após o procedimento de busca local traga uma melhoria na função objetivo; porém não sendo obtida, k será atualizado ($k = k + 1$), caso contrário o procedimento continuará com $k = 1$. Conforme pode ser verificado no pseudo código (Figura 2.4.3) (MLADENOVIć; HANSEN, 1997).

Figura 2.10: Pseudo código VNS

procedimento VNS ($MaxIterILS$, s)

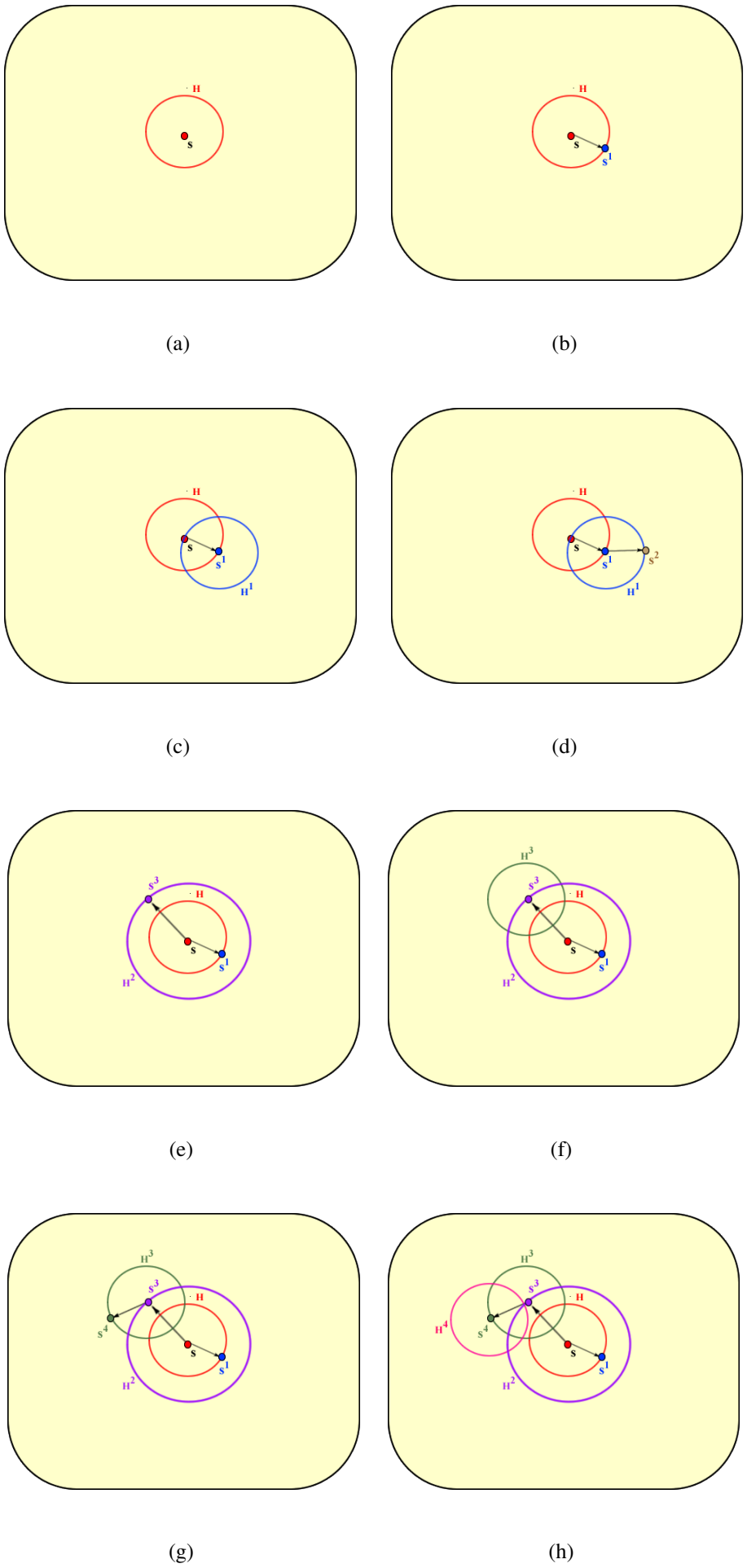
1. Selecione vizinhanças N_k, $k = 1..n$;
2. $s \leftarrow$ solucaoInicial();
3. **enquanto** (criterio de parada não satisfeito) **faça**
4. $k \leftarrow 1$;
5. **enquanto** $k \leq n$ **faça**
6. $s^1 \leftarrow$Pertubacao($N_k(s)$);
7. $s^2 \leftarrow$BuscaLocal($N_k(s^1)$);
8. **se** ($f(s^2) < f(s)$) **faça**
9. $s \leftarrow s^2$;
10. $k \leftarrow k + 1$;
11. **fim-se**
12. **fim-enquanto**
13. **fim-enquanto**

fim VNS.

O procedimento é melhor detalhado na Figura 2.11, como pode ser observado a seguir. Partindo de uma solução $s \in S$ (Figura 2.11a) é iniciado um processo de busca na região

de vizinhança H por uma nova solução s^1 (Figura 2.11b). Esta solução é gerada a partir de um processo de perturbação, onde o valor da função é verificado para as duas, sendo $f(s^1) < f(s)$ (Figura 2.11c). Tendo s^1 melhor valor, o procedimento é renovado tendo como solução atual s^1. A partir disso, o processo de busca local é novamente realizado, agora na vizinhança H^1, em que a solução s^2 é encontrada da mesma maneira que s1 e tem seu valor da função objetivo validado $f(s^2) < f(s^1)$. Caso esta tenha um insucesso (Figura 2.11d), o VNS reinicia sua operação aumentando o espaço de vizinhança $k = 1$(Figura 2.11e), e continua o procedimento de busca, visto anteriormente, em torno de uma nova solução. Este processo continua somente se um movimento de melhora é realizado; caso contrário, serão realizadas trocas sistemáticas de estruturas de vizinhança (Figura 2.11e), até que o critério de parada seja satisfeito, ou o número total de iterações seja atingido ($k = n$).

Figura 2.11: Representação do procedimento VNS na exploração do espaço de soluções S



(a) (b)

(c) (d)

(e) (f)

(g) (h)

O procedimento de perturbação, também conhecido como *shaking*, corresponde à etapa de diversificação da busca, responsável por realizar os saltos para exploração de regiões distantes, permitindo à busca local realizar um processo de intensificação (CARRIZOSA et al., 2012). Deste modo, o procedimento pode realizar piora a qualidade da solução, porém permite um bom e simples mecanismo de fuga de soluções ótimas locais, além de uma exploração mais abrangente do espaço de soluções.

Tendo sido apresentados estes conceitos básicos, no capítulo seguinte serão abordadas meta-heurísticas no domínio contínuo e sequencial.

Capítulo 3

Trabalhos Relacionados

3.1 *Continuous Greedy Randomized Adaptive Search Procedure* (C-GRASP)

Formulado por Hirsch et al. (2007), apresenta um comportamento semelhante ao procedimento GRASP proposto por Feo e Resende (1995), mantendo a característica *multistart*, que realiza o procedimento adaptativo e guloso para a geração da solução inicial e mantendo o procedimento de busca local. A diferença existente entre os dois métodos encontra-se na não utilização de um único procedimento de construção seguido por um procedimento de busca local, mas sim na utilização de uma série de ciclos construção-busca local, conforme pode ser verificado na Figura 3.1.

Figura 3.1: Pseudo código C-GRASP

procedimento C-GRASP (n, l, u, $f(.)$, h_s, h_e, ρ_{lo})

1. $f^* \leftarrow \infty$;
2. **enquanto** ψ **faça**
3. $x \leftarrow k, k \in [l, u]$;
4. $h \leftarrow h_s$;
5. **enquanto** $h \geq h_s$ **faça**
6. $Impr_c \leftarrow$ falso;
7. $Impr_L \leftarrow$ falso;
8. [x,$Impr_c$] $\leftarrow$ Construção-CGRASP(x, $f(.)$, n, h, l, u, $Impr_c$);
9. [x,$Impr_L$] $\leftarrow$ BuscaLocal-CGRASP(x, $f(.)$, n, h, l, u, ρ_{lo}, $Impr_L$);
10. **se** $f(x) < f^*$ **então**
11. $x^* \leftarrow x$;
12. $f^* \leftarrow f(x)$;
13. **fim se**
14. **se** $Impr_c$ = falso **e** $Impr_L$ = falso **então**
15. $h \leftarrow h/2$; /*faz a grade ficar mais densa*/
16. **fim se**
17. **fim enquanto**
18. **fim enquanto**
19. **retorne** x^*;

fim `C-GRASP`.

O procedimento utiliza como argumentos de início: a dimensão do problema n, os vetores l e u que correspondem respectivamente ao limite inferior e superior do espaço de soluções discretizado (grade), l, u$\in \mathbb{R}^n$, a função objetivo, a densidade inicial (h_s) e final(h_e) da grade de discretização e, por fim, o parâmetro ρ_{lo}, que define a porção de vizinhança que será visitada na fase de busca local, a partir da solução corrente (HIRSCH; PARDALOS; RESENDE, 2010).

Observando o algoritmo apresentado na Figura 3.1, em cada iteração (linha 3–17), até que o critério de aceitação (ψ) seja atingido, a solução x é definida de forma aleatória em

uma distribuição uniforme sobre a grade, cujo os limites inferiores e superiores são definidos por l e u. O critério de aceitação, que pode ser mais de um, pode ser definido, por exemplo, como o número total de avaliações da função objetivo ou o tempo decorrido desde o início do procedimento.

Voltando a avaliar o comportamento do algoritmo, a variável que controla a densidade inicial da grade é reiniciada com o valor passado por parâmetro (h_s), e, enquanto a densidade da grade for maior ou possuir igual valor, serão executados, de forma sequencial, a fase de construção e busca local, retornando uma nova solução que terá o seu valor de função objetivo comparada. Havendo melhoria ($f(x) < f^*$), a solução x será atualizada para a solução corrente, porém não havendo melhoria na qualidade da solução de forma simultânea na fase de construção ($Impr_c$ = falso) e na fase de busca local ($Impr_L$ = falso), a densidade da grande será atualizada, tornando-se menor (mais densa), a fim de melhorar a qualidade de tais processos.

A seguir, serão detalhados os procedimentos de construção e de busca local (HIRSCH et al., 2007), que não utilizam cálculo do gradiente.

3.1.1 Fase de Contrução

O método C-GRASP é constituído por duas fases: construção e busca local. A fase de construção, como mencionado anteriormente, utiliza uma solução x inicial, gerada a partir de um procedimento randômico que permite, por sua vez, a coleta de um conjunto diversificado de soluções. Ao iniciar a fase de construção, será permitido a x que todas as suas coordenadas mudem seus valores, podendo essas serem chamadas de coordenadas não-fixas (HIRSCH; PARDALOS; RESENDE, 2010).

Figura 3.2: Pseudo código referente à fase de construção do método C-GRASP

procedimento Construção-CGRASP(x, $f(.)$, n, h, l, u, $Impr_c$)

1. $nFixa \leftarrow [1, n]$
2. $\alpha \leftarrow k, k \in [0, 1]$;
3. $ReUse \leftarrow$ falso;
4. **while** $nFixa \neq \emptyset$ **faça**
5. $g \leftarrow +\infty$;
5. $\bar{g} \leftarrow -\infty$;
6. **para** i = 1 ... n **faça**
7. **se** $i \in nFixa$ **então**
8. **se** $ReUse =$ falso **então**
9. $z_i \leftarrow$ buscaLinear(x, h, i, n, $f(.)$, l, u);
10. $g_i \leftarrow f(z_i)$;
11. **fim se**;
12. **se** $g > g_i$ **então** $g \leftarrow g_i$;
13. **se** $\bar{g} < g_i$ **então** $\bar{g} \leftarrow g_i$;
14. **fim se**
15. **fim para**
16. LCR $\leftarrow \emptyset$;
17. limiar $\leftarrow g + \alpha(\bar{g} - g)$;
18. **para** i = 1 ... n **faça**
19. **se** $i \in nFixa$ e $g_i \leq limiar$ **então**
20. LCR $\leftarrow$ LCR $\cup i$;
21. **fim se**
22. **fim para**
23. $j \leftarrow$ ElementoAleatorio(LCR);
24. **se** $x_j = z_j$ **então** ReUse $\leftarrow$ verdadeiro;
25. **senão**
26. $x_j \leftarrow z_j$;
27. ReUse $\leftarrow$ falso;
28. $Impr_c \leftarrow$ verdadeiro;
29. **fim senão**
30. $nFixa \leftarrow nFixa \setminus \{j\}$;
31. **fim enquanto**
32. **retorne**(x, $Impr_c$);

fim `Construção-CGRASP`.

Tendo o conjunto de coordenadas não fixas ($nFixa$) inicializado, conforme podemos verificar na Figura 3.2, é realizado um procedimento de busca linear para cada coordenada não fixa i, mantendo neste as outras $n-1$ em seus valores atuais. Após o procedimento de busca linear, o valor encontrado será armazenado na i-ésima coordenada, z_i e o valor correspondente a sua função objetivo na variável g_i. E através dos valores de g_i será formada a LCR, composta apenas por coordenadas não fixas que possuem melhores valores g_i, ou seja, cujo os valores são menores ou iguais ao limiar dado por $g + \alpha(\bar{g} - g)$, em que g e $\bar{g}$, representam respectivamente os valores máximo e mínimo de g_i, entre todas as coordenadas não fixas de x. A seguir, é escolhida, de forma aleatória, uma coordenada para ser retirada da LCR preenchida. O procedimento será repetido, até que LCR esteja vazia. Este modo de escolha aleatória da coordenada oferece ao processo um fator de gulosidade e de aleatoriedade, fato importante para a geração de uma solução de boa qualidade (HIRSCH et al., 2007).

3.1.2 Fase de Busca Local

Após a finalização do procedimento de construção é dado início ao processo de busca local, que segundo Hirsch, Pardalos e Resende (2010) pode ser considerado como uma função de aproximação entre a função do gradiente e a função objetivo. Dado uma solução $x \in \mathbb{R}^n$, o procedimento de busca local gera um conjunto de vizinhanças que serão verificadas para tentar encontrar uma nova solução que seja melhor que a solução atual. Este novo ponto então é adotado como ponto inicial para que o procedimento de busca continue.

Analisando o algoritmo apresentado na (Figura 3.3), no primeiro momento, a solução x é considerada como a melhor solução (x^*). Ela é recebida do processo de construção inicial (linha 1), que é valor necessário para determinar o número de pontos da grade direcionados através do parâmetro h, onde o máximo de pontos examinados será dado por $B_h(x^*)$ e ρ_{lo} corresponderá à proporção da vizinhança exposta para análise.

Figura 3.3: Pseudo código referente à fase de busca local do método C-GRASP

procedimento BuscaLocal-CGRASP(x, $f(.)$, n, h, l, u, ρ_{lo}, $Impr_L$)

1. $x^* \leftarrow x$
2. $f^* \leftarrow f(x)$;
3. $PontosGrade \leftarrow \prod_{i=1}^{n} \lceil (u_i - l_i/h \rceil$;
4. $MaximoPontosParaExaminar \leftarrow \lceil \rho lo \cdot PontosGrade \rceil$;
5. $PontosExaminados \leftarrow 0$;
6. **enquanto** $PontosExaminados \leq MaximoPontosParaExaminar$ **então**
7. $\quad PontosExaminados \leftarrow PontosExaminados + 1$;
8. $\quad x \leftarrow ElementoAleatorio(B_h(x^*))$;
9. $\quad$ **se** $l \leq x \leq u$ **e** $f(x) < f^*$ **então**
10. $\quad\quad x^* \leftarrow x$;
11. $\quad\quad f^* \leftarrow f(x)$;
12. $\quad\quad Impr_L \leftarrow verdadeiro$;
13. $\quad\quad PontosExaminados \leftarrow 0$;
14. $\quad$ **fim se**
15. **fim enquanto**
16. $\quad$ **retorne**(x, $Impr_L$);

fim `BuscaLocal-CGRASP`.

Dentro deste cenário, podemos definir a vizinhança do algoritmo C-GRASP através de uma solução atual $\bar{x} \in \mathbb{R}^n$ (representada em vermelho na Figura 3.4a), discretizada através de h, onde:

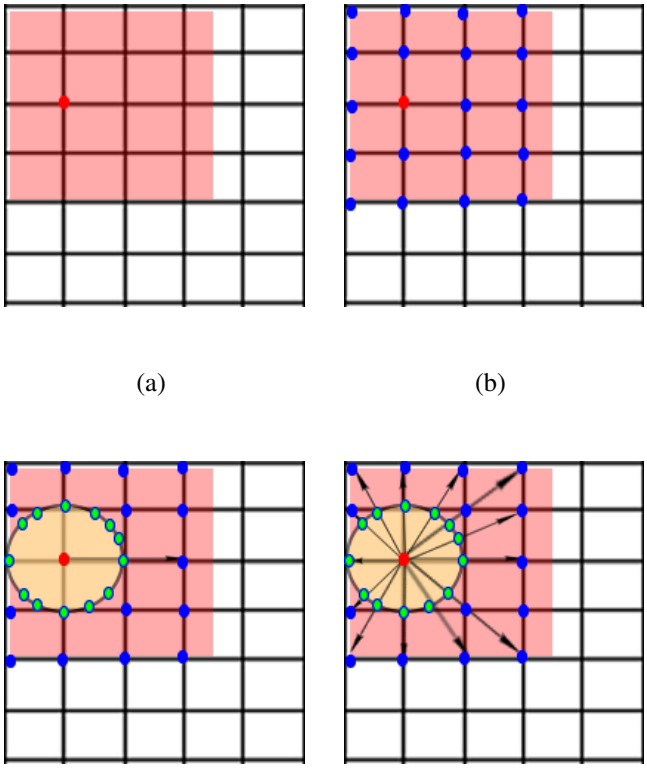
$$S_h = \{x \in S | l \leq x \leq u, x = \bar{x} + \tau \cdot h, \ \tau \in \mathbb{Z}^n\}, \tag{3.1}$$

é o conjunto de pontos em S, que são medidas de tamanho h a partir de $\bar{x}$ (representados em azul na Figura 3.4b). Desta maneira:

$$B_h(\bar{x}) = \{x \in S | x = \bar{x} + h \cdot (x' - \bar{x}) / \|x' - \bar{x}\|, x' \in S_h(\bar{x}) \setminus \{\bar{x}\}\}, \tag{3.2}$$

corresponde à projeção dos pontos $S_h(\bar{x}) \setminus \{\bar{x}\}$, dentro da hiper esfera centrada em $\bar{x}$ de raio h (Figura 3.4c). Ou seja, o h-ésimo vizinho de $\bar{x}$ é definido através dos pontos contidos em $B_h(\bar{x})$ (Figura 3.4d).

Figura 3.4: Representação do espaço de vizinhança C-GRASP



(a) (b)

(c) (d)

Na linha 6 à linha 15, o algoritmo seleciona $MaximoPontosParaExaminar$ vezes, aleatoriamente, um ponto pertencente a $B_h(x^*)$. Caso a solução atual x selecionada de $B_h(x^*)$ seja viável, então x^* será definida por x ($x^* \leftarrow x$), e $Impr_L$ como verdadeiro. $Impr_L$, assim como $Impr_c$ no procedimento de construção, possui a funcionalidade de sinalizar se o procedimento produziu ou não melhoria na qualidade da solução gerada.

3.2 *Enhanced Continuous Greedy Randomized Adaptive Search Procedure* (EC-GRASP)

O C-GRASP proporciona bons recursos para a fuga de mínimos locais sem um elevado esforço computacional, porém por utilizar um procedimento de construção pseudo-aleatório apresenta, assim como o GRASP, certa dificuldade na convergência da solução. Baseado nesta dificuldade, e com o objetivo de acelerar o encontro de uma direção de busca por vizinhanças promissoras surgiu o EC-GRASP, proposto por Araújo, Cabral e Nascimento (2008), que consiste em uma hibridização do método C-GRASP com o método de busca local direcionada denominada *Adaptive Pattern Search* (APS), elaborada por Hedar e Fukushima (2006).

Sendo uma adaptação do procedimento C-GRASP o método EC-GRASP, mantém sua característica *multistart*, constituída de uma fase de construção gulosa e adaptativa, seguida por uma fase de busca local. Mantendo o mesmo procedimento de construção, a diferença apresenta-se na fase de busca local, ao utilizar um método de busca por padrões, conforme podemos verificar na Figura 3.5, linha 9.

Figura 3.5: Pseudo código EC-GRASP

procedimento EC-GRASP (n, l, u, $f(.)$, h_s, h_e, $MaxIter$)

1. $f^* \leftarrow \infty$;
2. **enquanto** ψ **faça**
3. $x \leftarrow k, k \in [l, u]$;
4. $h \leftarrow h_s$;
5. **enquanto** $h \geq h_s$ **faça**
6. $Impr_c \leftarrow$ falso;
7. $Impr_L \leftarrow$ falso;
8. [x,$Impr_c$] $\leftarrow$ Construção-CGRASP(x, $f(.)$, n, h, l, u, $Impr_c$);
9. [x,$Impr_L$] $\leftarrow$ BuscaLocal-APS(x, $f(.)$, n, h, l, u,h_s, $Impri_L$, $MaxIter$);
10. **se** $f(x) < f^*$ **então**
11. $x^* \leftarrow x$;
12. $f^* \leftarrow f(x)$;
13. **fim se**
14. **se** $Impr_c$ = falso **e** $Impr_L$ = falso **então**
15. $h \leftarrow h/2$; /*faz a grade ficar mais densa*/
16. **fim se**
17. **fim enquanto**
18. **fim enquanto**
19. **retorne** x^*;

fim `EC-GRASP`.

O procedimento também faz uso dos mesmos argumentos de entrada do algoritmo C-GRASP: a dimensão do problema n, os vetores l e u que correspondem aos vetores no $\mathbb{R}^n$ que definem os limites inferiores e superiores, a função objetivo $f(.)$, a densidade inicial (h_s) e final(h_e) da grade de discretização. Adiciona-se apenas mais um argumento $MaxIter$, que corresponde ao número máximo permitido de iterações sem melhoras ao procedimento BuscaLocal-APS.

3.2.1 *Adaptive Pattern Search* (APS)

O procedimento APS baseia-se no método *Approximate Descent Direction* - (ADD), é livre de derivada assim como o C-GRASP e o EC-GRASP, porém possui elevado desempenho na geração de uma direção de descida eficiente (HEDAR; FUKUSHIMA, 2004). Através de um ponto inicial $x \in \mathbb{R}^n$, a obtenção da direção de aproximação descendente $v \in \mathbb{R}^n$ de f em x é dada através da geração de n pontos ao redor de p ($V = \{y_i\}_{i=1}^n$, ou seja, $y_i$vizinhos de p), como segue:

$$v = \sum_{i=1}^{n} w_i e_i, \tag{3.3}$$

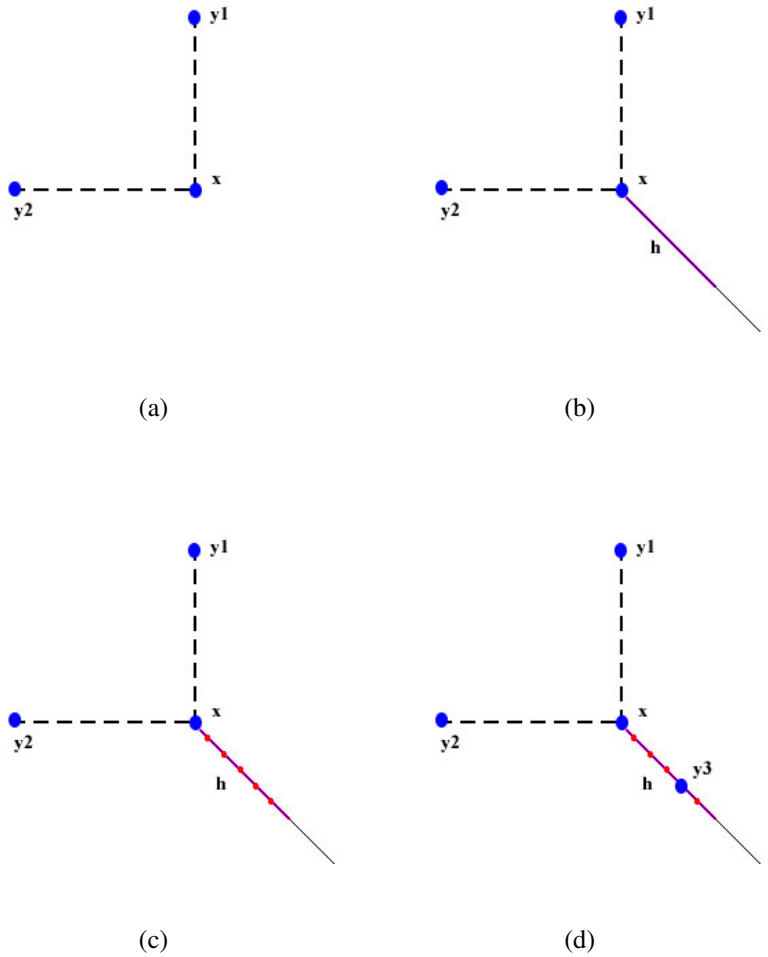
onde

$$\begin{aligned} w_i &= \frac{\Delta f_i}{\sum_{j=1}^{m} |\Delta f_j|}, \\ e_i &= -\frac{(y_i - x)}{\|y_i - x\|}, \\ \Delta f_i &= f(y_i) - f(x),\ i = 1, 2, ..., n. \end{aligned} \tag{3.4}$$

Na estratégia APS, são gerados n padrões de direção ao atual eixo de coordenadas que emanam de cada iteração i, em um ponto x, gerando assim vizinhos a cada passo. Após o cálculo da direção v, dois pontos y_{n+1} e y_{n+2} são gerados com tamanho de passos diferentes e utilizados na análise da direção v, em que o melhor dos $n + 2$ pontos será considerado como uma solução final do processo.

A Figura 3.6 ilustra o procedimento APS em duas dimensões, em que, dado os pontos y_1 e y_2 responsáveis por gerar uma direção de descida, assume-se que x é a melhor solução entre y_1 e y_2 (Figura 3.6a). O procedimento realiza uma busca com passos discretizados em h (Figura 3.6c), gerando o ponto y_3 (Figura 3.6d), onde o melhor ponto entre (y_1, y_2, y_3) será retornado para o método EC-GRASP.

Figura 3.6: Representação do procedimento APS em duas dimensões



Por sua vez, o método EC-GRASP utilizando o método APS, aplica a seção áurea no processo de análise da direção v calculada, através de um número inicial de passos h e um número final h_e para a discretização da direção v, conforme a Figura 3.7.

Figura 3.7: Pseudo código APS

procedimento APS (n, l, u, $f(.)$, h, h_e, $MaxIter$)

/*Gera $V = \{y_i\}_{i=1}^{n}$, conjunto de pontos y_i a partir de x com tamanho h*/

1. **para** i=1..n **faça**
2. **se** $f(y_i) < f(x)$ **então**
3. **retorne** y_i;
4. **fim se**;
5. **fim para**;
6. Calcule a direção v utilizando a equação 3.3
7. $y_{n+1} \leftarrow$ SecaoAurea(x, $f(.)$, h_s, h_e, v);
8. $V \leftarrow V \cup \{y_{n+1}\}$;
9. **retorne** $\{y_i*\}$; /*melhor elemento pertencente a V*/

fim APS.

3.2.2 Busca Local APS

É um procedimento de busca iterativo, que não utiliza cálculos de derivada e não aceita soluções com características inferiores à solução atual. Utiliza um conjunto de vizinhanças baseado na busca direcional do procedimento APS e na organização do conjunto $B_h(x)$, conforme apresentado na Figura 3.8.

Figura 3.8: Pseudo código BuscaLocal-APS

procedimento BuscaLocal-APS (x, $f(.)$, n, h, l, u,h_s, $Impri_L$, $MaxIter$)

1. $x^* \leftarrow x$;
2. $f(x^*) \leftarrow f(x)$;
3. $NIter \leftarrow 0$;
4. **enquanto** $NIter \leq MaxIter$ então
4. $NIter \leftarrow NIter + 1$;
5. $x \leftarrow$ APS (n, l, u, $f(.)$, h, h_e, $MaxIter$);
6. **se** $l \leq x \leq u$ e $f(x^*) > f(x)$ **então**
7. $x^* \leftarrow x$;
8. $f(x^*) \leftarrow f(x)$;
9. $Impri_L \leftarrow$ verdadeiro;
10. **fim se**
11. **senão** $x \leftarrow ElementoAleatorio(B_h(x^*))$;
12. **fim enquanto**
13. retorne (x, $Impri_L$);

fim `BuscaLocal-APS`.

O procedimento se inicia com a transferência da solução obtida no método de construção para a variável x^* e seu valor da função objetivo para $f(*)$. Ao iniciar o procedimento iterativo (linha 4), é executado o método APS, que transfere o resultado da busca linear à variável x. Caso esta apresente melhora na função objeto, então os dados serão atualizados e o processo é reiniciado ($NIter$ = 0); caso contrário, será escolhida de forma aleatória uma nova solução pertencente ao conjunto $B_h(x^*)$, que corresponde à vizinhança da solução atual.

Como apresentado anteriormente, o procedimento de busca local APS propõe uma nova exploração do espaço de vizinhança, com bons resultados como comprovado por Araújo, Cabral e Nascimento (2008). Porém tal exploração só ocorrerá caso não seja encontrada uma solução x que apresente melhor desempenho $f(x)$ que o valor da melhor solução conhecida $f(x^*)$, onde será modificada a vizinhança utilizando o conjunto $B_h(x^*)$.

3.3 BFGS *Continous*-GRASP (BC-GRASP)

O método desenvolvido por Araújo, Cabral e Nascimento (2008) é constituído pela combinação do método EC-GRASP e o procedimento exato BFGS com memória limitada (Limited Memory BFGS - LBFGS). Ele aplica uma estratégia que proporciona convergência mais eficiente, em comparação ao EC-GRASP, por utilizar métodos de convergências locais diversificadas, constituídas pelo método APS, apresentado na seção 3.2.2. Apresenta também o procedimento quase-Newton, com bons resultados.

A formação do método ocorre na utilização do procedimento EC-GRASP, na fase inicial do processo e no refinamento da solução realizado pelo procedimento BFGS, conforme ilustrado na Figura 3.9.

Figura 3.9: Pseudo código BC-GRASP

procedimento BC-GRASP (n, l, u, $f(.)$, h_s, h_e, $MaxIter$, m)

1. $f^* \leftarrow \infty$;
2. **enquanto** ψ **faça**
3. $x \leftarrow k, k \in [l, u]$;
4. $h \leftarrow h_s$;
5. **enquanto** $h \geq h_s$ **faça**
6. $Impr_c \leftarrow$ falso;
7. $Impr_L \leftarrow$ falso;
8. [x,$Impr_c$] $\leftarrow$ Construção-CGRASP(x, $f(.)$, n, h, l, u, $Impr_c$);
9. [x,$Impr_L$] $\leftarrow$ BuscaLocal-APS(x, $f(.)$, n, h, l, u,h_s, $Impri_L$, $MaxIter$);
10. **se** $Impr_c$ = falso e $Impr_L$ = false **então**
11. $x \leftarrow$ LBFGS(x, $f(.)$, n, m);
12. $h \leftarrow h/2$;
13. **fim se**
14. **se** $f(x) < f^*$ **então**
15. $x^* \leftarrow x$;
16. $f^* \leftarrow f(x)$;
17. **fim se**
18. **se** $Impr_c$ = falso **e** $Impr_L$ = falso **então**
19. $h \leftarrow h/2$; /*faz a grade ficar mais densa*/
20. **fim se**
21. **fim enquanto**
22. **fim enquanto**
23. **retorne** x^*;

fim `BC-GRASP`.

Conforme pode ser observado, o único argumento de entrada que o diferencia do procedimento EC-GRASP é a variável m, que corresponde ao número máximo de atualizações realizadas na matriz quase-Newton, equivalente assim à limitação de memória do procedimento LBFGS. O procedimento mantém a utilização de um conjunto de ciclos de construção-

busca local utilizados no EC-GRASP. Caso não ocorra uma melhora na solução x, sinalizada com valor falso pelas variáveis $Impr_c$ e $Impr_L$, o procedimento LBFGS é acionado, intensificando a busca local na região com o uso do cálculo de derivada à primeira da função objetivo, que segundo os autores caracteriza o método como eficiente e robusto, comparado ao EC-GRASP.

Outro trabalho CGRASP-GEVAN que aborda a otimização contínua e recentemente foi apresentado por Birgin et al. (2010), possui estrutura semelhante ao procedimento EC-GRASP, mantendo a fase de construção e substituindo apenas o procedimento de busca local padrão pelo método GEVAN (BIRGIN; MARTÍNEZ, 2002). Este emprega gradientes espectrais com o objetivo de minimizar o valor da função objetivo, permitindo que o método encontre soluções altamente precisas que satisfazem a otimalidade. É importante lembrar que este trabalho, por maior que seja sua relevância, não será utilizado nesta pesquisa, já que o conjunto de funções objetivos utilizadas, 52 no total, diferem em 47 das que serão abordadas no Capítulo 6.

Desta forma, pode-se afirmar que as meta-heurísticas desenvolvidas para a resolução de problemas de otimização global contínua tem obtido bons resultados, tais como o C-GRASP, um método iterativo que não faz uso de cálculos de derivada, porém apresenta uma baixa convergência e o EC-GRASP, uma evolução do C-GRASP que permite uma melhor convergência na busca de boas soluções, porém não realiza uma exploração mais abrangente do espaço de soluções.

Dessa forma apresentamos no próximo capítulo o algoritmo GGVNS em sua forma contínua.

Capítulo 4

Continuous General Variable Neighborhood Search (C-GVNS)

Analisando os procedimentos aplicados aos problemas contínuos, o C-GRASP ainda que que desenvolva bom desempenho ao escapar de mínimos locais, possui grande dificuldade ao calcular direções de busca promissoras, acarretando uma convergência lenta, por utilizar um procedimento de construção aleatória.

Com o objetivo de solucionar tais dificuldades, Araújo, Cabral e Nascimento (2008) desenvolveram duas hibridizações baseadas no método C-GRASP, como apresentado respectivamente nas Seções 3.2 e 3.3. A primeira denominada EC-GRASP apresenta bons resultados quando comparados ao método C-GRASP, porém seu desempenho poderia ser elevado com a exploração de outras vizinhanças. Da mesma forma, ocorreu com o procedimento BC-GRASP, que manteve-se robusto e eficiente em comparação ao procedimento EC-GRASP, devido à utilização de cálculos de derivada de primeira ordem das funções objetivos.

Acreditando na combinação do procedimento C-GRASP com um procedimento de capacidade elevada para a exploração em grandes espaços de vizinhança, propomos uma nova meta-heurística híbrida, C-GGVNS, com o objetivo de manter uma boa convergência na busca local e ser capaz de investigar um número consideravelmente de vizinhanças.

O procedimento GGVNS discreto é uma generalização do procedimento VNS clássico, combinado ao método GRASP, apresentado na Seção 2.4.3, que utiliza em seu processo de busca local uma estratégia de descida, diferente do procedimento de busca local utilizado no procedimento GRASP, classificando o método como GVNS (HANSEN et al., 2010).

4.1 Método *Continuos* GGVNS (C-GGVNS)

O C-GGVNS é uma meta-heurística proposta para resolução de problemas de otimização contínua sujeito a restrições nos limites das variáveis. Assim como definimos na Seção 2.2, seja S um hiper retângulo e x um vetor, $S = \{x \in \mathbb{R}^n : l \leq x \leq u\}$, em que $l, u \in \mathbb{R}$, onde $u_i \geq l_i \; \forall \; i = 1, ..., n$. Desta forma, o problema é definido como:

$$\begin{aligned} &\min f(x), \\ &\text{Sujeito a: } l \leq x \leq u \end{aligned} \tag{4.1}$$

Possui características semelhantes ao procedimento EC-GRASP, por utilizar o procedimento C-GRASP como base para a geração de uma solução de boa qualidade, que posteriormente será refinada através do método GVNS contínuo, mantendo para isso o mesmo ciclo *multi-start* (Figura 4.1).

Desta forma, tendo o C-GRASP na primeira fase do procedimento (linha 8), e buscando melhorar a geração da solução inicial, foram propostas algumas alterações no procedimento de construção do método C-GRASP, de maneira que as coordenadas não fixas possam ser consultadas várias vezes, mesmo estas não obtendo sucesso em sua busca linear. Anteriormente, o procedimento descartava a coordenada caso esta não apresentasse bons resultados em determinada direção na busca linear, impossibilitando assim a verificação de outras trajetórias de busca linear. Além da modificação no procedimento de construção do C-GRASP, foram desenvolvidos dois métodos para o procedimento C-GGVNS: um para fase de perturbação e outro para fase de busca local GGVNS, que serão apresentadas nas Seções 4.1.1 e 4.1.2.

Figura 4.1: Pseudo código C-GGVNS

procedimento C-GGVNS (n, l, u, $f(.)$, h_s, h_e, ρ_{lo})

1. $f* \leftarrow \infty$;
2. **enquanto** ψ **faça**
3. $x \leftarrow k, k \in [l, u]$;
4. $h \leftarrow h_s$;
5. **enquanto** $h \geq h_s$ **faça**
6. $Impr_c \leftarrow$ falso;
7. $Impr_L \leftarrow$ falso;
8. [x] $\leftarrow$ CGRASP(x, $f(.)$, n, h, l, u, $Impr_c$);
9. [x] $\leftarrow$ GVNS(n, l, u, $f(.)$, f, x, $MaxViz$);
10. **se** $f(x) < f*$ **então**
11. $x* \leftarrow x$;
12. $f* \leftarrow f(x)$;
13. **fim se**
14. **se** $Impr_c$ = falso **e** $Impr_L$ = falso **então**
15. $h \leftarrow h/2$; /*faz a grade ficar mais densa*/
16. **fim se**
17. **fim enquanto**
18. **fim enquanto**
19. **retorne** $x*$;

fim `C-GGVNS`.

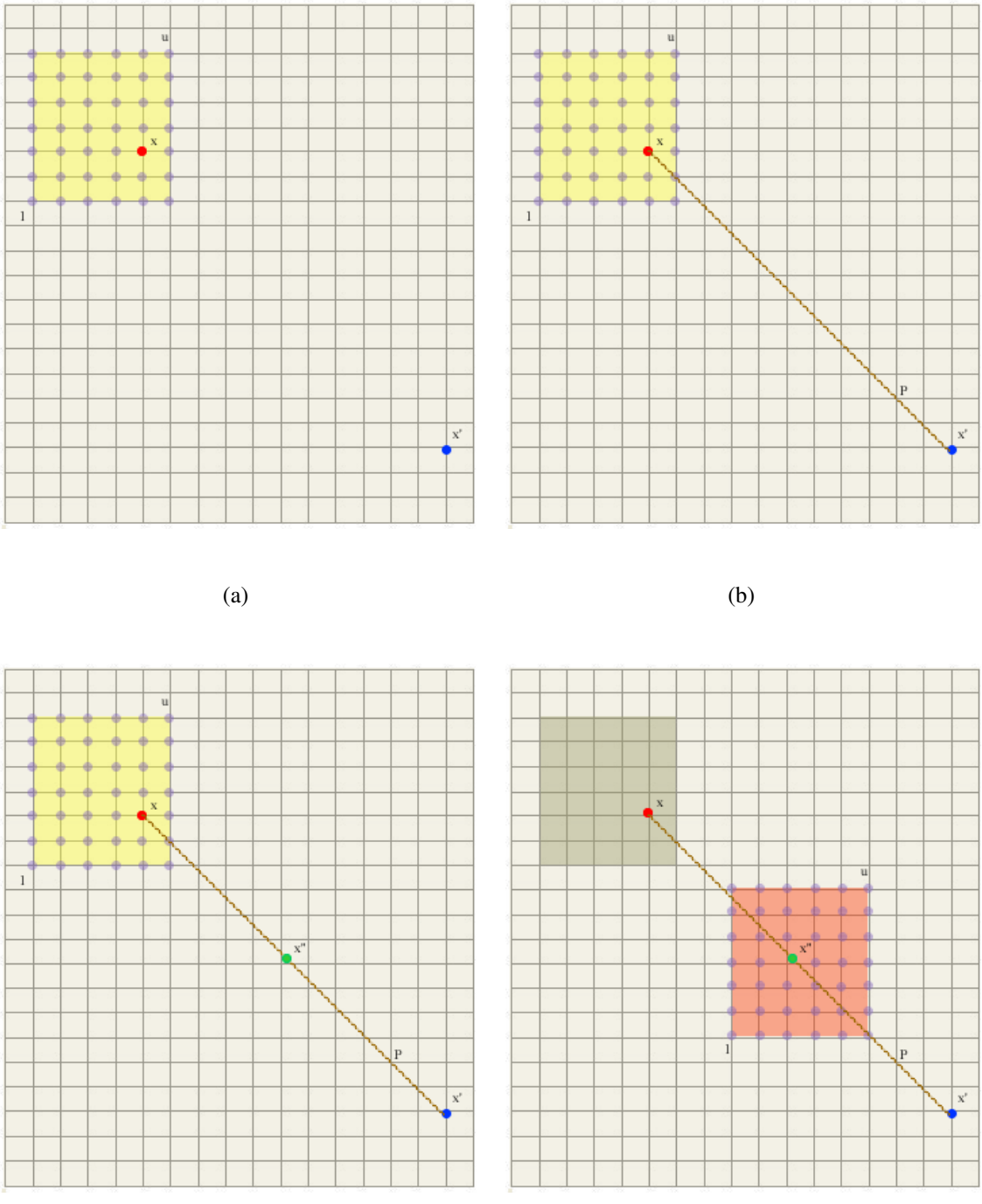
4.1.1 Procedimento de Perturbação

Contrariamente a outros métodos heurísticos, o procedimento GVNS e o GGVNS, em suas formas originais, como já mencionado anteriormente, promovem a troca sistemática de estruturas de vizinhanças, permitindo a exploração de vizinhanças gradativamente mais distantes da solução atual através do procedimento de Perturbação.

Desta forma, o procedimento de perturbação desenvolvido para o C-GGVNS parte da escolha aleatória de uma coordenada x' pertencente ao espaço de solução(Figura 4.2a), que,

ao traçar uma semi-reta que passa pela solução atual x (Figura 4.2b), gera uma nova solução x'' por meio do centro da semi-reta P(Figuras 4.2c e 4.2d).

Figura 4.2: Representação do procedimento de Perturbação C-GGVNS



O pseudo código do método é apresentado na Figura(4.3), em que y corresponde à uma coordenada no espaço de soluções ($grid$), e x'' o ponto médio pertencente à semi-reta que passa pelas coordenadas x e x'. É importante ressaltar que a escolha da coordenada x'' como sendo a solução corrente (linha 3) é realizada sem a validação de melhoria na qualidade da função objetivo ($f(x)$) como estratégia de exploração de estruturas de vizinhanças desconhecidas. que aparentemente não ofereçam mínimos locals, porém promovam o encontro de mínimos globais.

Figura 4.3: Pseudo código do procedimento de Perturbação C-GGVNS

procedimento Pertubacao_C-GGVNS(x, $f(x)$, $f(.)$, l,u)

1. $x' \leftarrow y \in \mathbb{R}^n$;
2. $x'' \leftarrow PontoMedio(x, x')$;
3. $x \leftarrow x''$;
4. $f(x) \leftarrow f(x'')$;
5. redefine(l, u);// redefine os limites superiores e inferiores
6. retorne x;

fim `Pertubacao_C-GGVNS`.

4.1.2 Procedimento de busca local

O procedimento de busca local, desenvolvido para o método C-GGVNS (Figura 4.4, possui características de descida direcional, por ser baseado no procedimento de busca APS, utilizado no método EC-GRASP 3.2.2. O procedimento é constituído por um conjunto de direções d, calculadas através da equação 3.3 (linha 1), utilizada no procedimento APS. Nestas direções, será aplicado o procedimento de busca busca linear, que, caso traga melhora no valor da função objetivo, ao encontrar uma solução vizinha melhor do que a solução corrente x, finaliza o procedimento.

Caso o processo de busca linear não conduza à descoberta de uma solução melhor, o processo armazenara todos os pontos x' visitados (linha 9), assim como a norma entre as coordenadas x e x' (linha 13), a cada passo. A finalidade deste processo é escolher a melhor solução entre os vizinhos e de forma aleatória realizar um processo de busca linear na direção

d. Isso é alcançado com passos discretizados de acordo com a norma escolhida (linha 19), e, em último caso de insucesso, é calculado a derivada primeira ($\nabla f(x)$) da função objetivo no ponto x (linha 25).

Figura 4.4: Pseudo código do procedimento de Busca Local C-GGVNS

procedimento BuscaLocal_C-GGVNS(x, $f(x)$, $f(.)$, l, u, h)

1.$d[] \leftarrow$ direção v utilizando a equação 3.3

2.**para** i=1..n **faça**

3. $x' \leftarrow$ BuscaLinear(x, $d[i]$, $f(x)$, h);

4. **se** ($f(x') < f(x)$) **então**

5. $x^* \leftarrow x'$;

6. $f(x^*) \leftarrow f(x')$;

7. **retorne** x^*;

8. **fim se**

9. $y[] \leftarrow x'$;

10. **fim para**

11. $x' \leftarrow \infty$;

12. **para** i=1..n **faça**

13. $norma[] \leftarrow$ (x,$y[i]$,n);

14. **se** $f(x') > f(y[i])$ **entao**

15. $x' \leftarrow y[i]$; //escolhe o melhor vizinho

16. **fim se**

17. **fim para**

18. $k \leftarrow Random(1, n)$;

19. $x' \leftarrow$ BuscaLinear(x', $d[k]$, $f(x)$, $norma[k]$);

20. **se** ($f(x') < f(x)$) **então**

21. $x^* \leftarrow x'$;

22. $f(x^*) \leftarrow f(x')$;

23. **fim se**;

24. **senão**

25. $x^* \leftarrow \nabla f(x)$;

26. retorne x^*;

fim `BuscaLocal_C-GGVNS`.

Segundo Ahmed (2004), a utilização do cálculo do gradiente de algumas funções ob-

jetivos são incapazes de serem calculadas, ou acarretam um consumo elevado de tempo computacional. Desta maneira, realizaremos a construção de dois procedimento C-GGVNS, que primeiramente chamaremos de C-GGVNS$_1$ que não possui as linhas 24 e 25 do procedimento de busca local (Figura 4.4) e C-GGVNS$_2$ que faz uso do cálculo do gradiente, linhas 24 e 25 do mesmo pseudo código.

O desempenho computacional deste procedimento será apresentado na Seção 6.2, presente no capítulo 6, porém antes, o próximo capítulo apresentará o segundo procedimento construído, utilizando como base o algoritmo C-GRASP.

Capítulo 5

Paralela Construção para C-GRASP

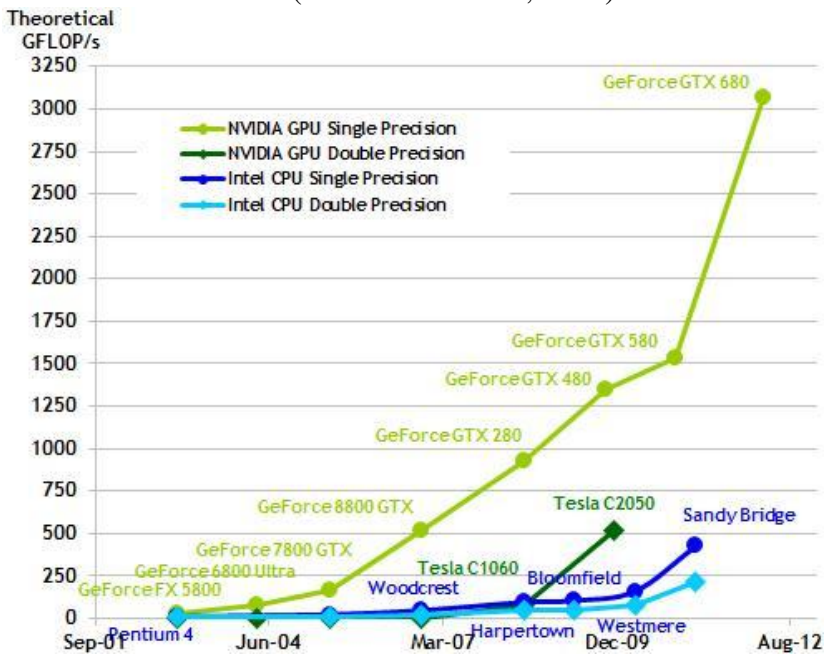
Muitas tecnologias têm evoluído e alcançado novos níveis de aplicabilidade e poder computacional, com o objetivo de solucionar e apoiar problemas de Otimização, problemas de arquiteturas maciçamente paralelas, assim como o processamento de vídeos e processamento vetorial.

Desta maneira, a plataforma de computação paralela *Compute Unified Device Architecture* (CUDA), segundo Hwu Christopher Rodrigues e Stratton (2009), oferece amplo e total acesso e manipulação do hardware do escopo industrial de *Graphics Processing Unit's* [1] (GPUs) desenvolvidos pela farbicante nVidia [2], que apresentaram melhorias substanciais e muito superiores comparado às *Central Processing Units* (CPU's), como pode ser observado na Figura 5.1

[1] Tipo de microprocessador especializado em processar gráficos em computadores pessoais.
[2] Multinacional, fabricante de componentes de computadores.

Figura 5.1: Cálculo de Pontos Flutuantes por Segundo - CPU x GPU

Fonte:(CORPORATION, 2013)



A principal razão de tamanha discrepância no desempenho computacional entre a GPU e a CPU é que a GPU foi elaborada para trabalhar sobre uma computação intensiva, altamente paralela e, portanto, foi concebida e projetada de tal forma que mais transistores são dedicados ao processamento de dados, ao invés de *cachê* de dados e controle de fluxo, o que toma muito espaço no chip final (Figura 5.2).Mas para obter este desempenho, os problemas devem ter computação de dados paralelos.

Figura 5.2: Diferenças na arquitetura de CPU e GPU

Fonte:(CORPORATION, 2013)

5.1 Fundamentos de CUDA

O modelo de programação CUDA é baseado na linguagem de programação C, possuindo uma baixa curva de aprendizado e amplo acesso a toda arquitetura das GPU's, como memórias compartilhadas e sincronismo, que são expostos ao programador como um conjunto mínimo de extensões da linguagem C (CORPORATION, 2013).

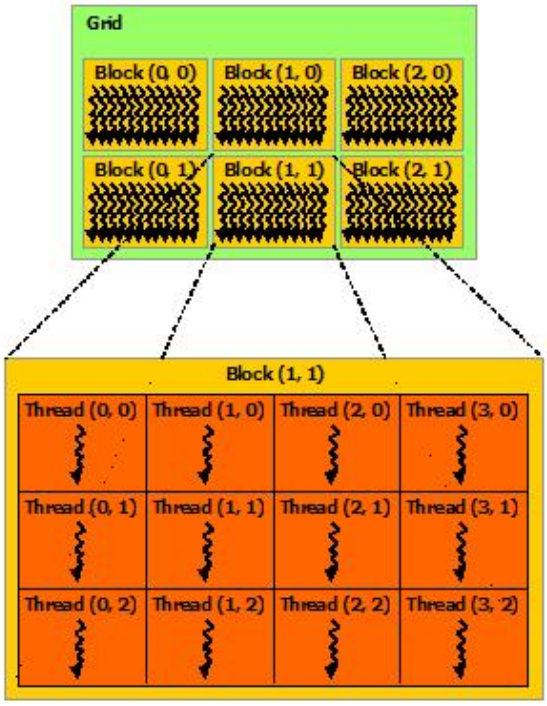
CUDA C estende C de modo que permite definir funções C, chamadas de kernels, que são executadas N vezes em N diferentes CUDA *threads*, e cada thread pode ser diferenciada pelo seu identificador, com a variável *threadIdx*; unidimensional, bidimensional ou tridimensional, para conveniência de computação de problemas com estas dimensionalidades.

As *threads* são agrupadas por blocos, que atualmente podem conter até 1024 *threads* residindo no mesmo multiprocessador, e também podem ser identificadas pela variável *blockIdx*, igualmente com possibilidade de até três dimensionalidades.

Por fim, os blocos são organizados em *grids* unidimensionais, bidimensionais ou tridimensionais, como ilustrado na Figura 5.3. O número de blocos em um *grid* é usualmente ditado pelo tamanho do dado a ser processado ou o número de processadores no sistema. Um exemplo disto é o processamento de imagens, em que cada *thread* pode executar o processamento de um pixel, logo o tamanho total da imagem definirá o tamanho final do *grid* a depender do tamanho dos blocos de *threads*.

Figura 5.3: *Grid*, blocos e *threads*

Fonte:(CORPORATION, 2013)



Após instanciação do *grid*, o processamento dos blocos é transparente ao programador.

5.2 Método Proposto

Observando C-GRASP apresentado no Capítulo 3, é verificado que este expõe certa dificuldade de convergência, à medida que o encontro de uma solução, que concederia à função objetivo uma melhora, não ocorre, pois a grade projetada com limites h_s e h_e torna-se mais densa a cada passo, e esta característica torna-se evidente a medida em que as funções utilizadas possuem alta dimensionalidade. Com o objetivo de minimizar este esforço computacional, foi proposto uma nova abordagem para a construção do procedimento C-GRASP.

5.2.1 Fase de Construção

Experimentos realizados evidenciaram os aspectos apresentados no início desta seção isto é, utilizando funções bidimensionais, constatou-se que menos de 10% do tempo total gasto foi atribuído ao procedimento de construção, enquanto que para funções de 30 dimensões, mais

de 40% do tempo total é dedicado. Desta maneira, a fase de construção é revelada como uma parte crítica da meta-heurística C-GRASP para resolução de problemas com grandes dimensões.

Como exposto na Seção 5.1, as GPUs apresentam uma solução de hardware de baixo custo e alto poder computacional disponível para solucionar problemas. Portanto, a segunda proposta apresentada trata da paralelização do procedimento de construção do C-GRASP padrão, devido a seu lento desempenho promovido à medida que ocorre um aumento na dimensionalidade das funções utilizadas. No entanto, poucos algoritmos podem ser diretamente traduzidos para execuções em GPUs sem exigirem mudança em sua estrutura. Deste modo, um grande cuidado é necessário para obtenção de qualquer ganho de eficiência quando utilizado este tipo de arquitetura paralela. Então, para garantir boa eficácia em GPUs, foi necessário organizar as estruturas de dados do algoritmo, adaptando sua arquitetura para GPU.

A parte dispendiosa na fase de construção C-GRASP padrão (mostrada na Figura 3.2) é o laço entre as linhas 6 e 15, onde uma pesquisa linear é executada em cada direção sobre as coordenadas não-fixas. O algoritmo paralelo poderia ser apenas uma versão paralela deste loop, mas a gestão das coordenadas não-fixas de CPU para GPU promoveria grande sobrecarga (*overhead*).

Desta maneira, a maior parte do processo de construção proposto é feito na GPU. O procedimento $ContruoParalela$, presente na Figura 5.4, promove primeiro uma cópia dos vetores importantes da CPU para a memória da GPU, que inicializa a lista de coordenadas não-fixas na memória da GPU (chamando um núcleo de inicialização) e, finalmente, chama o procedimento de construção em GPU (Figura 5.5), com n *threads*. Ao final, o vetor resultante x é copiado para a memória de CPU que continuará o processo do C-GRASP padrão em CPU. Nota-se que os vetores dos limites l e u não mudam durante o processo de otimização C-GRASP, por isso pode ser copiado uma única vez para a memória da GPU, como na linha 1 da Figura 5.4.

Figura 5.4: Pseudo código da construção paralela proposta para C-GRASP

procedimento ContruçãoParalela(x, $f(.)$, n, h, l, u, $Impr_c$)

1. Copia x, l and u para memoria GPU
2. Inicialização *nFixa* em GPU
3. ConstruçãoKernel$<<<n>>>$(x, $f(.)$, n, h, l, u);
4. Copia Resultados x da memoria GPU para memoria principal
5. **returna**(x, $Impr_c$);

fim `Contrução-Paralala-CGRASP`.

O procedimento $ConstruoKernel$ chamado pelo algoritmo $ContruoParalela$ é mostrado na Figura 5.5. O kernel gerencia as coordenadas não-fixas e chama o $BuscaLinearKernel$, para cada *thread*, utilizando seu índice como parâmetro.

O procedimento $BuscaLinearKernel$ verifica as coordenadas não-fixas, realizando uma busca linear à procura de uma solução que ofereça o mínimo valor para a função objetivo; como um detalhe de implementação, a lista de coordenadas não-fixas é armazenada como um vetor de booleanos.

Figura 5.5: Pseudo código do procedimento de Construção para C-GRASP no kernel

procedimento ConstruçãoKernel<<<n>>>(x, $f(.)$, n, h, l, u)

3. $ReUse \leftarrow$ falso;
4. **enquanto** $nFixa \neq \emptyset$ **faça**
5. $g \leftarrow +\infty$;
6. $\bar{g} \leftarrow -\infty$;
7. BuscaLinearKernel($threadIdx$, x, h, i, n, $f(.)$, l, u, $nFixa$);
8. LCR $\leftarrow \emptyset$;
9. threshold $\leftarrow g + \alpha(\bar{g} - g)$;
10. **Para** i = 1 ... n **faça**
11. **se** $i \in nFixa$ **E** $g_i \leq threshold$ **então**
12. LCR $\leftarrow$ LCR $\cup\, i$;
13. **fim se**
14. **fim para**
15. $j \leftarrow$ ElementoAleatorio(LCR);
16. **se** $x_j = z_j$ **então** ReUse $\leftarrow$ verdadeiro;
17. **senão**
18. $x_j \leftarrow z_j$;
19. ReUse $\leftarrow$ falso;
20. $Impr_c \leftarrow$ verdadeiro;
21. **fim senão**
22. $nFixa \leftarrow nFixa \setminus \{j\}$;
23. **fim enquanto**
24. **returne**(x, $Impr_c$);

fim `ConstruçãoKernel`.

A partir da apresentação destes procedimentos, no próximo capítulo, será exposta a verificação dos seus desempenhos, através de experimentos computacionais realizados.

Capítulo 6

Resultados Computacionais

Este Capítulo apresentará os resultados obtidos nos experimentos realizados para a validação do método proposto C-GGVNS, com suas variações apresentadas na Seção 4.1, e a comparação envolvendo os procedimentos EC-GRASP e BC-GRASP (apresentados no Capítulo 3).

Os procedimentos serão avaliados quanto à média do valor da função objetivo, seu valor mínimo, estabelecendo um número máximo de execução, e em relação ao esforço computacional, medido em número de chamadas ao procedimento de avaliação da função objetivo e cálculo do gradiente da função. A Seção 6.2 tratará da apresentação dos experimentos computacionais e das discussões dos resultados obtidos entre estes procedimentos.

Já o procedimento C-GRASP, com fase de construção paralela (Seção 5.2) será comparado com sua versão padrão (Seção 3) em experimentos de teste de avaliação média do tempo de execução. A Seção 6.3, apresentará o desempenho computacional e as discussões dos resultados.

O cenário utilizado para a execução dos experimentos e os parâmetros de entrada utilizados em cada procedimento, será apresentado na Seção 6.1.

6.1 Ambiente de Teste

Os experimentos computacionais referentes à execução das variações do C-GGVNS foram executados em um processador Intel Core 2 Duo, tendo 8GB de memória RAM, sob um sistema operacional Ubuntu 13.04 Raring Ringtail. Os algoritmos foram implementados

utilizando a linguagem de programação C++ no compilador GNU GCC 4.8.1. Já os experimentos para determinar o desempenho do algoritmo C-GRASP com construção paralela foram executados em uma máquina com processador Core i3 da Intel, com 4GB de memória RAM e GPU GeForce 330M. Foi utilizada linguagem CUDA C e a versão sequencial em linguagem C++, executada também no mesmo ambiente.

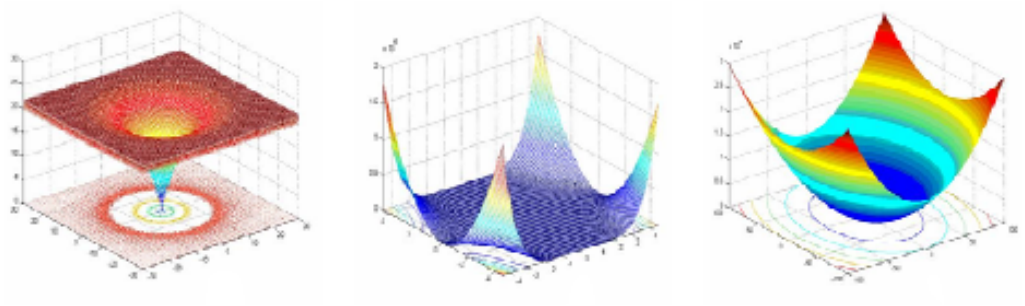
As funções de teste utilizadas nos experimentos de análise do algoritmo C-GGVNS (Tabela 6.1) são parte do conjunto de funções utilizadas por Araújo, Cabral e Nascimento (2008). O Apendice A apresenta a definição de cada função utilizada.

A representação geométrica das funções para duas dimensões (n=2) estão presentes nas Figuras 6.1 e 6.2, com exceção das Funções Hartmann, Levy, Powell, Power Sum e Shekel.

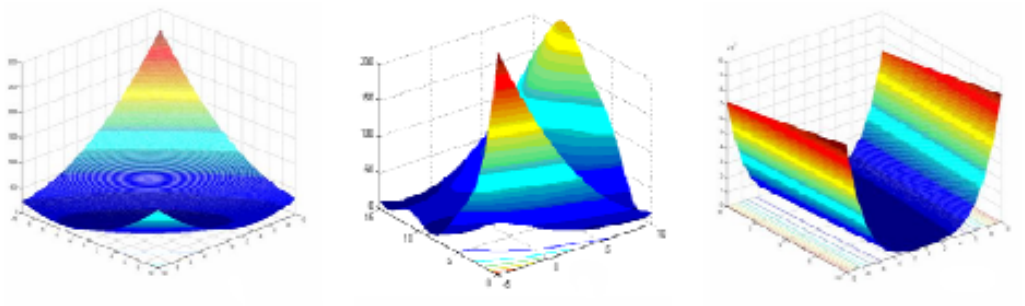
Tabela 6.1: Funções teste utilizadas

Função	**Dimensão**	**Função**	**Dimensão**
Ackley (A_{30})	30	Rastrigin (RA_{10})	10
Beale (BE)	2	Rastrigin (RA_{20})	20
Bohachevsky (B_2)	2	Rosenbrock (R_2)	2
Booth (BO)	2	Rosenbrock (R_{10})	10
Branin (BR)	2	Rosenbrock (R_{20})	20
Colville (CV)	3	Schwefel (SC_2)	2
Dixon and Price (DP_{25})	25	Schwefel (SC_6)	6
Easom (EA)	2	Shekel ($S_{4,10}$)	10
Goldstein and Price (GP)	2	Shekel ($S_{4,5}$)	5
Griewank (GR_{10})	10	Shekel ($S_{4,7}$)	7
Griewank (GR_{20})	20	Shubert (SH)	2
Hartmann ($H_{3,4}$)	3	Sphere (SP_3)	3
Hartmann ($H_{6,4}$)	6	Sphere (SP_{30})	30
Six-Hump Camelback (CA)	2	Sum Squares (SS_{10})	10
Levy (L_{30})	30	Sum Squares (SS_{20})	20
Matyas (M)	2	Trid (T_{10})	10
Perm ($P_{4,\frac{1}{2}}$)	4	Trid (T_6)	6
$Perm_0$ ($P_{4,10}$)	4	Zakharov (Z_{10})	10
Powell Sum ($PS_{4,\{8,18,44,114\}}$)	24	Zakharov (Z_{20})	20

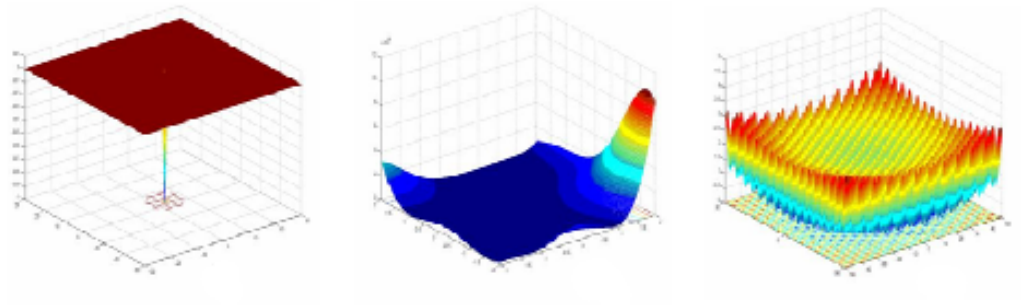
Figura 6.1: Representação geométrica de um conjunto de funções de teste para n=2. (a) Ackley; (b) Beale; (c) Bohachevsky; (d) Booth; (e) Branin; (f) Dixon and Price; (g) Easom; (h) Goldstein and Price; (i) Griewank; (j) Six-Hump Camelback; (l) Matyas; (m) Perm.



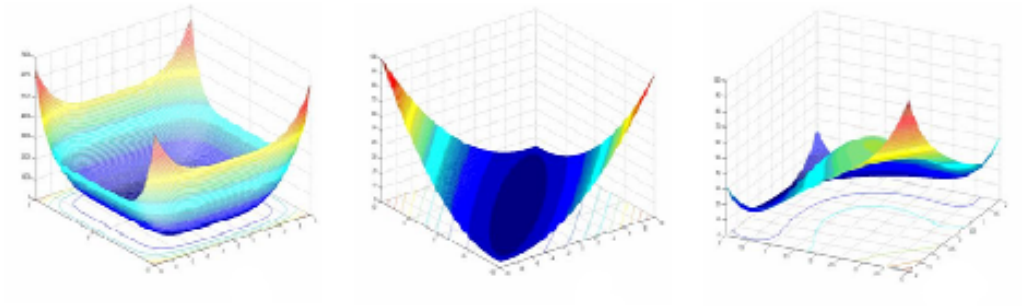
(a) (b) (c)



(d) (e) (f)

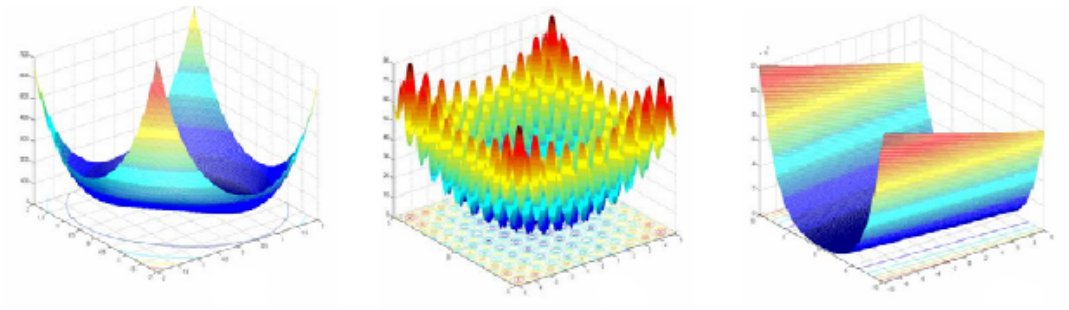


(g) (h) (i)

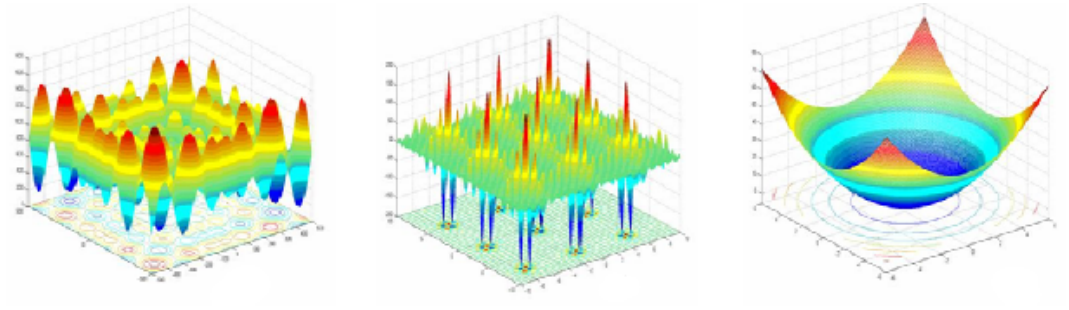


(j) (k) (l)

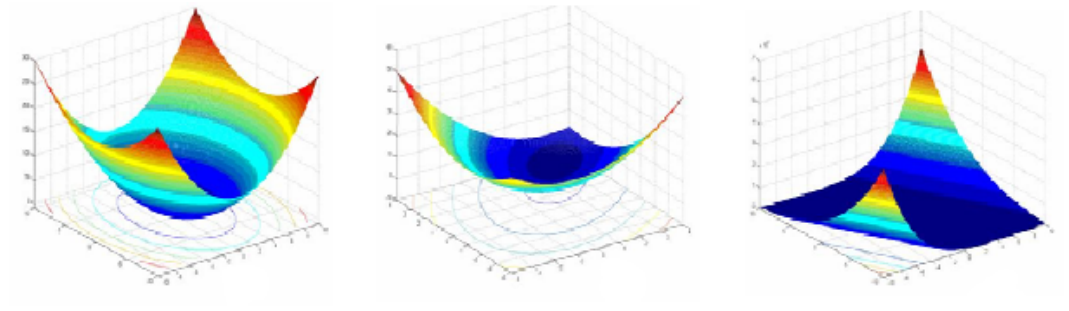
Figura 6.2: Representação geométrica de um conjunto de funções de teste para n=2. (a) Perm_0; (b) Rastrigin; (c) Rosenbrock; (d) Schwefel; (e) Shubert; (f) Sphere; (g) Sum Squares; (h) Trid; (i) Zakharov.



(a) (b) (c)



(d) (e) (f)



(g) (h) (i)

Os parâmetros h_s e h_e utilizado nos procedimentos C-GGVNS$_1$, C-GGVNS$_2$, EC-GRASP e BC-GRASP e C-GRASP sequencial e paralelo são apresentados na Tabela 6.2. Já o parâmetro m, utilizado no procedimento BC-GRASP, foi fixado em 2.

Tabela 6.2: Valor dos parâmetros h_s e h_e utilizado nos procedimentos

Função	h_s	h_e	Função	h_s	h_e
A_{30}	5,0	0,05	RA_{10}	0,5	0,1
BE	1,0	0,5	RA_{20}	0,5	0,1
B_2	1,0	0,5	R_2	1,0	0,1
BO	1,0	0,5	R_{10}	1,0	0,1
BR	1,0	0,001	R_{20}	0,1	0,05
CV	1,0	0,1	SC_2	5,0	0,25
DP_{25}	5,0	0,25	SC_6	50,0	0,25
EA	1,0	0,1	$S_{4,10}$	1,0	0,5
GP	1,0	1,0	$S_{4,5}$	1,0	0,5
GR_{10}	10,0	0,25	$S_{4,7}$	1,0	0,5
GR_{20}	10,0	0,25	SH	1,0	0,01
$H_{3,4}$	0,5	0,001	SP_3	2,0	0,05
$H_{6,4}$	0,1	0,01	SP_{30}	2,0	0,05
CA	1,0	0,01	SS_{10}	1,0	0,5
L_{30}	1,0	0,1	SS_{20}	1,0	0,5
M	1,0	0,1	T_{10}	5,0	0,1
$P_{4,\frac{1}{2}}$	1,0	0,0125	T_6	1,0	0,1
$P_{4,10}$	1,0	0,01	Z_{10}	1,0	0,005
$PS_{4,\{8,18,44,114\}}$	1,0	0,1	Z_{20}	5,0	0,05

6.2 Experimentos Computacionais C-GGVNS e Outras Meta-heurísticas

Foram realizados quatro experimentos computacionais, descritos a seguir.

O primeiro consiste da comparação da eficiência dos procedimentos através do GAP médio entre as meta-heurísticas testadas, em que, quanto menor o valor do GAP, mais próxima a meta-heurística estará do valor mínimo global. O cálculo utilizado para a obtenção do GAP está representado na Equação 6.2.

$$GAP = \|f(x) - f(x*)\| \leq 10^{-4} \tag{6.1}$$

Cada meta-heurística foi executada 100 vezes para 100, 500, 1.000, 5.000, 10.000, 20.000 e 50.000 avaliações da função objetivo, conforme os experimentos realizados por Araújo, Cabral e Nascimento (2008), com critério de parada estabelecido para o encontro da solução significativamente próxima do ótimo, ou a conclusão de 100 ciclos *multi-start*.

Observando a Tabela B.1 e a coluna de 100 avaliações da função na Tabela 6.3, é verificado um bom desempenho do procedimento C-GGVNS$_2$ em comparação aos demais, com o encontro de 14 melhores valores para a função objetivo, onde em 9 funções, as quatro meta-heurísticas obtiveram desempenho igual, tenho a EC-GRASP 2 sucessos, C-GGVNS$_1$ 8 e o BC-GRASP 5.

O bom desempenho referente a meta-heurística C-GGVNS$_2$ prossegue como pode ser observado nos resultados (Tabelas B.2, B.3, B.4, B.5, B.6 e B.7), presentes no Apêndice B, cujo o número total de sucesso de cada procedimento pode ser verificado na Tabela 6.3. Além do desempenho do C-GGVNS$_2$ em 98 casos de sucesso e 89 casos de empate, quando comparado as meta-heurísticas EC-GRASP, BC-GRASP e C-GGVNS$_1$, o bom desempenho do C-GGVNS sem cálculo do gradiente (o C-GGVNS$_1$), também ganha destaque com a obtenção de 40 casos de sucesso.

Tabela 6.3: Total de sucessos para os 38 problemas testes

	Avaliação da Função						
Meta-Heurística	100	500	1.000	5.000	10.000	20.000	50.000
EC-GRASP	2	1	2	2	3	1	1
C-GGVNS$_1$	8	0	9	4	6	6	7
C-GGVNS$_2$	14	24	16	12	10	12	10
BC-GRASP	5	6	1	5	5	2	3

Verificado o bom desempenho das duas versões do procedimento C-GGVNS, através do GAP médio obtido, as meta-heurísticas comparadas foram submetidas ao segundo experimento, que por meio de 100 iterações, utilizando os mesmos parâmetros de h_s e h_e presentes

na Tabela 6.2, com critério de parada estabelecido para o encontro do menor GAP, alcançaram os resultados presentes na Tabela 6.4.

Tabela 6.4: Chamada à função objetivo e gradiente em 100 iterações

FUNÇÃO	**C-GGVNS$_1$**	**EC-GRASP**	**C-GGVNS$_2$**		**BC-GRASP**	
			Ava.Função	**Ava.Gradiente**	**Ava.Função**	**Ava.Gradiente**
A$_{30}$	6950	6935	6937	0	6957	0
BE	108	100	**105**	0	100	0
B$_2$	476	476	476	0	476	0
BO	84	84	84	0	84	0
BR	600	474	540	24	694	26
EA	4690	2589	3372	2	1075	13
GP	35	35	35	0	35	0
GR$_{10}$	5199	5224	5234	0	5215	0
GR$_{20}$	20095	20118	**20058**	0	20090	0
H$_{3,4}$	512	650	**473**	23	808	23
H$_{6,4}$	413	419	1065	0	1403	70
CA	2272	3576	**344**	0	3930	0
L$_{30}$	12052	12053	12063	0	12062	0
M	74	73	74	0	72	0
P$_{4,\frac{1}{2}}$	379289	117704	**4588**	0	33578	3
P$_{4,10}$	133	134	**130**	0	132	0
PS$_{4,\{8,18,44,114\}}$	4589	4592	4610	0	4587	0
R$_2$	72	72	72	0	72	54
R$_{10}$	1074	1057	1079	0	1056	71
R$_{20}$	67206	66323	**66250**	307	67042	351
S$_{4,5}$	149	434	148	69	147	115
SH	909	944	**172**	22	271	11
SP$_3$	7352	2060	**323**	5	493	5
SS$_{20}$	4395	4412	**4385**	0	4409	0
SS$_{10}$	1070	1046	1057	0	1069	0
Z$_{20}$	2886	2723	3030	0	2659	6
Z$_{10}$	3247	4088	**3086**	0	3309	5

O número mínimo de chamada à função objetivo não é a única métrica conveniente, devido aos procedimento BC-GRASP e C-GGVNS$_2$, utilizarem o cálculo de derivada de

primeira ordem. Porém, mesmo observando o número de avaliações do gradiente, tanto para a meta-heurística C-GGVNS$_2$, como para BC-GRASP, o bom desempenho do procedimento C-GGVNS$_2$ é verificando quando comparado isoladamente à EC-GRASP e C-GGVNS$_1$ através do número de avaliação da função objetivo, assim como o BC-GRASP, com o número inferior de avaliação do gradiente e função objetivo.

O terceiro experimento realizado consiste da aplicação de 34 funções a 50, 100, 500, 1.000 iterações, com critério de parada estabelecido para o encontro da solução ótima global, o resultado da convergência pode ser observados nas Tabelas B.8 e B.9 , presentes no Apêndice B.

Verificando os dados obtidos, constata-se que o único procedimento que obteve convergência em todos os casos de testes foi o C-GGVNS$_2$, em que as funções Colville, Dixon-price, Schwefel e Trid não obtiveram nenhuma convergência junto às outras meta-heurísticas. Além disso, apresentou o maior número de convergências 100%, totalizando 85 casos, contra 78 do BC-GRASP e 72 do EC-GRASP e C-GGVNS$_1$. Acredita-se que este bom desempenho é devido à exploração de regiões de vizinhanças promissoras.

Com o objetivo de avaliar o efeito de alta dimensionalidade do método C-GGVNS$_2$ em comparação a BC-GRASP, o ultimo experimento utilizou as funções Rosenbrock (R$_n$) e Zankharov (Z$_n$), em altos valores de n (dimensões). A função Rosenbrock foi avaliada em 20, 50, 100 e 500 dimensões (R$_{20}$, R$_{50}$, R$_{100}$ e R$_{500}$, respectivamente) e a função Zankharov em 20, 50 e 100 (Z$_{20}$, Z$_{50}$ e Z$_{100}$, respectivamente), conforme experimentos realizados por Araújo, Cabral e Nascimento (2008).

Diferentemente dos experimentos anteriores, cada função foi analisada no intervalo de 10 à 80 iterações, tendo o critério de parada estabelecido no número máximo de iterações multi-start ou o encontro da solução x^*. Conforme Araújo, Cabral e Nascimento (2008), o parâmetro $MaxIters$ foi configurada para os dois métodos como $2n$, e o parâmetro m utilizado no procedimento BC-GRASP foi configurado com o valor 7, para todos os problemas testes. Os dois métodos utilizaram os parâmetros de discretização h_s e h_e, apresentados na Tabela 6.5.

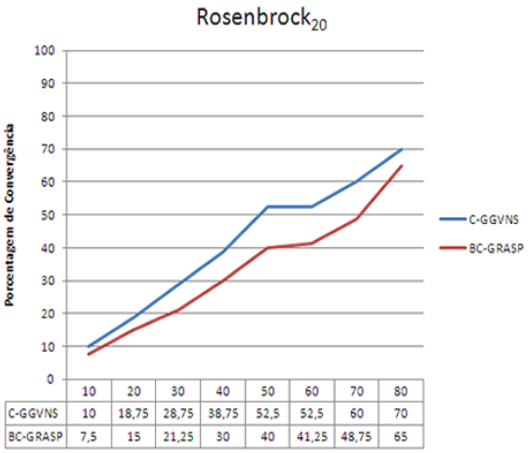
Tabela 6.5: Parâmetros de discretização dos métodos BC-GRASP e C-GGVNS$_2$, para análise em alta dimensionalidades

Problema Teste	h_s	h_e
R_{20}	1,0	0,1
R_{50}	1,0	0,1
R_{100}	1,0	0,1
R_{500}	1,0	0,01
Z_{20}	2,5	0,05
Z_{50}	5,0	0,005
Z_{100}	5,0	0,005

De acordo com os resultados presentes nas Tabelas B.10 e B.11, no Apendice B, todas as funções convergiram em 100% para todas iterações com exceção das R_{20}, R_{500}, Z_{100}, como ilustrado na Figura 6.3. Observando o desempenho dos algoritmos para a função R_{20}, é verificado que nenhum procedimento para o conjunto de iterações (10 à 80), conseguiu convergir em 100%, ficando a meta-heurística C-GGVS$_2$, com o maior valor de convergência. Contudo, ao analisar as funções R_{500} e Z_{100}, o procedimento BC-GRASP possui melhor convergência, apesar dos dois algoritmos desempenharem convergência equivalentes no início do processo de execução.

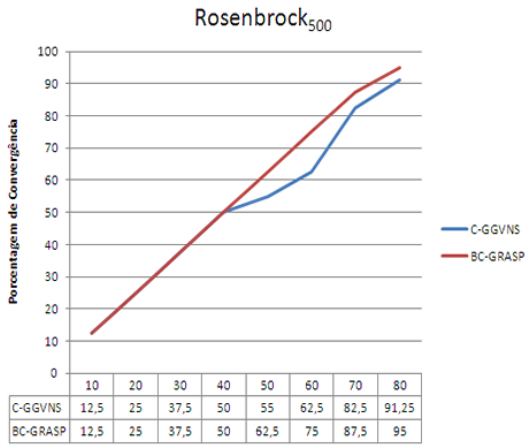
Desta maneira, os resultados apresentados nesta seção comprovam a eficiência e robustez do procedimento C-GGVNS$_2$, mesmo utilizando do cálculo do gradiente da função objetivo, para análise de funções de baixa convergência, tendo porém pequenas dificuldades ao trabalhar funções de dimensões elevadas.

Figura 6.3: Representação do procedimento de Perturbação C-GGVNS



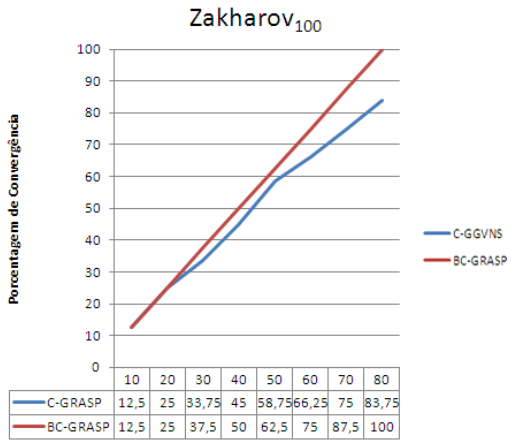
	10	20	30	40	50	60	70	80
C-GGVNS	10	18,75	28,75	38,75	52,5	52,5	60	70
BC-GRASP	7,5	15	21,25	30	40	41,25	48,75	65

(a)



	10	20	30	40	50	60	70	80
C-GGVNS	12,5	25	37,5	50	55	62,5	82,5	91,25
BC-GRASP	12,5	25	37,5	50	62,5	75	87,5	95

(b)



	10	20	30	40	50	60	70	80
C-GRASP	12,5	25	33,75	45	58,75	66,25	75	83,75
BC-GRASP	12,5	25	37,5	50	62,5	75	87,5	100

(c)

6.3 Experimentos Computacionais C-GRASP Sequencial e Paralelo

A primeira fase de testes foi realizada com funções de baixa dimensionalidade (menos de 20 dimensões), apresentadas na Tabela 6.1. Neste caso foi comprovado que a versão com construção paralela não obteve aceleração significativa sobre a versão sequencial. Ao medir

o impacto da fase de construção do algoritmo C-GRASP padrão para as funções de baixa dimensionalidade, verificou-se que a fase de construção pode consumir tão pouco, como 10% do tempo total de execução para o algoritmo. Desta maneira, melhorar a fase de construção sobre esses tipos de problemas é, significativamente, ineficazes.

No entanto, ao verificar o desempenho do C-GRASP padrão para funções objetivo de mais de 20 dimensões, observou-se que a fase de construção poderia levar até 30% ou até mesmo 40% do tempo total de execução. Foi este comportamento, já descrito na Seção 5.2, que motivou a construção do experimento de análise de desempenho do algoritmo C-GRASP com construção paralela envolvendo funções com dimensionalidade elevada (GR_{20}, RA_{20}, PS_{24}, DP_{25}, SP_{30} e L_{20}).

Os resultados são apresentados na Tabela 6.6, que revela os tempos médios para dez execuções do algoritmo em cada função objetivo. Os mesmos resultados são apresentados graficamente nas Figura 6.4.

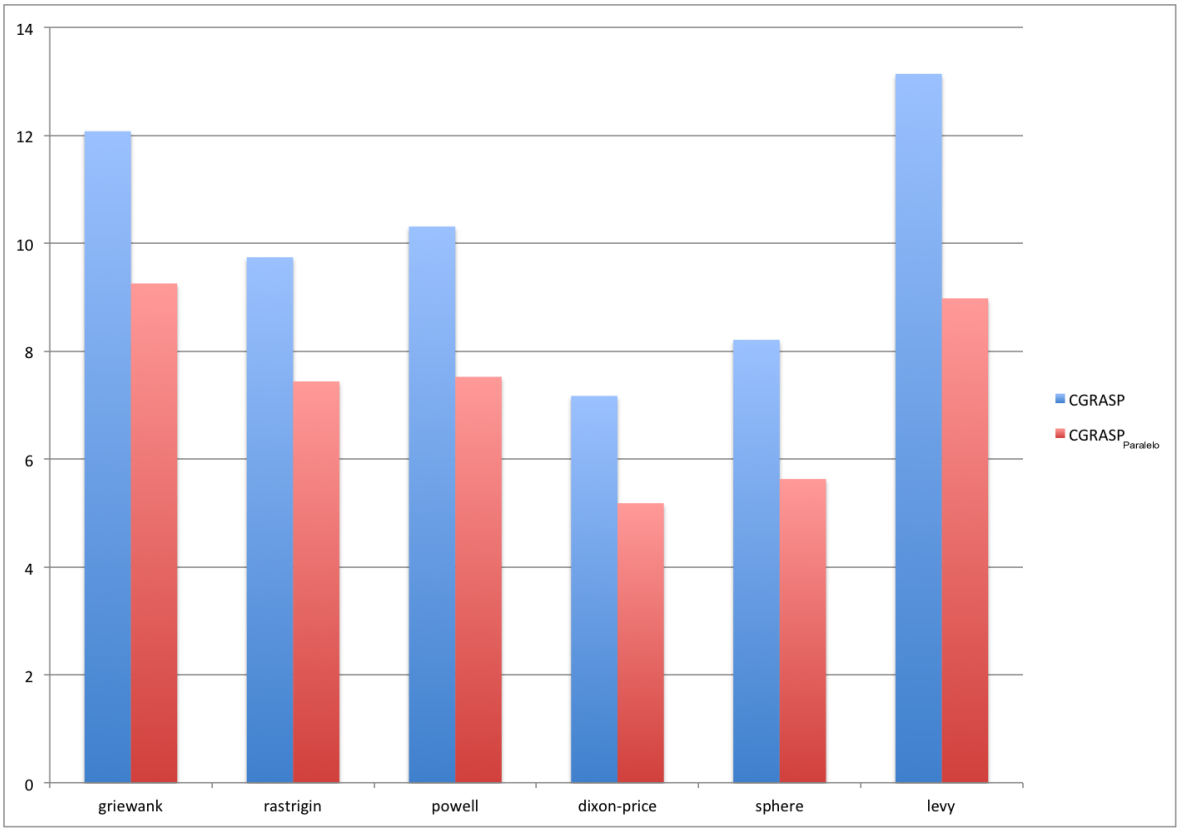
Tabela 6.6: Desempenho em relação ao tempo médio entre C-GRASP padrão e sua versão com construção paralela

Funções	C-GRASP	C-GRASP Paralelo	Speedup
GR_{20}	12.076s	9.254s	1.3049
RA_{20}	9.740s	7.439s	1.3093
PS_{24}	10.312s	7.524s	1.3726
DP_{25}	7.170s	5.181s	1.3832
SP_{30}	8.212s	5.630s	1.4582
L_{20}	13.144s	8.978s	1.4636

Estes resultados preliminares indicam claramente melhorias na aplicação da versão C-GRASP utilizando a construção paralela. Outra tendência que é revelada na observação dos dados é que a aceleração aumenta em conjunto com a dimensionalidade da função objetivo. Porém, o desempenho em tempo desta abordagem não depende apenas da dimensionalidade da função, pois algumas funções são mais difíceis do que outras (a função SP_{30}, por exemplo, é apenas uma hiper-esfera com 30 dimensões, que com grande facilidade pode ser otimizada).

É também importante notar que a dimensão das funções de teste utilizadas ainda é muito baixa para muitas áreas de aplicação - como computacional química e máquina de aprendizagem - onde as funções podem possuir centenas de dimensões. Assim, podemos esperar que o procedimento proposto obtenha um desempenho ainda melhor aplicado a estes tipos de problemas.

Figura 6.4: Representação gráfica do desempenho entre C-GRASP padrão e sua versão com construção paralela



Capítulo 7

Considerações Finais

O presente trabalho, propos a implementação de uma abordagem sequencial e outra paralela para o procedimento *Continuous Greedy Randomized Adaptive Search Procedure* (C-GRASP) (HIRSCH et al., 2007), na resolução de problemas de Otimização Global Contínua.

A primeira meta-heurística, sequencial e híbrida foi elaborada motivada pela dificuldade de exploração do espaço de vizinhança por alguns métodos presentes na literatura. Composta pela junção do procedimento C-GRASP padrão e a meta-heurística *General Variable Neighborhood Search* (C-GGVNS), que para fins de avaliação foi denominada de C-GGVNS$_1$, que não utiliza cálculo do gradiente, e C-GGVNS$_2$, que faz uso do cálculo.

A segunda proposta do presente trabalho, é a elaboração da primeira versão do procedimento C-GRASP paralelo, utilizando a plataforma *Compute Unified Device Architecture* (CUDA). O procedimento C-GRASP teve sua fase de construção paralelizada, devido ao consumo elevado do tempo de processamento realizado pelo procedimento em análise de funções com alta dimensionalidade.

Para a validação dos métodos C-GGVNS, estes foram comparados aos procedimentos EC-GRASP e BC-GRASP (ARAÚJO; CABRAL; NASCIMENTO, 2008), utilizando o mesmo conjunto de funções que Araújo, Cabral e Nascimento (2008). Resultados coletados evidenciam a eficiência dos procedimentos elaborados.

Os métodos C-GGVNS$_1$, C-GGVNS$_2$, EC-GRASP e BC-GRASP, foram analisados quando ao GAP médio, número mínimo de avaliações da função objetivo e análise de convergência para funções com baixa e alta dimensionalidade. Os dados analisaram revelam que através do GAP médio o procedimento C-GGVNS$_2$ obteve aproximadamente 37% de

sucesso empatando em 89 caso de testes, contra aproximadamente 15% de sucesso do procedimento C-GGVNS$_1$, 10% do BC-GRASP e 5% do EC-GRASP.

Ao observar o número mínimo de avaliações da função objetivo, o procedimento C-GGVNS$_2$, também demonstra bom desempenho, tendo 40% de chamadas mínimas a função objetivo, equiparando em aproximadamente 15%. Na análise de convergências para funções com poucas dimensões foi o único procedimento que convergiu em todos os casos de testes, principalmente nas funções Colville, Dixonprice, Schwefel e Trid, em que os outros métodos não obtiveram nenhuma convergência. Já na análise de convergência para funções com alta dimensionalidade, o desempenho do C-GGVNS$_2$, é mantido em 5 casos de teste, sendo superado pelo procedimento BC-GRASP nas funções Rosenbrock e Zankarov, com 500 e 100 dimensões respectivamente.

O procedimento C-GRASP com construção paralela, segunda proposta deste trabalho, foi submetido à testes de comparação com o procedimento C-GRASP padrão, utilizando um conjunto reduzido de casos de testes também trabalhados por Araújo, Cabral e Nascimento (2008), com funções de 20 à 30 dimensões. Resultados revelaram promissora aceleração, embora estas funções tenham um número pequeno de dimensões considerando problemas em outras áreas de aplicação, como a aprendizagem de máquina, por exemplo, o procedimento reduziu em tempo de execução, em média 3s.

Assim, pode-se inferir que os procedimentos propostos neste trabalho favorecem os resultados finais, superando alguns métodos presentes na literatura e apresentando uma nova abordagem promissora para o procedimento clássico C-GRASP.

7.1 Trabalhos Futuros

Como trabalhos futuros, sugere-se a elaboração de testes de comparação entre o procedimento C-GGVNS$_2$, e o recém trabalho desenvolvido por Birgin et al. (2010), que faz uso de um diversificado conjunto de funções de testes e revela bons resultados computacionais em comparação ao procedimento C-GRASP padrão.

Uma segunda sugestão seria a aplicação da versão do C-GRASP com construção paralela a problemas maiores, para a realização de experiências mais abrangentes sobre hardware. E por fim, a paralelização de outras partes do algoritmo C-GRASP para o alcanço pleno do

poder de processamento computacional das GPUs.

Bibliografia

AGUILERA, B.; ROLI, A.; SAMPELS, M. *Hybrid Metaheuristics: An Emerging Approach to Optimization*. [S.l.]: Springer, 2008. (Studies in Computational Intelligence). ISBN 9783540782940.

AHMED, A.-R. H. A. *Studies on Metaheuristics for Continuous Global Optimization Problems*. DissertaÁ„o (Mestrado) — Kyoto University, 2004.

ALBA, E. *Parallel Metaheuristics: A New Class of Algorithms*. [S.l.]: Wiley, 2005. (Wiley Series on Parallel and Distributed Computing). ISBN 9780471678069.

ANDRÉASSON, N.; EVGRAFOV, A.; PATRIKSSON, M. *An introduction to continuous optimization: foundations and fundamental algorithms*. [S.l.]: Professional Publishing Svc., 2007. ISBN 9789144044552.

ARAÚJO, T. M. U. de; CABRAL, L. dos A. F.; NASCIMENTO, R. Q. do. Hibridizando a metaheurística C-GRASP com o método de busca por padrões adaptativos para resolução de problemas de otimização global contínua. *Gestão da Produção, Operações e Sistemas – GEPROS*, v. 3, n. 4, p. 155–167, 2008.

BAZARAA, M. S.; SHERALI, H. D.; SHETTY, C. M. *Nonlinear Programming: Theory and Algorithms*. [S.l.]: Wiley, 2006. ISBN 9780471486008.

BECK, A. On the linear search problem. *Israel Journal of Mathematics*, v. 2, n. 4, p. 221–228, 1964.

BIRGIN, E. G. et al. Continuous GRASP with a local active-set method for bound-constrained global optimization. *Journal of Global Optimization*, v. 48, n. 2, p. 289–310, 2010.

BIRGIN, E. G.; MARTÍNEZ, J. M. Large-Scale Active-Set Box-Constrained Optimization Method with Spectral Projected Gradients. *Computational Optimization and Applications*, v. 23, n. 1, p. 101–125, 2002.

CARRIZOSA, E. et al. Gaussian Variable Neighborhood Search for Continuous Optimization. *Computers & Operations Research*, v. 39, n. 9, p. 2206–2213, 2012.

CHAPRA, S. C.; CANALE, R. P. *Métodos Numéricos para Engenharia*. [S.l.]: McGraw Hill Brasil, 2008. ISBN 9788580550115.

CHAVES, A. A. *Uma Meta-heurística Híbrida com Busca por Agrupamentos Aplicada a Problemas de Otimização Combinatória*. DissertaÁ„o (Mestrado) — Instituto Nacional de Pesquisas Espaciais, 2009.

CORPORATION, N. CUDA C Programming Guide. 2013. Acesso em: 1 jul. 2013. Disponível em: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/>.

DUNLAP, R. A. *The Golden Ratio and Fibonacci Numbers*. [S.l.]: World Scientific, 1997. ISBN 9789810232641.

FEO, T. A.; RESENDE, M. G. C. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, v. 6, n. 2, p. 109–133, 1995.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers and Operation Research*, v. 13, n. 5, p. 533–549, 1986.

GLOVER, F.; MARTI, R. Tabu Search. *Metaheuristic Procedures for Training Neutral Networks*, v. 36, p. 53–69, 2006.

HANSEN, P. et al. Variable Neighborhood Search. *Handbook of Metaheuristics*, v. 146, p. 61–86, 2010.

HEDAR, A.-R.; FUKUSHIMA, M. Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization. *Optimization Methods and Software*, v. 19, n. 3–4, p. 291–308, 2004.

HEDAR, A.-R.; FUKUSHIMA, M. Tabu Search directed by direct search methods for nonlinear global optimization. *European Journal of Operational Research*, v. 170, n. 2, p. 329–349, 2006.

HIRSCH, M. J. et al. Global optimization by continuous grasp. *Optimization Letters*, v. 1, n. 2, p. 201–212, 2007.

HIRSCH, M. J.; PARDALOS, P. M.; RESENDE, M. G. C. Speeding up continuous GRASP. *European Journal of Operational Research*, v. 205, n. 3, p. 507–521, 2010.

HWU CHRISTOPHER RODRIGUES, S. R. W.-M.; STRATTON, J. UCompute Unified Device Architecture Application Suitability. *Computing in Science & Engineering*, v. 11, n. 3, p. 16–26, 2009.

KIRKPATRICK, S. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, v. 34, n. 5–6, p. 975–986, 1984.

LOURENÇO, H.; MARTIN, O.; STÜTZLE, T. Iterated Local Search. *Handbook of Metaheuristics*, v. 57, p. 320–353, 2003.

MLADENOVIć, N. et al. General variable neighborhood search for the continuous optimization. *European Journal of Operational Research*, v. 191, p. 753–770, 2008.

MLADENOVIć, N.; HANSEN, P. Variable Neighborhood Search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100, 1997.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. [S.l.]: Dover Publications, 1998. (Dover books on mathematics). ISBN 9780486402581.

RESENDE, M.; RIBEIRO, C. Greedy Randomized Adaptive Search Procedures. *Handbook of metaheuristics*, v. 57, p. 219–249, 2003.

STÜTZLE, T. *Applying iterated local search to the permutation flow shop problem*. DissertaÁ„o (Mestrado) — University of Technology Darmstadt, 1998.

TALBI, E.-G. *Metaheuristics: From Design to Implementation*. [S.l.]: Wiley, 2009. (Wiley Series on Parallel and Distributed Computing). ISBN 9780470496909.

TONETTO, L. M. et al. O papel das heurísticas no julgamento e na tomada de decisão sob incerteza . *Estudos de Psicologia*, v. 23, n. 2, p. 181–189, 2006.

YANG, X.-S. *Introduction to Mathematical Optimization: From Linear Programming to Metaheuristics*. [S.l.]: Ingram Pub Services, 2008. ISBN 9781904602828.

Apêndice A

Definições das Funções de Teste

(A_n) Função Ackley

Definição: $A_n(x) = -10\exp^{-0,2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}} - \exp^{\frac{1}{n}\sum_{i=1}^{n}cos2\pi x_i} + 20 + \exp$

Domínio: [-15; 30]^n

Mínimo Global: x^* = (0;...;0); $A_n(x^*) = 0$

(BE) Função Beale

Definição: BE(x) = (1,5 - x_1 x_1x_2) + (2,25 - x_1 - $x_1x_2^2$)^2 + (2,65 - x_1 - $x_1x_2^3$)^2

Domínio: $[-4,5; 4,5]^2$

Mínimo Global: x^* = (3; 0,5); BE(x^*) = 0

(B2) Função Bohachevsky

Definição: B2(x) = $x_1^2 + 2x_2^2 - 0,3cos(3\pi x_1) - 0,4cos(4\pi x_2) + 0,7$

Domínio: [-50; 100]^2

Mínimo Global: x^* = (0; 0); B2(x^*) = 0

(BO) Função Booth

Definição: BO(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2

Domínio: [-10; 10]^2

Mínimo global: x^* = (1; 3); BO(x^*) = 0

(BR) Função Branin

Definição: BR(x)= $(x_2 - \frac{5}{4\pi^2}x_1^2 + \frac{5}{\pi}x_16)^2 + 10(1 - \frac{1}{8\pi})cos(x_1) + 10$

Domínio: [-5; 15]^2

Mínimo global (um de vários): $x^* = (\pi; 2, 275)$; $BR(x^*)$ = 0,397887

(CA) Função Six-Hump CamelBack

Definição: CA(x) = $4x_1^2 2, 1x_1^4 + \frac{1}{3}x_1^6 + x_1x_24x_2^2 + 4x_2^4$

Domínio: [-5; 5]^2

Mínimo Global (um de vários): x^* = (0,0894375; -0,71289062); CA(x^*) = -1,03162801

(CV) Função Colville

Definição: CV(x) = $100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10, 1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19, 8(x_2 - 1)(x_4 - 1)$

Domínio: [-10, 10]^4

Mínimo Global: x^*= (1; 1; 1; 1); CV(x^*) = 0

(DP$_n$) Função Dixon and Price

Definição: DP$_n$(x) = $(x_i - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$

Domínio:[-10; 10]^n

Mínimo Global: $x_i^* = 2^{-\frac{2^i-2}{2^i}}$, $\forall i = 1, ..., n$. DP$_n$(x^*) = 0

(EA) Função Easom

Definição: EA(x)= -cos(x_1) * cos(x_2)*$\exp^{-(x_1-\pi)^2-(x_2-\pi)^2}$

Domínio: [-100; 100]^2

Mínimo Global: x^* = (π ;π); EA(x*) = -1

(GP) Função Goldstein and Price

Definição: GP(x)= $[1 + (x_1 + x_2 + 1)^2 * (19 - 14x_1 + 3x_1 - 14x_2 + 6x_1x_2 + 3x_2^2)] * [30 + (2x_1 - 3x_2)^2 * (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$

Domínio: [-2; 2]^2

Mínimo Global: x^* = (0; 1); GP(x^*) = 3

(GR$_n$) Função Griewank

Definição: GR$_n(x)$ = $\sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1)$

Domínio: [-300; 600]^n

Mínimo Global: x^* = (0;...;0); GR$_n(x^*)$ = 0

(Hn,m) Função Hartmann

Definição: H$_{n,m}(x)$ = $-\sum_{i=1}^{m} a_i \exp^{-t(x)}$

Domínio: [0; 1]^n

Mínimo Global (n=3,m=4): x^* = (0,114614; 0,555469; 0,852547); H(x^*) = -3,86278

Mínimo Global (n=6, m=4): x^* = (0,20169; 0,150011; 0,476874;0,2785332; 0,311652;0,657300) ; H(x^*) = - 3,32237

Parâmetros: $\sum_{j=1}^{n} A_{ij}^{(n)}(x_j - P_{ij}^{(n)})^2$

$$A^{(3)} = \begin{vmatrix} 3 & 10 & 30 \\ 0,1 & 10 & 35 \\ 3 & 10 & 30 \\ 0,1 & 10 & 35 \end{vmatrix}$$

$$P^{(3)} = 10^{-4} \begin{vmatrix} 6890 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8838 \end{vmatrix}$$

$$A^{(6)} = \begin{vmatrix} 10 & 3 & 17 & 3,5 & 1,7 & 8 \\ 0,05 & 10 & 17 & 0,1 & 8 & 14 \\ 3 & 3,5 & 1,7 & 10 & 17 & 8 \\ 17 & 8 & 0,05 & 10 & 0,1 & 14 \end{vmatrix}$$

$$P^{(6)} = 10^{-4} \begin{vmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{vmatrix}$$

$$\alpha = \begin{vmatrix} 1; 1, 2; 3; 3, 2 \end{vmatrix}$$

(M) Função Matyas

Definição: M(x) = $0,26(x_1^2 + x_2^2) - 0,48x_1x_2$

Domínio: [-5; 10]^2

Mínimo Global: x^* = (0; 0); M(x^*) = 0

(P$_{n,\beta}$) Função Perm

Definição: P$_{n,\beta}$(x) = $\sum_{k=1}^{n}[\sum_{i=1}^{n}(i^k + \beta)((\frac{x_i}{i})^k - 1)]^2$

Domínio: [-n; n]^n

Mínimo Global: x^* = (1;2;...; n); $P_{n,\beta}(x^*) = 0$

($P^0_{n,\beta}$) Função Perm$_0$

Definição: $P^0_{n,\beta}(x) = \sum_{k=1}^{n}[\sum_{i=1}^{n}(i^k + \beta)((x_i^k - \frac{1}{i^k}))]^2$

Domínio: [-n; n]^n

Mínimo Global: x^* = (1; 1/2;...;1/n); $P^0_{n,\beta} = 0$

(RA$_n$) Função Rastrigin

Definição: = $RA_n(x) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$

Domínio: [-2,56; 5,12]^n

Mínimo Global: x^* = (0;...;0); $RA_n(x^*) = 0$

(Rn) Função Rosenbrock

Definição: $Rn(x) = \sum_{j=1}^{n-1}[100(x_j - x_{j+1})^2 + (x_j - 1)^2]$

Domínio: [-10;10]^n

Mínimo Global: x^* = (1,...,1); $Rn(x^*) = 0$

($S_{4,m}$)Função Shekel

Definição: $S_{4,m}(x) = -\sum_{i=1}^{m}[\sum_{i=1}^{4}(x_i - a_i)^2 + c_i]^{-1}$

Domínio: $0 < x_j < 10, j = 1, ..., 4$

Mínimo Global: $S_{4,5}(x^*) = -10.1532$, $S_{4,7}(x^*) = -10.4029$ e $S_{4,10}(x^*) = -10.5364$

(SC2) Função Schwefel

Definição: $SC2(x) = 418,9829 - \sum_{i=1}^{n} x_1 * sin(\sqrt{|x|})$

Domínio: [-500; 500]^n

Mínimo Global: x^* = (420,9687;âÄ¶; 420,9687); $SC2(x^*) = 0$

(SH) Função Shubert

Definição: SH(x) = $\sum_{i=1}^{5}[i * cos[(i+1)x_1 + i]] \sum_{i=1}^{5}[i * cos[(i+1)x_2 + i]]$

Domínio: [-10;10]^2

Mínimo Global (um de vários): x^* = (5,48242188; 4,84742188); SH(x^*) = -186,7309

(SS$_n$) Função Sum of Squares

Definição: SS$_n(x)$ = $\sum_{i=1}^{n} ix_1^2$

Domínio: [-5; 10]^n

Mínimo Global: x^* = (0;...;0); SS$_n(x^*)$ = 0

(PW_n) Função Powell

Definição: $PW_n(x)$ = $\sum_{i=1}^{n/4}[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4]$

Domínio: [-4; 5]^n

Mínimo Global: x^* = (3; -1; 0; 1; 3;âÄ¶;3;-1;0;1); $PW_n(x^*)$ = 0

(PS$_{n,b}$) Função Power Sum

Definição: PS$_{n,b}$ = $\sum_{i=1}^{n}((\sum_{i=1}^{n} x_i^k) - b_k)^2$

Domínio: [0; n]^n

Mínimo Global: (n=4; b = 8, 18, 44, 114): x^* = (1;2;2;3); PS$_{4,\{8,18,44,114\}}(x^*)$ = 0

(T$_n$) Função Trid

Definição: T$_n$ = $\sum_{i=1} n^(x_i - 1)^2 + \sum_{i=2}^{n} x_i x_{i-1}$

Domínio: [-n^2; n^2]^n

Mínimo Global: x_i^* = $i(n + 1 - i), \forall i = 1, ..., n$; T$_6(x^*)$ = -50 e T$_{10}(x^*)$ = -210

(Zn) Função Zakharov

Definição: Zn(x) = $\sum_{i=1}^{n}[x_i^2 + (\sum_{i=1}^{n} 0,5ix_i)^2 + (\sum_{i=1}^{n} 0,5ix_i)^4]$

Domínio: [-5; 10]^n

Mínimo Global: x^* = (0, 0, ,0); Zn (x^*) = 0

Apêndice B

Experimentos Computacionais

Tabela B.1: Valores do GAP médio para cada meta-heurística em 100 avaliações da função objetivo

FUNÇÃO	EC-GRASP	C-GGVNS$_1$	C-GGVNS$_2$	BC-GRASP
A$_{30}$	0.000000	0.000000	0.000000	0.000000
BE	0.746099	0.670892	**0.624510**	0.836358
B$_2$	0.000000	0.000000	0.000000	0.000000
BO	0.223613	0.434114	**0.207407**	0.313840
BR	0.146441	0.192356	**0.133739**	0.152291
CV	207165000	155081000	150379000	185758000
DP$_{25}$	524800000	538800000	530800000	582000000
EA	0.830506	0.802258	**0.792844**	0.839925
GP	1746636	4964667	13092080	3.955.050
GR$_{10}$	0.690089	0.631601	**0.609629**	0.633906
GR$_{20}$	0.733712	0.569540	0.591661	0.650699
H$_{3,4}$	0.603654	0.632081	0.636530	0.475515
H$_{6,4}$	0.472640	0.349559	**0.347160**	0.689750
CA	0.250465	0.230510	0.243784	0.322086
L$_{30}$	0.000000	0.000000	0.000000	0.000000
M	0.024736	0.022936	**0.016644**	0.017990
P$_{4,\frac{1}{2}}$	42134601	44305830	**41708207**	50848795
P$_{4,10}$	232792848	167714693	**108667099**	239553093
PS$_{4,\{8,18,44,114\}}$	143380000	103590000	125150000	101900000
RA$_{10}$	7058351	7058351	7058351	7058351
RA$_{20}$	14116703	14116703	14116703	14116703
R$_2$	3664430	1379111	1584363	2244664
R$_{10}$	6638420000	5818430000	**5677540000**	7421760000
R$_{20}$	15604166000	12615078000	**11643166800**	14666286400
SC$_2$	0.236765	0.236765	0.236765	0.236765
SC$_6$	322828798	322828798	322828798	322828798
S$_{4,10}$	7912710	6941576	7074907	7884952
S$_{4,5}$	6060460	6183825	6117180	5993865
S$_{4,7}$	6825519	6944616	**6222012**	7242858
SH	39628085	39011350	39766527	43471621
SP$_3$	0.259956	0.227105	**0.204762**	9408000
SP$_{30}$	9408000	9408000	9408000	0.223090
SS$_{10}$	0.000000	0.000000	0.000000	0.000000
SS$_{20}$	0.000000	0.000000	0.000000	0.000000
T$_{10}$	2188850000	2096300000	2124650000	2082400000
T$_6$	182470000	168390000	183160000	198450000
Z$_{10}$	190825000	178044375	182253125	187494375
Z$_{20}$	52351828125	40282109375	50583171875	53230218750

Tabela B.2: Valores do GAP médio para cada meta-heurística em 500 avaliações da função objetivo

FUNÇÃO	EC-GRASP	C-GGVNS$_1$	C-GGVNS$_2$	BC-GRASP
A$_{30}$	0.000000	0.000000	0.000000	0.000000
BE	0.137282	0.081120	**0.073230**	0.123005
B$_2$	0.000000	0.000000	0.000000	0.000000
BO	0.434114	0.223613	0.000000	0.000000
BR	0.192356	0.146441	**0.013758**	0.015197
CV	155081000	207165000	**7169673**	9286172
DP$_{25}$	538800000	524800000	**3176621**	3591029
EA	0.802258	0.830506	**0.671200**	0.737212
GP	4964667	1746636	0.000000	0.000000
GR$_{10}$	0.631601	0.690089	**0.522670**	0.588257
GR$_{20}$	0.569540	0.733712	**0.515277**	0.525237
H$_{3,4}$	0.632081	0.603654	0.086268	0.075930
H$_{6,4}$	0.349559	0.472640	0.065731	0.064411
CA	0.230510	0.250465	**0.013891**	0.014470
L$_{30}$	0.000000	0.000000	0.000000	0.000000
M	0.022936	0.024736	**0.000000**	0.000297
P$_{4,\frac{1}{2}}$	44305830	42134601	**20851798**	30560011
P$_{4,10}$	167714693	232792848	**26672831**	25284691
PS$_{4,\{8,18,44,114\}}$	103590000	143380000	**0.056946**	0.076854
RA$_{10}$	7058351	7058351	6906056	6612244
RA$_{20}$	14116703	14116703	**13768292**	13812310
R$_2$	1379111	3664430	**0.044547**	12993932
R$_{10}$	5818430000	6638420000	**98380070**	107789623
R$_{20}$	12615078000	15604166000	**6694629**	75213761
SC$_2$	0.236765	0.236765	**0.099047**	0.160751
SC$_6$	322828798	322828798	70160094	0.109883
S$_{4,10}$	6941576	7912710	**5127201**	5786798
S$_{4,5}$	6183825	6060460	**3409989**	3494784
S$_{4,7}$	6944616	6825519	**3825885**	4669596
SH	39011350	39628085	5116070	2535695
SP$_3$	0.227105	0.259956	**0.009281**	0.009334
SP$_{30}$	9408000	9408000	**1610838**	1671404
SS$_{10}$	0.000000	0.000000	0.000000	0.000000
SS$_{20}$	0.000000	0.000000	0.000000	0.000000
T$_{10}$	2096300000	2188850000	27645055	24517951
T$_6$	168390000	182470000	0.383722	0.460948
Z$_{10}$	178044375	190825000	**6916100**	6794064
Z$_{20}$	40282109375	52351828125	**17337510**	18981367

Tabela B.3: Valores do GAP médio para cada meta-heurística em 1.000 avaliações da função objetivo

FUNÇÃO	EC-GRASP	C-GGVNS$_1$	C-GGVNS$_2$	BC-GRASP
A$_{30}$	0.000000	0.000000	0.000000	0.000000
BE	0.019395	0.005166	**0.000000**	0.013314
B$_2$	0.000000	0.000000	0.000000	0.000000
BO	0.000000	0.000000	0.000000	0.000000
BR	0.002353	0.003539	**0.002035**	0.002866
CV	3776942	3771868	**3037012**	3370457
DP$_{25}$	2000000	2000000	2000000	2000000
EA	0.733164	0.675151	**0.665034**	0.732863
GP	0.000000	0.000000	0.000000	0.000000
GR$_{10}$	0.557484	0.500723	**0.476955**	0.481070
GR$_{20}$	0.520681	0.478802	0.483091	0.543633
H$_{3,4}$	0.014333	0.023947	0.020862	0.016569
H$_{6,4}$	0.051381	0.041281	**0.040520**	0.054920
CA	0.002353	0.002103	0.002127	0.002085
L$_{30}$	0.000000	0.000000	0.000000	0.000000
M	0.000025	0.000000	0.000000	0.000071
P$_{4,\frac{1}{2}}$	20259213	18408288	19525438	25967993
P$_{4,10}$	15414750	10616921	10972159	12625007
PS$_{4,\{8,18,44,114\}}$	0.010605	0.000000	0.000000	0.009679
RA$_{10}$	6788989	6707011	**6654256**	6612244
RA$_{20}$	13850455	13799024	**13702873**	13812310
R$_2$	0.414270	0.168600	**0.005608**	0.033988
R$_{10}$	95319072	79986949	84873359	93957639
R$_{20}$	9339433	7112896	**5524181**	9822349
SC$_2$	0.103716	0.096275	0.099047	0.109883
SC$_6$	45702462	44718673	**43933310**	50549360
S$_{4,10}$	4061583	3664724	3964238	4176900
S$_{4,5}$	1647358	2358657	1959583	1956378
S$_{4,7}$	2836170	2737229	2745155	3236189
SH	0.997400	0.674902	**0.000000**	0.02387
SP$_3$	0.001868	0.002078	**0.001164**	0.001500
SP$_{30}$	1768583	1626545	**1610838**	1671404
SS$_{10}$	0.000000	0.000000	0.000000	0.000000
SS$_{20}$	0.000000	0.000000	0.000000	0.000000
T$_{10}$	29773316	27814296	**26739475**	24234784
T$_6$	0.366335	0.306414	0.310134	0.390225
Z$_{10}$	7894734	6152205	6626450	6542827
Z$_{20}$	18950954	17142822	**15959161**	18131901

Tabela B.4: Valores do GAP médio para cada meta-heurística em 5.000 avaliações da função objetivo

FUNÇÃO	EC-GRASP	C-GGVNS$_1$	C-GGVNS$_2$	BC-GRASP
A$_{30}$	0.000000	0.000000	0.000000	0.000000
BE	0.000000	0.000000	0.000000	0.000000
B$_2$	0.000000	0.000000	0.000000	0.000000
BO	0.000000	0.000000	0.000000	0.000000
BR	0.000043	0.000063	0.000046	0.000058
CV	1459450	1468322	**0.854151**	1451650
DP$_{25}$	1608824	1990776	1997615	1546112
EA	0.595951	0.303029	**0.173201**	0.645278
GP	0.000000	0.000000	0.000000	0.000000
GR$_{10}$	0.534048	0.473632	**0.441590**	0.485114
GR$_{20}$	0.524030	0.419963	0.430896	0.457076
H$_{3,4}$	0.007732	0.013902	0.010071	0.007732
H$_{6,4}$	0.034589	0.022946	**0.022694**	0.036149
CA	0.000059	0.000059	0.000066	0.000051
L$_{30}$	0.000000	0.000000	0.000000	0.000000
M	0.000000	0.000000	0.000000	0.000000
P$_{4,\frac{1}{2}}$	11922925	10252886	9724440	9445479
P$_{4,10}$	2714317	2855090	**2006040**	2201283
PS$_{4,\{8,18,44,114\}}$	0.000874	0.000000	0.000000	0.000079
RA$_{10}$	0.965611	0.923301	**0.893422**	0.960641
RA$_{20}$	6771338	6422521	**6363135**	6570724
R$_2$	0.016518	0.003944	0.000000	0.000000
R$_{10}$	25486674	30091061	30876273	33187211
R$_{20}$	6621098	4562495	**4508109**	5868297
SC$_2$	0.003346	0.003181	**0.003006**	0.003560
SC$_6$	1407292	1515352	**1350291**	1438677
S$_{4,10}$	0.600741	0.591088	0.842452	0.376190
S$_{4,5}$	0.000000	0.000000	0.151573	0.101049
S$_{4,7}$	0.158220	0.158220	0.158632	0.172281
SH	0.034738	0.033588	0.029656	0.006771
SP$_3$	0.000018	0.403203	0.000000	0.000000
SP$_{30}$	0.426889	0.000018	0.442549	0.467833
SS$_{10}$	0.000000	0.000000	0.000000	0.000000
SS$_{20}$	0.000000	0.000000	0.000000	0.000000
T$_{10}$	18750254	17518017	**16487271**	17109944
T$_6$	0.150970	0.092694	0.099174	0.150379
Z$_{10}$	2030966	1941583	**1821046**	7548140
Z$_{20}$	8984925	7121425	7678912	1894160

Tabela B.5: Valores do GAP médio para cada meta-heurística em 10.000 avaliações da função objetivo

FUNÇÃO	EC-GRASP	C-GGVNS$_1$	C-GGVNS$_2$	BC-GRASP
A$_{30}$	0.000000	0.000000	0.000000	0.000000
BE	0.000000	0.000000	0.000000	0.000000
B$_2$	0.000000	0.000000	0.000000	0.000000
BO	0.000000	0.000000	0.000000	0.000000
BR	0.000002	0.000002	**0.000001**	0.000002
CV	1.378.601	1.381.767	**0.303487**	1.382.070
DP$_{25}$	1.040.517	1.522.768	**0.525593**	1.024.087
EA	0.275329	0.215709	**0.102448**	0.276035
GP	0.000000	0.000000	0.000000	0.000000
GR$_{10}$	0.324283	0.306792	0.384149	0.343919
GR$_{20}$	0.411288	0.314385	0.407685	0.446354
H$_{3,4}$	0.000603	0.011037	0.010071	0.007731
H$_{6,4}$	0.020591	0.009850	0.010937	0.018637
CA	0.000026	0.000024	0.000023	0.000022
L$_{30}$	0.000000	0.000000	0.000000	0.000000
M	0.000000	0.000000	0.000000	0.000000
P$_{4,\frac{1}{2}}$	8.295.181	5.416.448	6.249.207	7.068.782
P$_{4,10}$	2.038.808	5.416.448	**1.346.126**	1.672.064
PS$_{4,\{8,18,44,114\}}$	0.000000	0.000000	0.000000	0.000000
RA$_{10}$	0.351908	0.305676	0.311877	0.351674
RA$_{20}$	6.893.674	6.422.521	**6.363.135**	6.570.724
R$_2$	0.000000	0.000000	0.000000	0.000000
R$_{10}$	11.908.479	13.909.868	14.313.509	12.799.618
R$_{20}$	6.526.975	4.421.749	**4.295.658**	5.649.075
SC$_2$	0.445666	0.001141	0.001172	0.000969
SC$_6$	0.401453	0.437240	0.425011	0.361551
S$_{4,10}$	0.000000	0.000000	0.053607	0.000000
S$_{4,5}$	0.000000	0.000000	0.000000	0.000000
S$_{4,7}$	0.000000	0.000000	0.000000	0.000000
SH	0.001817	0.001740	0.001274	0.000363
SP$_3$	0.000017	0.000017	**0.000000**	0.000000
SP$_{30}$	0.451735	0.403203	0.442549	0.467833
SS$_{10}$	0.000000	0.000000	0.000000	0.000000
SS$_{20}$	0.000000	0.000000	0.000000	0.000000
T$_{10}$	11.340.298	10.822.229	**9.890.543**	10.839.660
T$_6$	0.016508	0.012983	**0.012471**	0.017676
Z$_{10}$	0.558908	0.552678	0.463929	0.454288
Z$_{20}$	2.911.162	4.691.079	3.402.773	3.980.504

Tabela B.6: Valores do GAP médio para cada meta-heurística em 20.000 avaliações da função objetivo

FUNÇÃO	EC-GRASP	C-GGVNS$_1$	C-GGVNS$_2$	BC-GRASP
A$_{30}$	0.000000	0.000000	0.000000	0.000000
BE	0.000000	0.000000	0.000000	0.000000
B$_2$	0.000000	0.000000	0.000000	0.000000
BO	0.000000	0.000000	0.000000	0.000000
BR	0.000000	0.000000	0.000000	0.000000
CV	1.354.242	1.354.108	**0.123029**	1.354.158
DP$_{25}$	0.620726	0.889905	**0.324241**	0.602734
EA	0.080854	0.062411	0.050996	0.030984
GP	0.000000	0.000000	0.000000	0.000000
GR$_{10}$	0.200127	0.214525	0.202438	0.217685
GR$_{20}$	0.385053	0.285321	**0.281368**	0.369151
H$_{3,4}$	0.000000	0.000000	0.000000	0.000000
H$_{6,4}$	0.000251	0.000670	0.000205	0.000455
CA	0.000010	0.000010	0.000008	0.000010
L$_{30}$	0.000000	0.000000	0.000000	0.000000
M	0.000000	0.000000	0.000000	0.000000
P$_{4,\frac{1}{2}}$	5.318.537	3.138.273	3.385.693	4.394.683
P$_{4,10}$	1.119.891	1.442.672	**0.830800**	1.043.059
PS$_{4,\{8,18,44,114\}}$	0.000000	0.000000	0.000000	0.000000
RA$_{10}$	0.194990	0.190812	**0.168710**	1.758.905
RA$_{20}$	1.772.255	1.648.821	1.706.401	0.180839
R$_2$	0.000000	0.000000	0.000000	0.000000
R$_{10}$	5.965.587	5.787.899	5.901.518	3.953.982
R$_{20}$	5.801.232	3.805.485	3.833.739	4.478.860
SC$_2$	0.103520	0.000230	**0.000219**	0.000225
SC$_6$	0.103416	0.102546	**0.098508**	0.110945
S$_{4,10}$	0.000000	0.000000	0.000000	0.000000
S$_{4,5}$	0.000000	0.000000	0.000000	0.000000
S$_{4,7}$	0.000000	0.000000	0.000000	0.000000
SH	0.000000	0.000000	0.000000	0.000000
SP$_3$	0.000016	0.000015	**0.000000**	0.000003
SP$_{30}$	0.212661	0.175789	0.196813	0.212974
SS$_{10}$	0.000000	0.000000	0.000000	0.000000
SS$_{20}$	0.000000	0.000000	0.000000	0.000000
T$_{10}$	4.568.829	3.976.895	**3.496.317**	4.401.884
T$_6$	0.003283	0.000947	0.001799	0.002469
Z$_{10}$	0.109765	0.096022	**0.074986**	0.082377
Z$_{20}$	1.510.860	1.040.402	1.276.356	1.599.478

Tabela B.7: Valores do GAP médio para cada meta-heurística em 50.000 avaliações da função objetivo

FUNÇÃO	EC-GRASP	C-GGVNS$_1$	C-GGVNS$_2$	BC-GRASP
A$_{30}$	0.000000	0.000000	0.000000	0.000000
BE	0.000000	0.000000	0.000000	0.000000
B$_2$	0.000000	0.000000	0.000000	0.000000
BO	0.000000	0.000000	0.000000	0.000000
BR	0.000000	0.000000	0.000000	0.000000
CV	1.338.891	1.338.329	**0.013697**	1.337.401
DP$_{25}$	0.521469	0.561457	**0.187062**	0.521552
EA	0.000328	0.000241	0.000198	0.000060
GP	0.000000	0.000000	0.000000	0.000000
GR$_{10}$	0.104673	0.103498	**0.094774**	0.100986
GR$_{20}$	0.329504	0.257297	0.260142	0.314625
H$_{3,4}$	0.000000	0.000000	0.000000	0.000000
H$_{6,4}$	0.012294	0.009411	0.010229	0.009407
CA	0.000004	0.000003	**0.000002**	0.000003
L$_{30}$	0.000000	0.000000	0.000000	0.000000
M	0.000000	0.000000	0.000000	0.000000
P$_{4,\frac{1}{2}}$	2.699.304	1.504.866	1.178.231	2.363.507
P$_{4,10}$	0.329081	0.721246	0.399620	0.343505
PS$_{4,\{8,18,44,114\}}$	0.000000	0.000000	0.000000	0.000000
RA$_{10}$	0.033350	0.028803	**0.025446**	0.025928
RA$_{20}$	0.695536	0.582567	0.595716	0.684198
R$_2$	0.000000	0.000000	0.000000	0.000000
R$_{10}$	1.625.182	1.975.194	2.315.903	1.038.528
R$_{20}$	5.663.893	3.663.862	**3.395.230**	4.050.967
SC$_2$	0.030348	0.000184	**0.000163**	0.000175
SC$_6$	0.029013	0.028207	0.030606	0.032092
S$_{4,10}$	0.000000	0.000000	0.000000	0.000000
S$_{4,5}$	0.000000	0.000000	0.000000	0.000000
S$_{4,7}$	0.000000	0.000000	0.000000	0.000000
SH	0.000000	0.000000	0.000000	0.000000
SP$_3$	0.000004	0.000012	**0.000000**	0.000002
SP$_{30}$	0.028686	0.027157	0.027566	0.028347
SS$_{10}$	0.000000	0.000000	0.000000	0.000000
SS$_{20}$	0.000000	0.000000	0.000000	0.000000
T$_{10}$	0.860884	0.739384	**0.671286**	0.772827
T$_6$	0.000251	0.000054	0.000068	0.000287
Z$_{10}$	0.009795	0.007044	**0.005507**	0.007773
Z$_{20}$	0.539558	0.227822	0.322160	0.869187

Tabela B.8: Comparação dos métodos BC-GRASP e C-GGVNS$_2$, quando convergem em 50,100, 500 e 1.000 iterações

	BC-GRASP				**C-GGVNS$_2$**			
Funções	**50**	**100**	**500**	**1000**	**50**	**100**	**500**	**1000**
A$_{30}$	100%	100%	100%	100%	100%	100%	100%	100%
BE	100%	100%	100%	100%	100%	100%	100%	100%
B$_2$	100%	100%	100%	100%	100%	100%	100%	100%
BO	100%	100%	100%	100%	100%	100%	100%	100%
BR	100%	100%	100%	100%	100%	100%	100%	100%
CV	0%	0%	0%	0%	100%	100%	99,60%	99,90%
DP$_{25}$	0%	0%	0%	0%	100%	98%	97%	96,80%
EA	98%	100%	99,80%	99,60%	98%	94%	94,80%	96,20%
GP	100%	100%	100%	100%	100%	100%	100%	100%
GR$_{10}$	92%	93%	89,20%	91,70%	100%	100%	100%	100%
GR$_{20}$	88%	83%	85%	84,10%	100%	99%	99,20%	98,30%
H$_{3,4}$	100%	100%	100%	100%	100%	100%	100%	100%
H$_{6,4}$	10%	4%	6,80%	5,60%	14%	6%	5,20%	4,30%
CA	62%	55%	61,80%	64,20%	68%	64%	66,80%	69,50%
L$_{30}$	100%	100%	100%	100%	100%	100%	100%	100%
M	100%	100%	100%	100%	100%	100%	100%	100%
P$_{4,\frac{1}{2}}$	4%	3%	2,40%	2,40%	88%	77%	83,40%	75,30%
P$_{4,10}$	70%	65%	68,40%	66,50%	98%	95%	94,80%	93,60%
PS$_{4,\{8,18,44,114\}}$	100%	100%	100%	100%	100%	100%	100%	100%
R$_2$	100%	100%	100%	100%	100%	100%	100%	100%
R$_{10}$	100%	100%	99,80%	99,90%	100%	100%	100%	100%
R$_{20}$	58%	66%	60,40%	60,80%	86%	69%	74,60%	71,60%
SC$_6$	0%	0%	0%	0%	38%	24%	17,20%	19,20%
SC$_2$	0%	0%	0%	0%	30%	59%	26,60%	33,40%
S$_{4,5}$	98%	100%	100%	100%	100%	100%	100%	100%
SH	100%	100%	100%	100%	100%	100%	100%	100%
SP$_3$	100%	100%	100%	100%	100%	100%	100%	100%
SP$_{30}$	100%	100%	100%	100%	100%	100%	100%	100%
SS$_{10}$	100%	100%	100%	100%	100%	100%	100%	100%
SS$_{20}$	100%	100%	100%	100%	100%	100%	100%	100%
T$_6$	0%	0%	0%	0%	96%	76%	72,40%	66,60%
T$_{10}$	0%	0%	0%	0%	48%	14%	25%	21,20%
Z$_{20}$	100%	100%	100%	100%	100%	100%	31%	100%
Z$_{10}$	100%	100%	100%	100%	100%	100%	50%	50,20%

Tabela B.9: Comparação dos métodos C-GGVNS$_1$ e EC-GRASP, quando convergem em 50,100, 500 e 1.000 iterações

	C-GGVNS$_1$				**EC-GRASP**			
Funções	**50**	**100**	**500**	**1000**	**50**	**100**	**500**	**1000**
A$_{30}$	100%	100%	100%	100%	100%	100%	100%	100%
BE	100%	100%	100%	100%	100%	100%	100%	100%
B$_2$	100%	100%	100%	100%	100%	100%	100%	100%
BO	100%	100%	100%	100%	100%	100%	100%	100%
BR	100%	100%	100%	100%	100%	100%	100%	100%
CV	0%	0%	0%	0%	0%	0%	0%	0%
DP$_{25}$	0%	0%	0%	0%	0%	0%	0%	0%
EA	74%	63%	72%	70,40%	58%	75%	72,60%	76,10%
GP	100%	100%	100%	100%	100%	100%	100%	100%
GR$_{10}$	92%	95%	91%	90,50%	90%	89%	90,60%	91,20%
GR$_{20}$	88%	83%	85,60%	86,20%	82%	83%	84%	86,10%
H$_{3,4}$	100%	100%	100%	100%	100%	100%	100%	100%
H$_{6,4}$	4%	8%	6,60%	5,20%	2%	8%	6,60%	6%
CA	56%	100%	63%	60,30%	58%	63%	58,20%	60,60%
L$_{30}$	100%	100%	100%	100%	100%	100%	100%	100%
M	100%	100%	100%	100%	100%	100%	100%	100%
P$_{4,\frac{1}{2}}$	2%	2%	2,80%	2,70%	4%	1%	2,60%	1,10%
P$_{4,10}$	70%	64%	68%	64,90%	72%	71%	70%	0%
PS$_{4,\{8,18,44,114\}}$	100%	100%	100%	100%	100%	100%	100%	100%
R$_2$	100%	100%	100%	100%	100%	100%	100%	100%
R$_{10}$	100%	100%	100%	99,70%	100%	100%	100%	100%
R$_{20}$	46%	49%	49,80%	52,90%	58%	52%	50,60%	50,40%
SC$_6$	0%	0%	0%	0%	0%	0%	0%	0%
SC$_2$	0%	0%	0%	0%	0%	0%	0%	0%
S$_{4,5}$	100%	100%	100%	100%	100%	100%	100%	100%
SH	100%	100%	100%	100%	100%	100%	100%	100%
SP$_3$	2%	1%	0,20%	0,60%	2%	1%	1,40%	0%
SP$_{30}$	0%	0%	0%	0%	0%	0%	0%	0%
SS$_{10}$	100%	100%	100%	100%	100%	100%	100%	100%
SS$_{20}$	100%	100%	100%	100%	100%	100%	100%	100%
T$_6$	0%	0%	0%	0%	0%	0%	0%	0%
T$_{10}$	0%	0%	0%	0%	0%	0%	0%	0%
Z$_{20}$	100%	100%	100%	100%	100%	100%	100%	100%
Z$_{10}$	100%	100%	100%	100%	100%	100%	100%	100%

Tabela B.10: Análise da convergência do procedimento C-GGVNS$_2$, em funções com alta dimensionalidade

Iterações	**R$_{20}$ (%)**	**R$_{50}$ (%)**	**R$_{100}$ (%)**	**R$_{500}$ (%)**	**Z$_{20}$ (%)**	**Z$_{50}$ (%)**	**Z$_{100}$ (%)**
10	**10**	11,25	12,5	12,5	12,5	12,5	12,5
20	**18,75**	25	25	25	25	25	25
30	**28,75**	37,5	37,5	37,5	37,5	37,5	33,75
40	**38,75**	50	50	50	50	50	45
50	**52,5**	62,5	62,5	55	62,5	62,5	58,75
60	**52,5**	75	75	62,5	75	75	66,25
70	**60**	87,5	87,5	82,5	87,5	87,5	75
80	**70**	100	100	91,25	100	100	83,75

Tabela B.11: Análise da convergência do procedimento BC-GRASP, em funções com alta dimensionalidade

Iterações	R$_{20}$ (%)	R$_{50}$ (%)	R$_{100}$ (%)	R$_{500}$ (%)	Z$_{20}$ (%)	Z$_{50}$ (%)	Z$_{100}$ (%)
10	7,5	**12,5**	12,5	12,5	12,5	12,5	12,5
20	15	25	25	25	25	25	25
30	21,25	37,5	37,5	37,5	37,5	37,5	**37,5**
40	30	50	50	50	50	50	**50**
50	40	62,5	62,5	62,5	62,5	62,5	**62,5**
60	41,25	75	75	**75**	75	75	75
70	48,75	87,5	87,5	**87,5**	87,5	87,5	**87,5**
80	65	100	100	**95**	100	100	**100**