

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

MÉTODOS HÍBRIDOS BASEADOS EM
CONTINUOUS-GRASP APLICADOS À
OTIMIZAÇÃO GLOBAL CONTÍNUA

TIAGO MARITAN UGULINO DE ARAÚJO

JOÃO PESSOA-PB
Fevereiro-2009

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**MÉTODOS HÍBRIDOS BASEADOS EM
CONTINUOUS-GRASP APLICADOS À OTIMIZAÇÃO
GLOBAL CONTÍNUA**

TIAGO MARITAN UGULINO DE ARAÚJO

JOÃO PESSOA-PB
Fevereiro-2009

TIAGO MARITAN UGULINO DE ARAÚJO

**MÉTODOS HÍBRIDOS BASEADOS EM
CONTINUOUS-GRASP APLICADOS À OTIMIZAÇÃO
GLOBAL CONTÍNUA**

DISSERTAÇÃO APRESENTADA AO CENTRO DE CIÊNCIAS EXATAS E
DA NATUREZA DA UNIVERSIDADE FEDERAL DA PARAÍBA, COMO
REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE EM
INFORMÁTICA (SISTEMAS DE COMPUTAÇÃO).

Orientador: Prof. Dr. Lucídio dos Anjos Formiga Cabral

JOÃO PESSOA-PB
Fevereiro-2009

A663m Araújo, Tiago Maritan Ugulino de.
Métodos híbridos baseados em continuos-Grasp aplicados
à otimização global contínua / Tiago Maritan Ugulino de
Araújo.- João Pessoa, 2009.
101f. : il.
Orientador: Lucídio dos Anjos Formiga Cabral
Dissertação (Mestrado) – UFPB/CCEN
1. Informática. 2. Otimização Global Contínua. 3.
Metaheurística. 4. C-GRASP.

UFPB/BC

CDU: 004(043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado do Tiago Maritan Ugulino de Araújo, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 06 de fevereiro de 2009.

Aos seis dias do mês de fevereiro do ano dois mil e nove, às quatorze horas, na Sala de Reuniões do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Processamento de Sinais e Sistemas Gráficos", o Sr. Tiago Maritan Ugulino de Araújo. A comissão examinadora foi composta pelos professores doutores: Lucídio dos Anjos Formiga Cabral (DI-UFPB), Orientador e Presidente da Banca Examinadora, Roberto Quirino do Nascimento (DE-UFPB), Leonardo Vidal Batista (DI-UFPB), como examinadores internos e Cláudio Nogueira de Menezes (UFG), com examinador externo. Dando início aos trabalhos, o Prof. Lucídio dos Anjos Formiga Cabral, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "MÉTODOS HÍBRIDOS BASEADOS EM CONTINUOUS-GRASP APLICADOS À OTIMIZAÇÃO GLOBAL CONTÍNUA". Concluída a exposição, o candidato foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "Aprovado". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Valéria Gonçalves Soares, coordenadora do Programa, servindo de secretária, lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 06 de fevereiro de 2009.

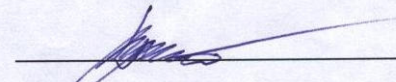
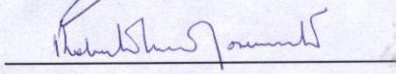
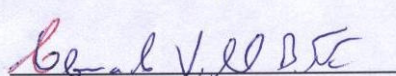
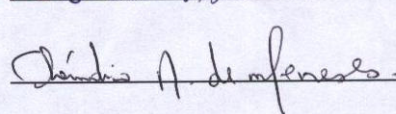
Valéria Gonçalves Soares
Valéria Gonçalves Soares

Prof. Dr. Lucídio dos Anjos Formiga Cabral
Orientador (DI-UFPB)

Prof. Dr. Roberto Quirino do Nascimento
Examinador Interno (DE-UFPB)

Prof. Dr. Leonardo Vidal Batista
Examinador Interno (DI-UFPB)

Prof. Dr. Cláudio Nogueira de Menezes
Examinador Externo (UFG)

Dedico esse trabalho a meus pais Mário e Regiane que me ensinaram desde meus primeiros anos de vida a importância de estudar para ter um objetivo de vida, que nunca mediram esforços para apoiar meus estudos, além do grande incentivo, amor e atenção que recebo deles todos os dias. A eles devo tudo o que sou e o que aprendi.

AGRADECIMENTOS

A Deus pelo dom da vida, pelas pessoas maravilhosas que ele colocou em meu caminho durante esse projeto e pelas inúmeras bênçãos que recebo todos os dias.

A meus pais, Mário e Regiane, meus exemplos de vida, que amo incondicionalmente, que sempre me ensinaram a importância de estudar para ter um objetivo de vida, que sempre me amaram e me apoiaram em todas as minhas decisões. Espero um dia ser pelo menos um pouco do que eles são.

A minha irmã Natália por seu grande carinho, apoio, incentivo e pela sua compreensão nos meus momentos de maior estresse.

A minha namorada Priscila, uma pessoa especial que Deus colocou ao meu lado. Uma mulher carinhosa, guerreira, determinada e destemida que sempre me incentivou a lutar pelos meus sonhos. Agradeço por estar sempre ao meu lado, me dando todo carinho, apoio e por me compreender nos momentos que estive ausente trabalhando nesse projeto.

A meu orientador professor Lucídio Cabral, exemplo de profissional, de pessoa, de educador. Uma pessoa de índole e caráter admiráveis, que sempre acreditou e confiou no meu potencial, mesmo nos momentos que não mereci. Uma daquelas pessoas que pretendo ter como amigo para o resto da vida.

Ao professor Guido Lemos, um profissional e uma pessoa diferenciada. Um profissional destemido, empreendedor, que não mede esforços para ver seus alunos crescerem na vida. Agradeço muito pelas oportunidades e pelos ensinamentos. Ele é sem dúvida, um orgulho para todos que trabalham com ele e junto com meus pais, o meu espelho profissional.

Ao professor Roberto Quirino pela sugestão do tema desse trabalho, pelas valiosas sugestões e pelo apoio na escrita dessa dissertação.

Aos amigos do LAVID e todos que ajudaram direta e indiretamente na conclusão desse trabalho.

RESUMO

Recentemente tem crescido na literatura o interesse em resolver problemas de otimização global contínua utilizando metaheurísticas. Algumas metaheurísticas originalmente propostas para resolução de problemas de otimização combinatória, dentre elas, *Greedy Randomized Adaptive Search Procedure* (GRASP) [1], Busca Tabu [2,3] e *Simulated Annealing* [4], têm sido adaptadas para resolver problemas de otimização global contínua. A metaheurística *Continuous-GRASP* (C-GRASP) [5-7] pertence a esse grupo de metaheurísticas. Proposta por Hirsch *et al.*, o C-GRASP, é até então, a primeira e única adaptação da metaheurística GRASP para resolução de problemas de otimização global contínua.

No entanto, por utilizar construções aleatórias, o C-GRASP e muitas outras metaheurísticas, em geral, apresentam uma convergência lenta. Com o objetivo de acelerar essa convergência, nesse trabalho propomos a criação de métodos híbridos baseados em C-GRASP para resolução de problemas de otimização global contínua. Os métodos propostos utilizam a metaheurística C-GRASP como base combinados com etapas de refinamento compostas por métodos de descidas.

Para validar os métodos propostos, esses métodos foram comparados com o C-GRASP e outras metaheurísticas da literatura para um conjunto de problemas testes da literatura com mínimo global conhecido. Os resultados computacionais atestam a eficiência e robustez dos métodos propostos, bem como, sua capacidade de acelerar a convergência do C-GRASP.

ABSTRACT

Recently, the interest in solving global continuous optimization problems by using metaheuristics has grown in the literature. Many of these metaheuristics, such as Greedy Randomized Adaptive Search Procedure (GRASP) [1], Tabu Search [2,3] and Simulated Annealing [4], were originally proposed for combinatorial problems and have been adapted to deal with continuous optimization problems. The metaheuristic Continuous-GRASP (C-GRASP) [5-7] can be included in this group of metaheuristics. C-GRASP was developed by Hirsch *et al.* and it is the first adaptation of GRASP for the continuous domain.

However, C-GRASP may suffer from slow convergence due to their random constructions. This work proposes the creation of hybrid methods based on C-GRASP for solving global continuous optimization problems. The proposed methods combine C-GRASP with local search methods.

We compare the proposed methods with C-GRASP and other metaheuristics from recent literature on a set of standard test problems whose global minima are known. The computational results show the efficiency and robustness of the methods, as well as their ability to accelerate the convergence of C-GRASP.

SUMÁRIO

LISTA DE FIGURAS	12
LISTA DE TABELAS	13
SIGLAS E ABREVIATURAS	14
1. INTRODUÇÃO.....	14
1.1 Objetivos do trabalho.....	16
1.2 Organização da dissertação	16
2. OTIMIZAÇÃO GLOBAL CONTÍNUA	18
2.1 Definição do problema	18
2.2 Otimização Sem Restrições.....	19
2.2.1 Busca Linear	19
2.2.1.1 Intervalo de incerteza.....	20
2.2.1.2 Método da seção áurea.....	20
2.2.1.3 Método de Fibonacci	22
2.2.2 Busca Multidimensional	23
2.2.2.1 Métodos de Busca Direcionada (<i>Direct Search Methods (DSM)</i>)	23
2.2.2.2 Métodos de Busca Multidimensional com Derivadas	27
3. METAHEURÍSTICAS	35
3.1. Metaheurísticas para otimização combinatória	36
3.1.1 <i>Greedy Randomized Adaptive Search Procedure</i> (GRASP)	36
3.1.2 Busca Tabu.....	38
3.1.3 Simulated Annealing	40
3.1.4 Algoritmos Genéticos (AG).....	42
3.1.4.1 Seleção	43
3.1.4.1 Cruzamento e mutação.....	44
3.2. Metaheurísticas para otimização contínua.....	44
3.2.1 <i>Continuous</i> -GRASP (C-GRASP).....	45
3.2.1.1 Procedimento de Construção.....	47
3.2.1.2 Procedimento de Busca Local	47
3.2.2 <i>Directed Tabu Search</i> (DTS)	50
3.2.2.1 Elementos de memória.....	51
3.2.2.2 Procedimento de Exploração.....	52
3.2.2.3 Procedimento de Diversificação.....	53
3.2.2.4 Procedimento de Intensificação.....	53
3.2.3 <i>Particle Swarm Optimization</i> (PSO)	53
4. MÉTODOS PROPOSTOS	56
4.1 Método EC-GRASP	56
4.1.1 <i>Adaptive Pattern Search</i> (APS)	58
4.1.2 Procedimento de Busca Local.....	60
4.2 Método BC-GRASP	62

5. RESULTADOS E DISCUSSÕES	65
5.1 Ambiente de Testes.....	65
5.2 Comparando EC-GRASP com C-GRASP	67
5.3 Comparando EC-GRASP com outras metaheurísticas	70
5.4 Comparando EC-GRASP com BC-GRASP.....	73
6. CONSIDERAÇÕES FINAIS.....	77
7. REFERÊNCIAS BIBLIOGRÁFICAS	79
ANEXO I - DEFINIÇÃO DOS PROBLEMAS TESTES	83
ANEXO II – ARTIGO PUBLICADO	89

LISTA DE FIGURAS

Figura 2.1. Redução do intervalo de incerteza.	20
Figura 2.2. Redução do intervalo de incerteza pelo método da seção áurea.	21
Figura 2.3. Pseudocódigo do método da seção áurea.	21
Figura 2.4. Pseudocódigo do procedimento Nelder-Mead.	25
Figura 2.5. Simplex de duas dimensões.	25
Figura 2.6. Pseudocódigo do procedimento GPS.	26
Figura 2.7. Pseudocódigo do método do gradiente.	28
Figura 2.8. Ilustração geométrica do método de máxima descida.	29
Figura 2.9. Ilustração geométrica do método de Newton.	31
Figura 2.10. Pseudocódigo de um método quase-Newton.	32
Figura 2.11. Ilustração do método DFP.	33
Figura 3.1. Pseudocódigo do GRASP.	37
Figura 3.2. Pseudocódigo do procedimento de construção do GRASP.	37
Figura 3.3. Pseudocódigo do procedimento de busca local do GRASP.	38
Figura 3.4. Pseudocódigo do procedimento Busca Tabu.	39
Figura 3.5. Pseudocódigo do procedimento <i>Simulated Annealing</i>	41
Figura 3.6. Pseudocódigo do procedimento Algoritmo Genético.	42
Figura 3.7. Exemplo de uma roleta para uma população de cinco cromossomos.	43
Figura 3.8. Exemplo gráfico de um procedimento de cruzamento.	44
Figura 3.9. Pseudocódigo do C-GRASP.	46
Figura 3.10. Pseudocódigo do procedimento de construção do C-GRASP.	48
Figura 3.11. Pseudocódigo do procedimento de busca local do C-GRASP.	49
Figura 3.12. Busca Local do C-GRASP.	49
Figura 3.13. Estrutura do método DTS.	51
Figura 3.14. Pseudocódigo do procedimento de exploração.	52
Figura 3.15. Pseudocódigo do procedimento de Diversificação.	53
Figura 3.16. Pseudocódigo do PSO.	55
Figura 4.1. Pseudocódigo do EC-GRASP.	57
Figura 4.2. Pseudocódigo do procedimento APS.	59
Figura 4.3. Estratégia APS em duas dimensões.	60
Figura 4.4. Pseudocódigo do procedimento BuscaLocal_APS.	61
Figura 4.5. Pseudocódigo do procedimento BC-GRASP.	63
Figura 5.1. Representação geométrica de um conjunto de funções de teste para $n=2$	67
Figura 5.2. Representação geométrica de um conjunto de funções de teste para $n=2$	68

LISTA DE TABELAS

Tabela 2.1. Resumo da execução do método da descida de máximo declive.....	28
Tabela 2.2. Resumo da execução do método de Newton.....	31
Tabela 5.1. 42 funções (ou problemas) testes.....	66
Tabela 5.2. Valor dos parâmetros h_s e h_e do EC-GRASP.	69
Tabela 5.3. Comparação entre EC-GRASP e as metaheurísticas C-GRASP.....	70
Tabela 5.4. Valores do GAP médio para os 40 problemas testes.	71
Tabela 5.5. Valores de GAP médio para cada um dos problemas testes.	72
Tabela 5.6. Valor dos parâmetros do EC-GRASP para cada um dos problemas testes.	73
Tabela 5.7. Comparação entre os métodos EC-GRASP e BC-GRASP.....	74
Tabela 5.8. Valor dos parâmetros do EC-GRASP e BC-GRASP.	75
Tabela 5.9. Comparação entre os métodos propostos com alta dimensionalidade.....	76

SIGLAS E ABREVIATURAS

ADD	<i>Approximate Descent Direction</i>
AG	Algoritmos Genéticos
APS	<i>Adaptive Pattern Search</i>
BC-GRASP	BFGS <i>Continuous</i> -GRASP
BFGS	Broyden-Fletcher-Goldfarb-Shanno
C-GRASP	<i>Continuous Greedy Randomized Adaptive Search Procedure</i>
DFP	Davidon-Fletcher-Powell
DSM	<i>Directed Search Methods</i>
DTS	<i>Directed Tabu Search</i>
EC-GRASP	<i>Enhanced Continuous</i> -GRASP
GENOCOP	<i>Genetic Algorithm for Numerical Optimization of Constrained Problems</i>
GPS	<i>Generalized Pattern Search</i>
GRASP	<i>Greedy Randomized Adaptive Search Procedure</i>
LBFGS	<i>Limited Memory BFGS Method</i>
LCR	Lista de Candidatos Restrita
NMM	<i>Nelder-Mead Method</i>
POGC	Problema de Otimização Global Contínua
PSM	<i>Pattern Search Method</i>
PSO	<i>Particle Swarm Optimization</i>
SDM	<i>Steepest Descent Method</i>
SS	<i>Scatter Search</i>
TS	<i>Tabu Search</i>
TL	<i>Tabu List</i>
TR	<i>Tabu Region</i>
VRL	<i>Visited Region List</i>

1. INTRODUÇÃO

Engenheiros, pesquisadores, gerentes e analistas de pesquisa operacional são confrontados diariamente por problemas de alto grau de complexidade. Esses problemas podem envolver a criação de um projeto ótimo, a alocação de recursos escassos, o planejamento de operações industriais, dentre outros. No passado, esses projetos podiam ser resolvidos com uma margem flexível de aceitação. Um projeto de engenharia, por exemplo, admitia, em geral, uma ampla margem de segurança.

Nos dias de hoje, contudo, com os elevados níveis de competitividade e o acelerado avanço tecnológico não é mais conveniente desenvolver apenas um projeto aceitável. Torna-se necessária a obtenção de soluções melhores, mais eficientes e que utilizem menos recursos. Aliado a isso, o rápido crescimento no tamanho e na complexidade dos problemas reforçam a necessidade de utilização de técnicas que otimizem o processo de obtenção de soluções de boa qualidade.

Em resposta a esses desafios, um grande e rápido crescimento das técnicas e dos modelos de otimização tem se desenvolvido ao longo dos últimos anos. Esse desenvolvimento, aliado aos fatores acima mencionados, tem tornado a otimização mais presente nas engenharias, na economia, nas ciências naturais, entre outras.

Os problemas de otimização têm por objetivo encontrar uma combinação de valores das variáveis do problema que otimizem, ou seja, minimizem (ou maximizem) uma função objetivo, sujeito a um conjunto de restrições nos possíveis valores das variáveis. A função objetivo, nesse contexto, pode representar um risco que precisa ser minimizado, uma performance que precisa ser maximizada.

Um problema de otimização pode ser discreto ou contínuo; global ou local. A classificação discreta ou contínua está relacionada à natureza do espaço de soluções do problema, ou seja, se o espaço de soluções é discreto ou contínuo. A classificação global ou local, por outro lado, está relacionado à existência de mais de um mínimo local nesse espaço de soluções. Mais especificamente, um problema de otimização global pode conter mais de um mínimo local, enquanto que um problema de otimização local contém apenas um mínimo local. Esse trabalho tem como proposta geral explorar a otimização global contínua.

Segundo Hedar [8], os métodos clássicos de otimização têm dificuldades em lidar com problemas de otimização global contínua. Os principais motivos seriam a facilidade de ficarem presos em mínimos locais e a dificuldade em diversificar a busca pelo espaço de soluções. Em outras palavras, a dificuldade em explorar as informações globais necessárias para encontrar um mínimo global quando a função possui vários mínimos locais. Além disso, alguns métodos utilizam cálculos de derivadas de primeira e segunda ordem no procedimento de busca, fato este que exige um elevado esforço computacional para funções objetivo complexas.

Para contornar esse problema, tem crescido recentemente na literatura o número de trabalhos [5-10] envolvendo a utilização de metaheurísticas para resolver esse tipo de problema. As metaheurísticas podem ser definidas como um conjunto de procedimentos de caráter geral, com capacidade de guiar o procedimento de busca, tornando-o capaz de escapar de ótimos locais. Elas têm como objetivo, encontrar uma solução tão próxima quanto possível de uma solução ótima do problema com baixo esforço computacional.

Em geral, as metaheurísticas são utilizadas na resolução de problemas de otimização discreta de forma aproximada. Esses problemas, também conhecidos como problemas NP-difíceis, podem ser definidos como um conjunto de problemas para os quais ainda não existe um algoritmo que os resolvam de forma exata em tempo polinomial [11].

Para esses problemas, o uso de métodos exatos é bastante restrito [11], uma vez que o esforço computacional para encontrar uma solução ótima pode ser consideravelmente alto. No entanto, na prática, em muitos casos, é suficiente encontrar uma solução próxima do ótimo global.

Recentemente, algumas dessas metaheurísticas, dentre elas, *Greedy Randomized Adaptive Search Procedure* (GRASP) [1], Busca Tabu (*Tabu Search*-TS) [2,3] e *Simulated Annealing* [4], têm sido adaptadas para resolver problemas de otimização global contínua. A metaheurística *Continuous-GRASP* (C-GRASP) [5-7] pertence a esse grupo de metaheurísticas. Proposta por Hirsch *et al.* em [5], o C-GRASP é a primeira, e até então, única, adaptação da metaheurística GRASP para resolução de problemas de otimização global contínua. Posteriormente, Hirsch *et al.* [6,7] desenvolveram um novo trabalho onde algumas melhorias no C-GRASP foram propostas.

De forma similar a outras metaheurísticas da literatura, o C-GRASP possui como característica principal a não utilização de cálculos de derivadas de primeira e

segunda ordem no procedimento de busca, reduzindo o esforço computacional necessário na busca global. Essa característica, aliada a capacidade de escapar de mínimos locais, torna o C-GRASP um procedimento de busca eficiente em problemas de otimização global contínua.

No entanto, um dos problemas dessa abordagem, segundo Hedar [8], é que as metaheurísticas, em geral, possuem uma convergência lenta, ou seja, elas se aproximam lentamente de uma solução ótima do problema. Uma das principais razões para essa convergência lenta, segundo Hedar [8], seria o fato das metaheurísticas, em geral, explorarem o espaço global de soluções utilizando movimentos aleatórios, o que dificultaria a detecção de direções de busca promissoras, especialmente na vizinhança de um mínimo local.

Com o objetivo de contornar esse problema, neste trabalho propõe-se a criação de métodos híbridos baseados em *Continuous-GRASP* (C-GRASP) para resolução de problemas de otimização global contínua. Os métodos propostos procuram explorar a eficiência do C-GRASP como procedimento de busca global combinados com métodos de busca local para acelerar sua convergência.

1.1 Objetivos do trabalho

Tendo em vista os aspectos apresentados, o objetivo principal deste trabalho consiste no desenvolvimento de métodos híbridos baseados em *Continuous-GRASP* (C-GRASP) para resolução de problemas de otimização global contínua. Em particular, os métodos propostos utilizam o C-GRASP como base, combinados com etapas de refinamentos compostas por métodos de busca local.

Essa combinação objetiva explorar as boas características do C-GRASP, como a eficiência computacional e a capacidade de diversificação no processo de busca combinadas com a rápida convergência dos métodos de busca local, como os métodos de busca local direcionada e os métodos de busca baseados em derivadas.

1.2 Organização da dissertação

Essa dissertação está estruturada em sete capítulos. Este primeiro capítulo apresenta uma motivação inicial sobre o trabalho, apresentando conceitos relevantes como otimização, otimização global contínua, metaheurísticas e sua aplicação em problemas de otimização. Além disso, são descritos os objetivos do presente trabalho. O segundo capítulo apresenta uma fundamentação teórica detalhada sobre otimização global contínua. Em particular, alguns métodos clássicos de resolução de problemas de otimização contínua são apresentados.

O terceiro capítulo apresenta uma fundamentação teórica sobre metaheurísticas. Importantes metaheurísticas utilizadas para resolução de problemas de otimização combinatória e otimização contínua são apresentadas. Em especial, a metaheurística *Continuous-GRASP* (C-GRASP), metaheurística foco deste trabalho, é apresentada de forma detalhada. No quarto capítulo é apresentada uma descrição detalhada sobre os métodos propostos neste trabalho.

O quinto capítulo descreve os experimentos computacionais utilizados para validar o método proposto e seus resultados computacionais. Esses resultados são confrontados com os dos principais trabalhos da literatura, com o objetivo de discutir a eficiência e robustez do método proposto. No sexto capítulo são apresentadas as considerações finais desse trabalho e descritas algumas propostas de trabalhos futuros. Por fim, no sétimo capítulo são apresentadas as referências bibliográficas utilizadas na composição desse trabalho.

2. OTIMIZAÇÃO GLOBAL CONTÍNUA

A otimização global contínua [12-15] é um campo bastante abrangente da pesquisa numérica. Nesse capítulo apresentaremos a definição formal de um problema de otimização global contínua, a definição de um problema de otimização sem restrições e alguns métodos clássicos para resolução desses problemas.

2.1 Definição do problema

Nessa seção apresentaremos a definição formal de um problema de otimização global contínua. Sem perda de generalidade, apenas os problemas de minimização serão estudados. Os problemas de maximização podem ser transformados em problemas de minimização invertendo-se o sinal de sua função objetivo. De forma geral, um Problema de Otimização Global Contínua (POGC) é definido matematicamente da seguinte forma:

$$\begin{aligned} &\text{Minimize} && f(x), \\ &\text{sujeito a} && g_i(x) \leq 0 \quad \text{para } i = 1, \dots, m \\ & && h_i(x) = 0 \quad \text{para } i = 1, \dots, l \\ & && x \in X \end{aligned}$$

A função f , as restrições g_i e h_i são funções definidas em \mathfrak{R}^n , X é um subconjunto de \mathfrak{R}^n e x é um vetor de n componentes, x_1, \dots, x_n . A função f é denominada função objetivo. As restrições g_i são denominadas restrições de desigualdade e as restrições h_i são denominadas restrições de igualdade. O conjunto X define os limites inferiores e superiores das variáveis. Um vetor $x \in X$ que satisfaz todas as restrições é uma *solução viável* do problema.

O problema de otimização global contínua é então encontrar um vetor \bar{x} , tal que $f(x) \geq f(\bar{x})$ para cada *solução viável* x . O vetor \bar{x} é denominado *solução ótima* (ou *solução*) do problema. Seja S um conjunto de soluções viáveis. Um vetor x' , tal que $f(x) \geq f(x')$ para cada solução viável $x \in S$ é denominado *ótimo local*.

De acordo com a presença ou não das restrições $g_i(x)$ ou $h_i(x)$, os POGC podem ser definidos como **Problemas de Otimização Com Restrições** ou **Problemas de**

Otimização Sem Restrições, respectivamente. Neste trabalho, apenas os problemas de otimização sem restrições serão explorados.

2.2 Otimização Sem Restrições

Conforme mencionado anteriormente, a otimização sem restrições lida com problemas de minimização ou maximização de uma função na ausência de restrições. Nessa seção discutiremos a minimização de funções de uma variável (Busca Linear) e a minimização de funções de várias variáveis (Busca Multidimensional).

Embora a maioria dos problemas práticos de otimização possua um conjunto de restrições, o estudo da otimização sem restrições é importante por vários motivos. Um dos principais motivos é que muitos algoritmos resolvem problemas de otimização com restrições, convertendo o problema numa sequência de problemas sem restrições. Essa conversão geralmente é feita utilizando funções de penalidades ou multiplicadores de Lagrange¹. Outro motivo importante é que os principais métodos de otimização encontram uma direção e minimizam a função nessa direção. Essa busca linear é equivalente a minimizar uma função de uma variável sem restrições. Além disso, várias técnicas de otimização sem restrições podem ser estendidas de forma natural para resolução de problemas de otimização com restrições.

2.2.1 Busca Linear

Resolver problemas de otimização global contínua envolve, em geral, realizar uma busca sobre o domínio do problema procurando uma solução ótima. Os principais métodos de busca utilizados para resolver esses problemas são métodos iterativos que, em cada iteração, calculam uma direção e tentam minimizar a função objetivo nessa direção. Essa busca sobre uma direção é denominada busca linear.

Nessa subseção apresentaremos alguns métodos que não utilizam cálculos de derivadas para minimização de funções de uma variável sobre um intervalo fechado.

¹ Multiplicadores de Lagrange é uma metodologia eficiente utilizada para eliminar a dependência de restrições.

2.2.1.1 Intervalo de incerteza

Considere o problema de busca linear de minimizar $f(x)$, onde $a \leq x \leq b$. Considere também que a localização exata do mínimo local de f sobre o intervalo $[a, b]$ é desconhecida. Nesse caso, o intervalo $[a, b]$ é denominado *intervalo de incerteza*. Durante o procedimento de busca, se porções desse intervalo (que não possuem o mínimo) podem ser excluídas então esse intervalo pode ser reduzido. Se uma função f é unimodal² então o intervalo de incerteza pode ser reduzido através da avaliação de dois pontos dentro do intervalo.

Seja, por exemplo, dois pontos $y, z \in [a, b]$ e $y < z$. Analisando a Figura 2.1, se f é uma função unimodal e $f(y)$ é maior que $f(z)$, então o intervalo de incerteza pode ser reduzido para $(y, b]$ (Figura 2.1 (a)). Por outro lado, se $f(y)$ é menor ou igual que $f(z)$, então o intervalo de incerteza pode ser reduzido para $[a, z]$ (Figura 2.1 (b)).

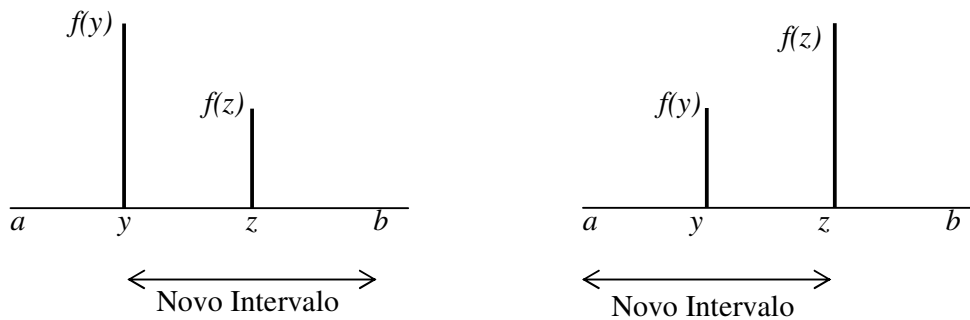


Figura 2.1. Redução do intervalo de incerteza.

2.2.1.2 Método da seção áurea

O método da seção áurea é um método de busca sequencial utilizado para minimização de funções unimodais. Os métodos de busca sequenciais são métodos iterativos, que têm como característica a utilização de informações de iterações anteriores na iteração atual.

A idéia do método da seção áurea é iterativamente reduzir o intervalo de incerteza do problema até um comprimento final l . No método da seção áurea, o

² Uma função $f: [a, b] \rightarrow \mathbb{R}$ é *unimodal* se ela possui apenas um mínimo global \bar{x} em $[a, b]$ e é estritamente decrescente em $[a, \bar{x}]$ e estritamente crescente em $[\bar{x}, b]$

intervalo de incerteza é reduzido em uma razão de 0,618, em cada iteração. Essa razão corresponde a parte fracionária da proporção áurea $\varphi \cong 1,618$.

Para cada iteração k do método da seção áurea, seja o intervalo de incerteza $[a_k, b_k]$. De acordo com a Seção 2.2.1.1, o novo intervalo de incerteza $[a_{k+1}, b_{k+1}]$ é equivalente a $[y_k, b_k]$ se $f(y_k) > f(z_k)$ ou equivalente a $[a_k, z_k]$ se $f(y_k) \leq f(z_k)$. Na Figura 2.2 está ilustrada a regra da seção áurea para os dois casos: $f(y) > f(z)$ (Caso 1) e $f(y) \leq f(z)$ (Caso 2).

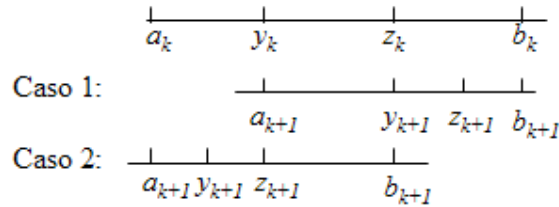


Figura 2.2. Redução do intervalo de incerteza pelo método da seção áurea.

```

procedimento MétodoSeçãoÁurea (x, f(.), a, b, l)
1    $\alpha \cong 0,618$ ;
2    $a_1 \leftarrow a$ ;  $b_1 \leftarrow b$ ;
3    $y_1 = a_1 + (1 - \alpha)(b_1 - a_1)$ ;
4    $z_1 = a_1 + \alpha(b_1 - a_1)$ ;
5    $k \leftarrow 1$ ;
6   enquanto  $|b_k - a_k| < l$  faça
7       se  $(f(y_k) > f(z_k))$  então
8            $a_{k+1} \leftarrow y_k$ ;
9            $b_{k+1} \leftarrow b_k$ ;
10           $y_{k+1} \leftarrow z_k$ ;
11           $z_{k+1} \leftarrow a_{k+1} + \alpha(b_{k+1} - a_{k+1})$ ;
12      senão
13           $a_{k+1} \leftarrow a_k$ ;
14           $b_{k+1} \leftarrow z_k$ ;
15           $z_{k+1} \leftarrow y_k$ ;
16           $y_{k+1} \leftarrow a_{k+1} + (1 - \alpha)(b_{k+1} - a_{k+1})$ ;
17      fim se;
18       $k \leftarrow k + 1$ ;
19  fim enquanto;
20  se  $f(a_k) < f(b_k)$  então
21      retorne  $a_k$ ;
22  senão
23      retorne  $b_k$ ;
fim MétodoSeçãoÁurea;

```

Figura 2.3. Pseudocódigo do método da seção áurea.

Na Figura 2.3 é apresentado o pseudocódigo do método da seção áurea. De acordo com a Figura 2.3, no método da seção áurea os pontos y_k e z_k são selecionados da seguinte forma:

$$y_k = a_k + (1 - \alpha)(b_k - a_k),$$

$$z_k = a_k + \alpha(b_k - a_k),$$

$$\text{onde } \alpha \cong 0,618$$

Em cada iteração k , se y_k e z_k são escolhidos de acordo com as equações acima, então o intervalo de incerteza é reduzido na razão 0,618. Na primeira iteração, são necessárias duas avaliações da função objetivo, uma no ponto y_1 e a outra no ponto z_1 . Nas iterações seguintes, no entanto, apenas uma avaliação da função objetivo é necessária, uma vez que, $y_{k+1} = z_k$ ou $z_{k+1} = y_k$.

2.2.1.3 Método de Fibonacci

O método de Fibonacci é um procedimento de busca linear utilizado para minimizar funções unimodais sobre um intervalo limitado. De forma análoga ao método da seção áurea, o método de Fibonacci realiza duas avaliações de função na primeira iteração e uma avaliação de função nas iterações seguintes. Contudo, diferentemente do método da seção áurea que reduz o intervalo de incerteza na razão 0,618, o método de Fibonacci reduz o intervalo de acordo com a sequência de Fibonacci.

A sequência de Fibonacci $\{F_v\}$ é definida da seguinte forma:

$$F_{v+1} = F_v + F_{v-1}, \quad v = 1, 2, \dots$$

$$F_0 = F_1 = 1$$

Para cada iteração k , seja o intervalo de incerteza $[a_k, b_k]$. Seja n o número total de avaliações de função planejadas, os pontos y_k e z_k são selecionados da seguinte forma:

$$y_k = a_k + \frac{F_{n-k-1}}{F_{n-k+1}}(b_k - a_k), \quad k = 1, \dots, n-1,$$

$$z_k = a_k + \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k), \quad k = 1, \dots, n-1,$$

De forma análoga ao método da seção áurea, o novo intervalo de incerteza $[a_{k+1}, b_{k+1}]$ é equivalente a $[y_k, b_k]$ se $f(y) > f(z)$ ou equivalente a $[a_k, z_k]$ se $f(y) \leq f(z)$.

2.2.2 Busca Multidimensional

Nessa seção consideraremos o problema de minimizar uma função f de várias variáveis. Na subseção 2.2.2.1 serão apresentados alguns métodos de resolução desses problemas que não utilizam cálculos de derivadas da função objetivo, também conhecidos como métodos de busca direcionada (*Direct Search Methods* - DSM). Na subseção 2.2.2.2 serão apresentados métodos que utilizam o cálculo de derivadas na determinação das direções de busca. De modo geral, esses métodos funcionam inicialmente calculando uma direção de descida d . Em seguida, realizam uma busca linear a partir de x na direção d .

2.2.2.1 Métodos de Busca Direcionada (*Direct Search Methods* (DSM))

Os métodos de busca direcionada podem ser definidos como procedimentos que tentam direcionar a busca para um mínimo global sem utilizar informações de derivadas da função objetivo como, por exemplo, o vetor gradiente. Propostos na década de 1950, esses métodos sofreram, inicialmente, certo desinteresse da comunidade de otimização, uma vez que não existiam provas matemáticas sobre a convergência desses métodos [16]. Além disso, esses métodos eram geralmente derivados de exemplos desenhados em duas dimensões sem provas matemáticas [16], conforme podemos observar em [17].

Contudo, por serem métodos simples de implementar e facilmente aplicáveis, esses métodos se tornaram populares na comunidade das engenharias e das ciências práticas. Além disso, muitos usuários preferiam evitar o cálculo de gradientes que eram a maior fonte de erro em bibliotecas de *software* para otimização [18].

Por volta de 1991, foi divulgado um trabalho [19] que apresentava um método de busca direcionado acompanhado de uma análise de convergência. A partir de então, os métodos de busca direcionada passaram a ser mais aceitos e difundidos na comunidade de otimização.

Nessa Seção serão apresentados dois métodos de busca direcionada, o método Nelder-Mead (*Nelder-Mead Method* - NMM) e o método de Busca por Padrões (*Pattern Search Method* - PSM).

2.2.2.1.1 Método Nelder-Mead

O método Nelder-Mead [20] é um dos mais populares métodos de busca direcionada. Ao invés de utilizar cálculos de derivada da função objetivo, o método Nelder-Mead utiliza em cada iteração, um simplex não-degenerado, um polítopo³ de $n+1$ vértices em n dimensões. Por exemplo, um segmento de linha sobre uma linha, um triângulo sobre um plano, um tetraedro em um espaço tridimensional são exemplos de simplex.

O método Nelder-Mead inicia a partir de um simplex de entrada. Em cada iteração, a partir do simplex atual, novos pontos são calculados para gerar um novo simplex. Para calcular esses novos pontos, quatro coeficientes escalares são definidos: reflexão ρ , expansão χ , contração γ e retração σ .

Mais especificamente, inicialmente, os $n+1$ pontos do simplex são ordenados em ordem crescente de valor da função objetivo, ou seja, x_1, x_2, \dots, x_{n+1} , onde $f(x_1) < f(x_2) < \dots < f(x_{n+1})$. Em seguida, calcula-se o centróide dos n melhores vértices $\bar{x} = (\sum_{i=1}^n x_i) / n$. Calcula-se então o ponto de reflexão x_r . O ponto x_r é calculado como um reflexo do ponto x_{n+1} , ou seja, na face oposta de x_{n+1} . Como o vértice x_{n+1} é o vértice com maior valor de função objetivo, espera-se encontrar um ponto melhor na face oposta de x_{n+1} .

Se o ponto de reflexão x_r calculado for o vértice com menor valor de função objetivo, ou seja, $f(x_r) < f(x_1)$, espera-se encontrar pontos ainda melhores na direção de \bar{x} para x_r . Nesse caso, calcula-se um ponto de expansão x_e nessa direção. Caso contrário, se $f(x_r) \geq f(x_n)$, um ponto melhor provavelmente estará localizado dentro do simplex x_1 a x_{n+1} . Calcula-se então o ponto de contração x_c . Se o ponto de contração não for melhor que x_{n+1} , aplica-se a operação de retração para gerar o novo simplex. O melhor simplex gerado na iteração formará o simplex da próxima iteração. O procedimento termina quando a diferença $f(x_{n+1}) - f(x_1)$ se torna menor que o escalar ε , um parâmetro configurado pelo usuário do método. O pseudocódigo do método Nelder-Mead é apresentado na Figura 2.4.

³ Em geometria, um polítopo é a generalização, para um número arbitrário de dimensões (finitas), dos conceitos de polígono e poliedro.

```

procedimento Nelder-Mead( $x_1, \dots, x_{n+1}, n, f(\cdot), \varepsilon$ )
1   repita
2       Ordene  $x_1, x_2, \dots, x_{n+1}$ , tal que  $f(x_1) < f(x_2) < \dots < f(x_{n+1})$ ;
3       Calcule o centróide  $\bar{x} = (\sum_{i=1}^n x_i) / n$ ;
4        $x_r = \bar{x} + \rho(\bar{x} - x_{n+1})$ ; /* Reflexão */
5       se  $f(x_1) \leq f(x_r) < f(x_{n+1})$  então
6            $x_{n+1} \leftarrow x_r$ ;
7           continue;
8       senão se  $f(x_r) < f(x_1)$  então
9            $x_e = \bar{x} + \chi(x_r - \bar{x})$ ; /* Expansão */
10          se  $f(x_e) \leq f(x_r)$  então
11               $x_{n+1} \leftarrow x_e$ ;
12          senão
13               $x_{n+1} \leftarrow x_r$ ;
14          fim se;
15          continue;
16      senão se  $f(x_r) \geq f(x_n)$  então
17           $x_c = \bar{x} + \gamma(x_{n+1} - \bar{x})$ ; /* Contração */
18          se  $f(x_c) \leq f(x_{n+1})$  então
19               $x_{n+1} \leftarrow x_c$ ;
20          fim se;
21          continue;
22      fim se;
23       $x'_i = x_1 + \sigma(x_i - x_1)$ ,  $i = 2, \dots, n+1$ ; /* Retração */
24      Troque os vértices  $x_2, \dots, x_{n+1}$  com  $x'_2, \dots, x'_{n+1}$ ;
25  até  $f(x_{n+1}) - f(x_1) < \varepsilon$ ;
26  fim repita;
fim Nelder-Mead

```

Figura 2.4. Pseudocódigo do procedimento Nelder-Mead.

Na Figura 2.5 é apresentado um exemplo de geração dos pontos de reflexão, expansão, contração e retração para um simplex de duas dimensões.

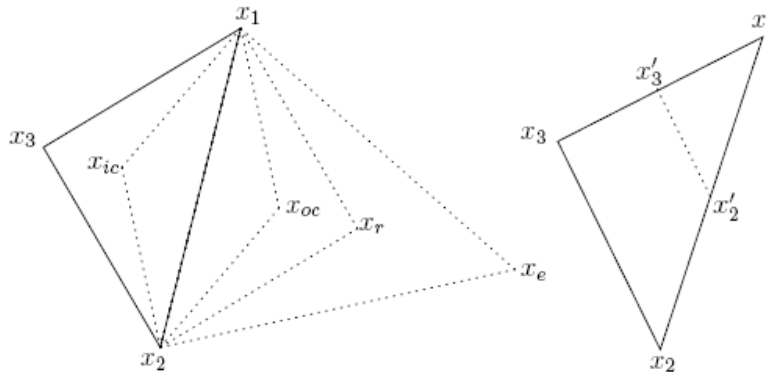


Figura 2.5. Simplex de duas dimensões. Operações de reflexão, expansão, contração e retração.

2.2.2.1.2 Método de Busca por Padrões (Pattern Search Method-PSM)

O método de Busca por Padrões foi originalmente proposto por Hooke e Jeeves [21]. Posteriormente, Torczon [19] apresentou uma generalização para os métodos de busca por padrões, denominada *Generalized Pattern Search* (GPS).

Os métodos de Busca por Padrões direcionam a busca para um mínimo local utilizando um padrão formado por $n+1$ pontos em cada iteração. Esse padrão é formado pela solução atual x e um conjunto de n pontos gerados a partir de x . Esse conjunto é gerado utilizando diferentes direções e um determinado tamanho do passo. Geralmente, essas direções de busca formam uma base do espaço vetorial \mathfrak{R}^n . Com o objetivo de se obter uma convergência mais rápida, contudo, outras direções de busca mais promissoras podem ser utilizadas. O tamanho do passo é reduzido sempre que a busca não consegue gerar um movimento melhor. Na Figura 2.6 é apresentado o pseudocódigo do procedimento GPS.

```

procedimento GPS ( $x_0, D, \lambda_0, f(\cdot)$ )
1    $k \leftarrow 0$ ;
2   enquanto Critério de parada não satisfeito faça
3       Calcule  $M_k$ ;
4       Invoque busca para recuperar um ponto melhor em  $M_k$ ;
5       se não houve melhora então
6           Escolha um conjunto  $D_k \subset D$ ;
7           Calcule  $P_k$ ;
8           Avalie todos os pontos em  $P_k$ ;
9       fim se;
10      se houve melhora então
11           $x_{k+1} \leftarrow$  ponto melhor;
12          Configure um  $\lambda_{k+1} > \lambda_k$ ;
13      senão
14           $x_{k+1} \leftarrow x_k$ ;
15          Configure um  $\lambda_{k+1} < \lambda_k$ ;
16      fim se;
17       $k \leftarrow k+1$ ;
18  fim enquanto;
fim GPS;

```

Figura 2.6. Pseudocódigo do procedimento GPS

De acordo com a Figura 2.6, o GPS possui como parâmetros de entrada: um conjunto de direções de descida D , uma solução inicial x_0 e um tamanho de passo $\lambda_0 > 0$. Em cada iteração, são invocadas duas fases de busca. A primeira fase, denominada *Search Step*, gera soluções a partir de um conjunto de pontos M_k . A segunda fase, denominada *Poll Step*, é formada por uma busca sistemática para explorar a região ao redor da solução atual utilizando o conjunto P_k . Se algum movimento melhor é gerado, então a busca continua com o mesmo tamanho de passo. Caso contrário, o tamanho de passo é reduzido. O procedimento termina quando o tamanho de passo se torna muito pequeno.

Os conjuntos M_k e P_k são definidos da seguinte forma:

$$M_k = \{y : y = x_k + \lambda_k dz \in X, d \in D, z \in \mathbb{Z}_+^{|D|}\},$$

$$P_k = \{y : y = x_k + \lambda_k d \in X, d \in D_k \subset D\}$$

Onde D é o conjunto de direções de descida inicial, D_k é o conjunto de direções atual, λ_k é o tamanho de passo atual e \mathbb{Z}_+ é o conjunto de todos os inteiros positivos.

2.2.2.2 Métodos de Busca Multidimensional com Derivadas

Nessa seção apresentaremos os métodos de busca multidimensional que utilizam cálculos de derivadas para determinar uma direção de descida d . Na Seção 2.2.2.2.1 será apresentado o método de máxima descida (*Steepest Descent Method* - SDM), na Seção 2.2.2.2.2 será apresentado o método de Newton e na Seção 2.2.2.2.3 serão apresentados os métodos quasi-Newton. Por fim, na Seção 2.2.2.2.4 será apresentado o método BFGS com memória limitada (*Limited Memory BFGS* – LBFGS), uma adaptação do método quasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS).

2.2.2.2.1 Método de máxima descida (*Steepest Descent Method*-SDM)

O método de máxima descida, ou método do gradiente, é um dos procedimentos mais fundamentais para minimização de funções diferenciáveis de várias variáveis. O método de máxima descida é um algoritmo seqüencial que utiliza uma direção de descida d baseada no vetor gradiente.

O vetor gradiente de uma função f em x é definido da seguinte forma:

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \dots, \frac{\partial f(x)}{\partial x_n} \right).$$

Se a função f é diferenciável em x e a norma do vetor gradiente é diferente de zero, então o vetor $d = -\nabla f(x)$ é a direção de máxima descida. A partir da solução atual x , o método realiza, em cada iteração, uma busca linear na direção $-\nabla f(x)$. Em seguida, atualiza-se a solução x e inicia-se uma nova iteração. O método termina quando a norma do vetor gradiente $\|\nabla f(x)\|$ é menor que um escalar ε , um parâmetro configurado pelo método. O pseudocódigo do método do gradiente é apresentado na Figura 2.7.

```

procedimento Método-Gradiente ( $x_1, f(\cdot), \varepsilon$ )
1    $k \leftarrow 1$ ;
2   enquanto  $\|\nabla f(x_k)\| < \varepsilon$  faça
3        $d_k = -\nabla f(x_k)$ ;
4       Minimize  $f(x_k + \lambda_k d_k)$ , sujeito a  $\lambda_k \geq 0$ ;
5        $x_{k+1} \leftarrow x_k + \lambda_k d_k$ ;
6        $k \leftarrow k+1$ ;
7   fim enquanto;
8   retorne  $x_k$ ;
fim Método-Gradiente;

```

Figura 2.7. Pseudocódigo do método do gradiente.

Tabela 2.1. Resumo da execução do método da descida de máximo declive.

Iteração k	x_k $f(x_k)$	$\nabla f(x_k)$	$\ \nabla f(x_k)\ $	λ_k	x_{k+1}
1	(0,00; 3,0) 52,0	(-44,0; 24,0)	50,12	0,062	(2,70; 1,51)
2	(2,70; 1,51) 0,34	(0,73; 1,28)	1,47	0,24	(2,52; 1,20)
3	(2,52; 1,20) 0,09	(0,80; -0,48)	0,93	0,11	(2,43; 1,25)
4	(2,43; 1,25) 0,04	(0,18; 0,28)	0,33	0,31	(2,37; 1,16)
5	(2,37; 1,16) 0,02	(0,30; -0,20)	0,36	0,12	(2,33; 1,18)
6	(2,33; 1,18) 0,01	(0,08; 0,12)	0,14	0,36	(2,30; 1,14)
7	(2,30; 1,14) 0,009	(0,15; -0,08)	0,17	0,13	(2,28; 1,15)
8	(2,28; 1,15) 0,007	(0,05; 0,08)	0,09		

Para exemplificar o funcionamento do método de máxima descida considere o seguinte problema: minimize $(x_1-1)^4 + (x_1-2x_2)^2$, partindo do ponto $(0,0;3,0)$. Um resumo da execução desse método para esse exemplo é apresentado na Tabela 2.1.

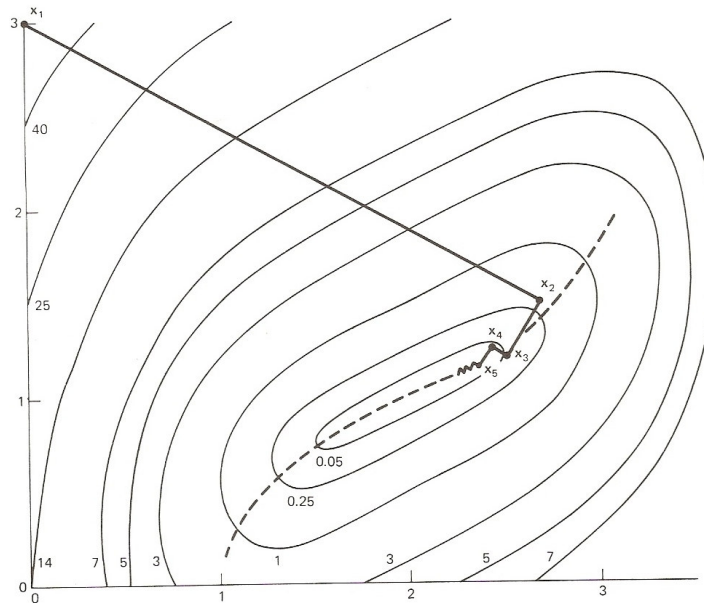


Figura 2.8. Ilustração geométrica do método de máxima descida.

De acordo com os dados da Tabela 2.1, após sete iterações o ponto $x_8 = (2,28; 1,15)$ é alcançado. O algoritmo termina porque a norma do gradiente de x_8 é pequena ($\|\nabla f(x_8)\| = 0,09$). Na Figura 2.8 é apresentado de que modo o método progride geometricamente ao longo da função f . O ponto de mínimo é o ponto $(2,0; 1,0)$.

Analisando a Figura 2.8, podemos observar que o método de máxima descida funciona bem durante os primeiros estágios do processo de otimização dependendo do ponto de inicialização. Contudo, quando se aproxima de um ponto de mínimo (local ou global), o método usualmente apresenta uma convergência mais lenta com passos curtos e ortogonais. Esse fenômeno é denominado *zigzagging*.

2.2.2.2.2 Método de Newton

O método de Newton é fundamental em várias áreas da matemática computacional. Em sua forma básica, o método de Newton é utilizado para resolver sistemas de equações não-lineares. No entanto, ele também admite uma interpretação ligada à otimização. Quando aplicado a problemas de otimização, o método de Newton,

diferentemente dos métodos de descida, tem a mesma preferência por minimizadores, maximizadores ou quaisquer outros pontos estacionários de f .

O método de Newton procura defletir a direção de descida de máximo declive através da multiplicação dessa direção pela inversa da matriz Hessiana, $H(x)^{-1}$. Para um dado ponto x , a matriz Hessiana é uma matriz composta pelas derivadas parciais de segunda ordem de x . A matriz Hessiana é então definida da seguinte forma:

$$H(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \dots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

O método de Newton é similar ao método do gradiente. É um procedimento iterativo que, em cada iteração, realiza uma busca linear em uma direção $d = -H(x)^{-1} \nabla f(x)$. Portanto, para uma dada iteração k , assumindo que a inversa da matriz Hessiana, $H(x_k)^{-1}$, existe, então a solução x_{k+1} é definida por:

$$x_{k+1} = x_k - H(x_k)^{-1} \nabla f(x_k)$$

Por utilizar informações de segunda ordem, o método de Newton possui uma rápida convergência. Contudo, uma iteração desse método exige um esforço computacional significativamente maior que o esforço computacional do método do gradiente.

Para exemplificar o funcionamento do método de Newton, como também a sua rápida convergência, tomemos o mesmo problema descrito na Seção 2.2.2.2.1: minimize $(x_1-1)^4 + (x_1-2x_2)^2$, partindo do ponto $(0,0; 3,0)$. Um resumo da execução do método de Newton para esse exemplo é apresentado na Tabela 2.2.

De acordo com os dados da Tabela 2.2, após seis iterações o ponto $x_7 = (1,83; 0,91)$ é alcançado. Nesse ponto, $\|\nabla f(x_7)\| = 0,04$ e o procedimento termina. Na Figura 2.9 é mostrado como o método progride geometricamente ao longo da função f .

Pelo exemplo acima, o valor da função objetivo sempre decresce a cada iteração. Contudo, isso nem sempre ocorrerá. Conforme já comentado anteriormente, o método de Newton não tem preferência por qualquer ponto estacionário de f , o que pode acarretar a não utilização de f como uma função de descida. De fato, o método de

Newton convergirá se o ponto de partida estiver próximo de um ponto ótimo. Por esse motivo, o método de Newton é considerado um método de *convergência local*.

Tabela 2.2. Resumo da execução do método de Newton.

k	x_k $f(x_k)$	$\nabla f(x_k)$	$H(x_k)^{-1}$	$-H(x_k)^{-1}\nabla f(x_k)$	x_{k+1}
1	(0,00; 3,0) 52,0	(-44,0; 24,0)	$\frac{1}{384} \begin{bmatrix} 8,0 & 4,0 \\ 4,0 & 50,0 \end{bmatrix}$	(-0,67; -2,67)	(0,67; 0,33)
2	(0,67; 0,33) 3,13	(-9,39; -0,04)	$\frac{1}{169,84} \begin{bmatrix} 8,0 & 4,0 \\ 4,0 & 23,3 \end{bmatrix}$	(0,44; 0,23)	(1,11; 0,56)
3	(1,11; 0,56) 0,63	(-2,84; -0,04)	$\frac{1}{76} \begin{bmatrix} 8,0 & 4,0 \\ 4,0 & 11,5 \end{bmatrix}$	(0,30; 0,14)	(1,41; 0,70)
4	(1,41; 0,70) 0,12	(-0,80; -0,04)	$\frac{1}{33,44} \begin{bmatrix} 8,0 & 4,0 \\ 4,0 & 6,18 \end{bmatrix}$	(0,20; 0,10)	(1,61; 0,80)
5	(1,61; 0,80) 0,02	(-0,22; -0,04)	$\frac{1}{14,64} \begin{bmatrix} 8,0 & 4,0 \\ 4,0 & 3,83 \end{bmatrix}$	(0,13; 0,07)	(1,74; 0,87)
6	(1,74; 0,87) 0,005	(-0,07; 0,00)	$\frac{1}{6,48} \begin{bmatrix} 8,0 & 4,0 \\ 4,0 & 2,81 \end{bmatrix}$	(0,09; 0,04)	(1,83; 0,91)
7	(1,83; 0,91) 0,009	(0,00; -0,04)			

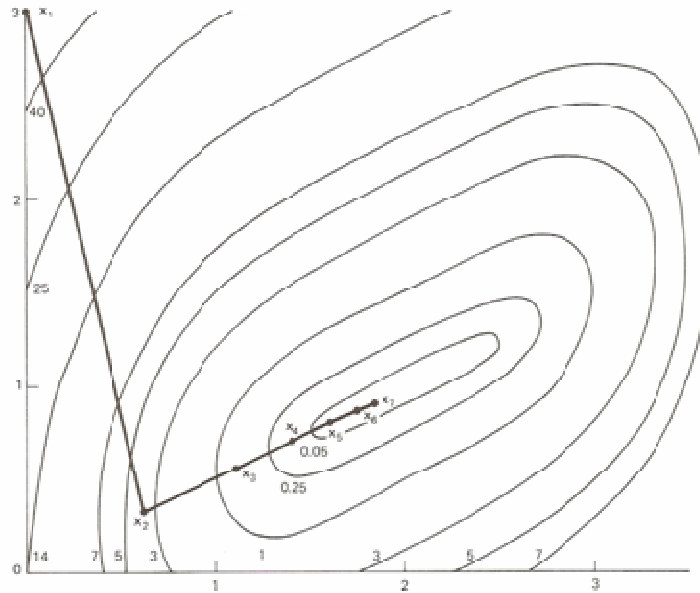


Figura 2.9. Ilustração geométrica do método de Newton.

2.2.2.2.3 Métodos quasi-Newton

Os métodos quasi-Newton, como o próprio nome indica, são uma classe de procedimentos inspirados no método de Newton. A direção de descida utilizada nos métodos quasi-Newton é a direção $-D_j \nabla f(y)$, em analogia à direção $-H^{-1}(y) \nabla f(y)$ utilizada pelo método de Newton. A matriz D_j é uma aproximação da inversa da matriz Hessiana.

Os dois principais métodos quasi-Newton são os métodos Davidon-Fletcher-Powell (DFP) e o método Broyden-Fletcher-Goldfarb-Shanno (BFGS). Nesses métodos, a matriz D_{j+1} é formada pela adição de D_j a uma matriz de correção C_j ($D_{j+1} = D_j + C_j$). A principal diferença entre os métodos é a forma como a matriz de correção C_j é calculada. O pseudocódigo dos métodos quasi-Newton é apresentado na Figura 2.10.

```

procedimento Quasi-Newton ( $x, n, f(\cdot), \varepsilon$ )
1    $y_1 \leftarrow x$ ;
2   enquanto  $\| \nabla f(y_j) \| < \varepsilon$  faça
3       para  $j = 1, \dots, n$  faça
4            $d_j = -D_j \nabla f(y_j)$ ;
5           Minimize  $f(y_j + \lambda d_j)$ , sujeito a  $\lambda \geq 0$ ;
6            $y_{j+1} = y_j + \lambda d_j$ ;
7            $D_{j+1} = D_j + C_j$ ;
8       fim para;
9        $y_1 = y_{n+1}$ ;
10  fim enquanto;
fim Quasi-Newton

```

Figura 2.10. Pseudocódigo de um método quase-Newton

Podemos observar analisando o pseudocódigo da Figura 2.10 que o algoritmo reinicia o procedimento de busca após n passos. O esquema de atualização das matrizes do DFP e do BFGS produz uma representação exata da matriz Hessiana após esses n passos.

No método DFP, a matriz C_j é calculada da seguinte forma:

$$C_j = \frac{p_j p_j^t}{p_j^t p_j} - \frac{D_j q_j q_j^t D_j}{q_j^t D_j q_j}, \text{ onde}$$

$$\mathbf{p}_j = \lambda d_j \equiv \mathbf{y}_{j+1} - \mathbf{y}_j$$

$$\mathbf{q}_j = \nabla f(\mathbf{y}_{j+1}) - \nabla f(\mathbf{y}_j)$$

No método BFGS, a matriz \mathbf{C}_j é calculada da seguinte forma:

$$\mathbf{C}_j = \frac{\mathbf{p}_j \mathbf{p}_j^t}{\mathbf{p}_j^t \mathbf{q}_j} \left[1 + \frac{\mathbf{q}_j^t \mathbf{D}_j \mathbf{q}_j}{\mathbf{p}_j^t \mathbf{q}_j} \right] - \frac{[\mathbf{D}_j \mathbf{q}_j \mathbf{p}_j^t + \mathbf{p}_j \mathbf{q}_j^t \mathbf{D}_j]}{\mathbf{p}_j \mathbf{q}_j}, \text{ onde}$$

$$\mathbf{p}_j = \lambda d_j \equiv \mathbf{y}_{j+1} - \mathbf{y}_j$$

$$\mathbf{q}_j = \nabla f(\mathbf{y}_{j+1}) - \nabla f(\mathbf{y}_j)$$

Para exemplificar a eficiência dos métodos quasi-Newton, na Figura 2.11 é ilustrado o funcionamento do método DFP para o problema: minimize $(x_1-1)^4 + (x_1-2x_2)^2$, partindo do ponto (0,0; 3,0), o mesmo problema descrito nas seções 2.2.2.2.1 e 2.2.2.2.2.

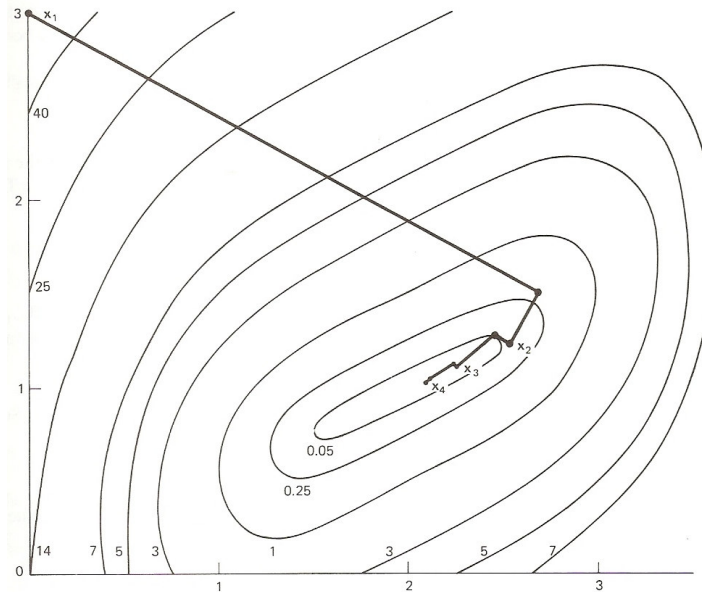


Figura 2.11. Ilustração do método DFP.

2.2.2.2.4 Método BFGS com memória limitada (LBFGS)

O método BFGS com memória limitada (*Limited Memory BFGS* - LBFGS), criado por Nocedal [22,23], é um procedimento derivado do método quasi-Newton BFGS para aplicações que possuem restrições de armazenamento.

Ao invés de utilizar informações sobre todas as j iterações anteriores (conforme o BFGS), o LBFGS utiliza uma fórmula de atualização da matriz quasi-Newton D_j contendo, no máximo, informações sobre as últimas m iterações, onde m é um parâmetro configurado pelo usuário. Em consequência disso, nas primeiras j iterações, quando j é inferior a m , a atualização da matriz quasi-Newton é similar a atualização do BFGS. Contudo, quando j torna-se superior a m , tem-se uma atualização da matriz quasi-Newton com restrições de memória (i.e. utilizando apenas informações sobre m iterações). Nesse caso, em cada passo, o LBFGS elimina a informação da iteração mais antiga e adiciona a informação da nova iteração.

Sejam as variáveis p_j e q_j , equivalentes as variáveis do BFGS (seção 2.2.2.2.3):

$$p_j = y_{j+1} - y_j, \quad q_j = \nabla f(y_{j+1}) - \nabla f(y_j)$$

Conforme deduzido por Nocedal [22], no método LBFGS, quando $j+1 \leq m$, onde j é a iteração corrente e m é o número máximo de iterações armazenadas, a atualização da matriz quasi-Newton é calculada da seguinte forma:

$$\begin{aligned} D_{j+1} &= v_j^t v_{j-1}^t \cdots v_0^t D_0 v_0 \cdots v_{j-1} v_j \\ &\quad + v_j^t \cdots v_1^t \rho_0 p_0 p_0^t v_1 \cdots v_j \\ &\quad \vdots \\ &\quad + v_j^t \rho_{j-1} p_{j-1} p_{j-1}^t v_j \\ &\quad + \rho_j p_j p_j^t \end{aligned}$$

Onde, $\rho_j = 1/q_j^t p_j$, $v_j = (I - \rho_j q_j p_j^t)$ e I é a matriz identidade.

Por outro lado, quando $j+1 > m$, a atualização da matriz quasi-Newton é calculada da seguinte forma:

$$\begin{aligned} D_{j+1} &= v_j^t v_{j-1}^t \cdots v_{j-m+1}^t D_0 v_{j-m+1} \cdots v_{j-1} v_j \\ &\quad + v_j^t \cdots v_{j-m+2}^t \rho_{j-m+1} p_{j-m+1} p_{j-m+1}^t v_{j-m+2} \cdots v_j \\ &\quad \vdots \\ &\quad + v_j^t \rho_{j-1} p_{j-1} p_{j-1}^t v_j \\ &\quad + \rho_j p_j p_j^t \end{aligned}$$

Nesse caso, podemos observar que apenas informações sobre as m últimas iterações são utilizadas no cálculo da matriz quasi-Newton, reduzindo a quantidade de informação armazenada.

3. METAHEURÍSTICAS

Conforme mencionado no Capítulo 1, até os dias de hoje, a utilização de métodos exatos para resolução de problemas de otimização combinatória é bastante restrita. Isso ocorre porque, para esses problemas, à medida que o tamanho da entrada aumenta o número de operações necessárias para resolvê-los aumenta exponencialmente, tornando sua resolução impraticável.

Devido a essas dificuldades, alguns pesquisadores concentraram esforços para desenvolver algoritmos eficientes capazes de guiar o procedimento de busca e de encontrar “boas” soluções. Esses algoritmos, denominados heurísticas [24], procuram encontrar boas soluções com um baixo custo computacional, mesmo sem garantir que a solução encontrada seja uma solução ótima do problema. Na prática, em geral, para esses problemas, é suficiente encontrar uma boa solução ao invés do ótimo global, o qual só poderia ser encontrado após um imenso esforço computacional.

As heurísticas, em geral, eram projetadas para resolver problemas específicos, não garantindo sua aplicabilidade para uma classe mais abrangente de problemas. Posteriormente, com o objetivo de contornar esse problema, os pesquisadores começaram a projetar heurísticas de caráter mais geral, as quais denominaram de metaheurísticas. As metaheurísticas são, portanto, heurísticas de caráter geral que possuem a capacidade de escapar de ótimos locais. Uma de suas características marcantes é a freqüente inspiração em conceitos de física, sistemas biológicos, inteligência artificial, entre outros.

De acordo com a forma que as metaheurísticas exploram o espaço de soluções, elas podem ser divididas em duas categorias: as metaheurísticas de busca local e as metaheurísticas de busca populacional. Nas metaheurísticas de busca local, o procedimento de busca utiliza uma solução como ponto de partida em cada iteração. As metaheurísticas GRASP, Busca Tabu e *Simulated Annealing* podem ser citadas como exemplos de metaheurísticas ponto-a-ponto. Nas metaheurísticas de busca populacionais, por outro lado, um conjunto de soluções de boa qualidade é combinado com o intuito de produzir soluções melhores. Podemos citar como exemplo de métodos populacionais, os Algoritmos Genéticos, Colônia de Formigas (*Ant Colony System*), *Particle Swarm Optimization* (PSO) etc.

Recentemente, algumas dessas metaheurísticas, como GRASP, Busca Tabu, *Simulated Annealing*, originalmente propostas para tentar resolver problemas de otimização combinatória, foram adaptadas para resolver problemas de otimização global contínua.

Nas próximas seções serão apresentadas algumas importantes metaheurísticas utilizadas para tentar resolver problemas de otimização combinatória e otimização contínua. Na Seção 3.1 serão apresentadas algumas metaheurísticas utilizadas para resolução de problemas de otimização combinatória. Na Seção 3.2 serão apresentadas algumas metaheurísticas utilizadas para resolução de problemas de otimização contínua.

3.1. Metaheurísticas para otimização combinatória

Nessa seção apresentaremos algumas importantes metaheurísticas desenvolvidas para resolução de problemas de otimização combinatória. Na Seção 3.1.1 será apresentada a metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP). Na Seção 3.1.2 será apresentada a metaheurística Busca Tabu (*Tabu Search* - TS). Na Seção 3.1.3 será apresentada a metaheurística *Simulated Annealing*. Finalmente na Seção 3.1.4 será apresentada a metaheurística populacional Algoritmos Genéticos.

3.1.1 *Greedy Randomized Adaptive Search Procedure* (GRASP)

A metaheurística GRASP, desenvolvida por Feo e Resende [1], é um método iterativo composto de duas fases: uma fase de construção de uma solução e uma fase de busca local. Na fase de construção, uma solução de boa qualidade é gerada através de uma mistura de gulosidade e aleatoriedade. A partir desta solução, uma busca local é aplicada com o objetivo de melhorar esta solução. Em cada iteração uma solução é encontrada, sendo a melhor de todas elas considerada a solução final do problema. O pseudocódigo do procedimento GRASP é apresentado na Figura 3.1.

Na fase de construção uma solução é iterativamente construída, elemento por elemento. A cada iteração, inicialmente, uma função adaptativa e gulosa estima o benefício da seleção de cada um dos elementos que podem ser adicionados à solução. Em seguida, uma lista com os melhores elementos, denominada Lista de Candidatos Restrita (LCR), é construída. O tamanho da LCR é determinado pelo parâmetro

$\alpha \in [0,1]$. Por fim, um elemento da LCR é escolhido aleatoriamente e adicionado à solução. O pseudocódigo do procedimento de construção do GRASP é apresentado na Figura 3.2.

```

procedimento GRASP( $f(\cdot)$ ,  $g(\cdot)$ ,  $N(\cdot)$ ,  $GRASPMax$ ,  $s$ )
1    $f^* \leftarrow \infty$ ;
2   para 1,2,...,  $GRASPMax$  faça
3       Construção( $g(\cdot)$ ,  $\alpha$ ,  $s$ );
4       BuscaLocal( $f(\cdot)$ ,  $N(\cdot)$ ,  $s$ )
5       se  $f(s) < f^*$  então
6            $s^* \leftarrow s$ ;
7            $f^* \leftarrow f(s)$ ;
8       fim se;
9   fim para;
10  retorne  $s^*$ ;
fim GRASP

```

Figura 3.1. Pseudocódigo do GRASP.

```

procedimento Construção( $g(\cdot)$ ,  $\alpha$ ,  $s$ )
1    $s \leftarrow \emptyset$ ;
2   Inicialize o conjunto  $C$  de candidatos;
3   enquanto  $C \neq \emptyset$  faça
4        $g(t_{min}) \leftarrow \min \{ g(t) \mid t \in C \}$ ;
5        $g(t_{max}) \leftarrow \max \{ g(t) \mid t \in C \}$ ;
6        $LCR \leftarrow \{ t \in C \mid g(t) \leq g(t_{min}) + \alpha(g(t_{max}) - g(t_{min})) \}$ ;
7       Selecione aleatoriamente um elemento  $t \in LCR$ ;
8        $s \leftarrow s \cup \{t\}$ ;
9       Atualize conjunto de candidatos;
10  fim enquanto;
11  retorne  $s$ ;
fim Construção

```

Figura 3.2. Pseudocódigo do procedimento de construção do GRASP.

O nome *Greedy Randomized Adaptive Search Procedure* (GRASP) deriva do caráter guloso, aleatório e adaptativo da fase de construção. Uma característica importante a ser observada é que o parâmetro α controla o nível de gulosidade e aleatoriedade do procedimento. Um valor de α igual a zero, por exemplo, acarreta a construção de soluções puramente gulosas, enquanto que um valor de α igual a um, acarreta construção de soluções totalmente aleatórias. O procedimento de construção do GRASP procura, portanto, conjugar os bons aspectos dos algoritmos puramente gulosos com os bons aspectos dos procedimentos aleatórios de construção de soluções.

Contudo, as soluções obtidas pelo procedimento de construção não são necessariamente soluções ótimas locais. Em outras palavras, essas soluções não são necessariamente a melhor solução de sua vizinhança⁴. Em consequência disso, o GRASP utiliza uma fase de busca local para tentar melhorar essa solução.

Na busca local, movimentos são definidos para o problema em particular e, a partir desses movimentos, é efetuada uma busca na vizinhança da solução. Sempre que uma solução visitada s' for melhor que a solução atual s , ela substitui a solução atual. O procedimento termina quando nenhuma solução melhor é encontrada na vizinhança da solução atual. O pseudocódigo do procedimento busca local do GRASP para uma vizinhança $N(\cdot)$ de s é apresentado na Figura 3.3.

```

procedimento BuscaLocal( $f(\cdot)$ ,  $N(\cdot)$ ,  $s$ )
1    $V \leftarrow \{s' \in N(s) \mid f(s') < f(s)\};$ 
2   enquanto  $|V| > 0$  faça
3       Selecione  $s'$  de  $V$ ;
4        $s \leftarrow s'$ ;
5        $V \leftarrow \{s' \in N(s) \mid f(s') < f(s)\};$ 
6   fim enquanto;
7   retorne  $s$ ;
fim BuscaLocal

```

Figura 3.3. Pseudocódigo do procedimento de busca local do GRASP.

3.1.2 Busca Tabu

A metaheurística Busca Tabu (*Tabu Search*), desenvolvida por Glover e Hansen [2,3], é um método de busca local que explora o espaço de soluções movimentando-se de uma solução para outra que seja seu melhor vizinho. Para escapar dos ótimos locais, esses movimentos no espaço de soluções se baseiam em uma estrutura de memória que armazena características das soluções, ou eventualmente, até as próprias soluções.

Mais especificamente, na Busca Tabu, em cada iteração, um subconjunto da vizinhança da solução atual é visitada. O melhor vizinho encontrado, ou seja, aquele que possui o menor valor de função objetivo passa a ser a solução atual, mesmo que esse vizinho seja pior que a solução atual. Essa estratégia permite que o método escape de

⁴ A vizinhança de uma solução s pode ser definida como o conjunto das soluções, com exceção do próprio s , que possui uma grande quantidade de elementos comuns a s .

ótimos locais. No entanto, ela pode fazer com que o algoritmo retorne a uma solução já gerada anteriormente, fazendo com que o algoritmo entre em um ciclo.

Para evitar que isso aconteça uma lista de movimentos proibidos, denominada *lista tabu*, armazena os movimentos reversos aos últimos movimentos realizados. Essa lista funciona como uma fila de tamanho fixo, onde sempre que um novo movimento é adicionado na lista, o movimento mais antigo é retirado dela.

```

procedimento BuscaTabu( $f(\cdot)$ ,  $N(\cdot)$ ,  $A(\cdot)$ ,  $|V|$ ,  $f_{min}$ ,  $|T|$ , BTmax,  $s$ )
1    $s^* \leftarrow s$ ; /* Melhor solução até o momento */
2    $Iter \leftarrow 0$ ;
3    $MelhorIter \leftarrow 0$ ;
4    $T \leftarrow \emptyset$ ; /* Lista tabu */
5   Inicialize a função de aspiração  $A$ ;
6   enquanto  $f(s) > f_{min}$  and  $Iter - MelhorIter < BTMax$  faça
7        $Iter \leftarrow Iter + 1$ ;
8       Seja  $s' \leftarrow s \oplus m$  o melhor elemento de  $V \subseteq N(s)$  tal que
          o movimento  $m$  não seja tabu ( $m \notin T$ ) ou
           $s'$  atenda a condição de aspiração  $f(s') < A(f(s))$ ;
9       Atualize a lista tabu  $T$ ;
10       $s \leftarrow s'$ ;
11      se  $f(s) < f^*$  então
12           $s^* \leftarrow s$ ;
13           $MelhorIter \leftarrow Iter$ ;
14      fim se;
15      Atualize a função de aspiração  $A$ ;
16  fim enquanto
17  retorne  $s^*$ ;
fim BuscaTabu

```

Figura 3.4. Pseudocódigo do procedimento Busca Tabu.

Apesar de reduzir o risco de acontecerem ciclos, a utilização da lista tabu pode proibir movimentos para soluções que ainda não foram visitadas. Para amenizar esse efeito, a Busca Tabu define uma *função de aspiração*. A função de aspiração permite, sob determinadas circunstâncias, que um movimento seja realizado mesmo que ela esteja na lista tabu. Cada valor da função objetivo possui um valor de função de aspiração.

Os principais parâmetros de entrada da Busca Tabu são a cardinalidade da lista tabu $|T|$, a função de aspiração A , a cardinalidade do conjunto V que representa quantas soluções na vizinhança devem ser testadas em cada iteração e $BTmax$, o número

máximo de iterações sem melhora no valor da melhor solução. O pseudocódigo do procedimento de Busca Tabu é apresentado na Figura 3.4.

É comum que métodos de Busca Tabu incluam estratégias de intensificação e diversificação [9]. As estratégias de intensificação, em geral, têm por objetivo concentrar a busca em algumas regiões consideradas promissoras, enquanto que as estratégias de diversificação, em geral, têm por objetivo pesquisar regiões ainda pouco exploradas no espaço de soluções. Alguns métodos de Busca Tabu também incluem listas tabus dinâmicas [25, 26]. Essas listas têm o seu tamanho atualizado de acordo com o progresso da pesquisa.

3.1.3 Simulated Annealing

A metaheurística *Simulated Annealing*, proposta por Kirkpatrick, Gelatt e Vecchi [4], é um método de busca local probabilístico inspirado na operação de recozimento da termodinâmica [27].

A operação de recozimento é bastante utilizada na metalurgia para obtenção de estados de baixa energia em um sólido. Nesse processo, inicialmente, aumenta-se a temperatura do sólido até que o sólido entre em estado de fusão. Em seguida, resfria-se lentamente o material até que ele se solidifique novamente. Com um resfriamento lento e controlado, os átomos que compõem o material organizam-se numa estrutura uniforme com energia mínima. Como resultado prático tem-se uma redução nos defeitos do material.

No método *Simulated Annealing*, os movimentos são aceitos, ou seja, a solução visitada substitui a solução atual de acordo com o valor da função objetivo e de uma variável T , denominada *temperatura*. Se a solução visitada possuir um valor de função objetivo menor que a solução atual, o método aceita o movimento. Caso contrário, o movimento poderá ser aceito ou não, de acordo com uma probabilidade $e^{-\Delta f / T}$, onde Δf representa a variação do valor da função objetivo e T representa a temperatura. O parâmetro T , portanto, regula a probabilidade de aceitar movimentos de piora.

Inicialmente, a temperatura T assume um valor elevado T_0 . Após um número fixo de iterações, a temperatura é gradativamente reduzida de acordo com uma razão de resfriamento α , tal que $T_k \leftarrow \alpha \cdot T_{k-1}$, onde $\alpha \in [0,1]$. Com isso, no início, o procedimento aceita movimentos de piora com uma probabilidade alta, aumentando a

possibilidade de escapar de ótimos locais. À medida que T se aproxima de zero, o algoritmo diminui a probabilidade de aceitar movimentos de piora, comportando-se como um método de descida. O procedimento termina quando a temperatura aproxima-se de zero. Nesse momento, nenhum movimento de piora é aceito e, em consequência, o sistema é considerado estável.

Os parâmetros de controle do método são a razão de resfriamento α , a temperatura inicial T_0 e o número de iterações para cada temperatura SA_{max} . O pseudocódigo do procedimento *Simulated Annealing* é apresentado na Figura 3.5.

```

procedimento Simulated_Annealing( $f(\cdot)$ ,  $N(\cdot)$ ,  $\alpha$ ,  $SA_{max}$ ,  $T_0$ ,  $s$ )
1    $s^* \leftarrow s$ ;
2    $IterT \leftarrow 0$ ;
3    $T \leftarrow T_0$ ;
4   enquanto  $T > 0$  faça
5       enquanto ( $IterT < SA_{max}$ ) faça
6            $IterT \leftarrow IterT + 1$ ;
7           Gere um vizinho qualquer  $s' \in N(s)$ ;
8            $\Delta = f(s') - f(s)$ ;
9           se  $\Delta < 0$  então
10               $s \leftarrow s'$ ;
11              se ( $f(s') < f(s^*)$ ) então  $s^* \leftarrow s'$ ;
12          senão
13              Tome  $x \in [0,1]$ ;
14              se ( $x < e^{-\Delta/T}$ ) então  $s \leftarrow s'$ ;
15          fim se;
16      fim enquanto;
17       $T \leftarrow \alpha \cdot T$ ;
18       $IterT \leftarrow 0$ ;
19  fim enquanto;
20  retorne  $s^*$ ;
fim Simulated_Annealing;

```

Figura 3.5. Pseudocódigo do procedimento *Simulated Annealing*.

Em determinadas circunstâncias, quando a quantidade de movimentos rejeitados de forma consecutiva é alto, alguns algoritmos baseados em *Simulated Annealing* normalmente incluem uma etapa de aquecimento para minimizar esse efeito [27]. Após o aquecimento, um novo resfriamento é aplicado. Também é comum que algumas implementações trabalhem com uma taxa de resfriamento menor nas temperaturas mais altas e aumentem essa taxa quando a temperatura se reduz.

3.1.4 Algoritmos Genéticos (AG)

A metaheurística Algoritmos Genéticos, proposta por Holland [28], é um método iterativo de busca populacional inspirado na teoria da evolução das espécies de Darwin. Para uma dada população, os indivíduos mais bem adaptados ao sistema têm maiores chances de sobreviver e de produzir filhos cada vez mais adaptados, enquanto os indivíduos menos adaptados tendem a desaparecer.

A população do AG, em cada iteração (ou geração), é formada por indivíduos (ou soluções) denominados *cromossomos*. Cada cromossomo é formado por um número fixo de variáveis denominados *genes* e tem sua aptidão avaliada de acordo com uma função de aptidão (ou função objetivo).

O procedimento AG inicia sua busca com uma população geralmente aleatória de tamanho n . Em cada iteração, um mecanismo de reprodução, baseado em processos evolutivos, é aplicado sobre a população com o objetivo de explorar o espaço de busca. Esse mecanismo de reprodução utiliza operações de seleção, cruzamento e mutação.

```

procedimento AG
1    $t \leftarrow 0$ ;
2   Gere a população inicial  $P(t)$ ;
3   Avalie  $P(t)$ ;
4   enquanto Critérios de parada não satisfeitos faça
5        $t \leftarrow t + 1$ ;
6       Gere  $P(t)$  a partir de  $P(t-1)$ ;
7       Avalie  $P(t)$ ;
8       Defina a população sobrevivente;
9   fim enquanto;
fim AG;

```

Figura 3.6. Pseudocódigo do procedimento Algoritmo Genético.

A operação de seleção elege uma população intermediária a partir da população atual, que será utilizada nas operações de cruzamento e mutação. Na operação de cruzamento ou recombinação, os genes de dois cromossomos pais são combinados para gerar cromossomos filhos. Com isso, cada cromossomo filho é formado por um conjunto de genes de cada cromossomo pai. A operação de mutação altera aleatoriamente um ou mais genes de cada cromossomo de acordo com uma probabilidade definida.

Gerada a nova população, o AG define quais serão os cromossomos sobreviventes, ou seja, os n cromossomos que integrarão a próxima geração (iteração).

Nesse caso, um operador de seleção é aplicado para selecionar os cromossomos sobreviventes. Quando o critério de parada é satisfeito, o procedimento termina, retornando o melhor cromossomo como solução final do problema. O pseudocódigo do AG é apresentado na Figura 3.6.

Nas próximas subseções apresentaremos as operações de seleção, cruzamento e mutação em maiores detalhes.

3.1.4.1 Seleção

Os métodos mais comumente utilizados para selecionar cromossomos em um AG são os métodos roleta, aleatório ou misto. Esses métodos admitem a seleção de indivíduos menos aptos, o que possibilita que o AG diversifique a busca pelo espaço de soluções e escape de mínimos locais.

O método roleta seleciona os cromossomos com uma probabilidade proporcional ao seu nível de aptidão. Nesse caso, os cromossomos são representados numa roleta com percentuais proporcionais ao seu nível de aptidão. Com isso, indivíduos com alta aptidão ocupam um percentual maior da roleta, enquanto que indivíduos com aptidão mais baixa ocupam uma percentual menor da roleta.

Definido os percentuais de cada cromossomo dentro da roleta, gira-se a roleta um determinado número de vezes. Em cada giro da roleta, um cromossomo é sorteado e selecionado para as próximas etapas do algoritmo. Na Figura 3.7 é apresentado um exemplo de roleta para uma população de cinco cromossomos C1, C2, C3, C4 e C5. Observa-se que o percentual de ocupação na roleta é proporcional ao valor de aptidão de cada cromossomo.

Cromossomo	Aptidão	Aptidão Relativa
C1	2,23	0.14
C2	7,27	0.47
C3	1,05	0.07
C4	3,35	0.21
C5	1,69	0.11

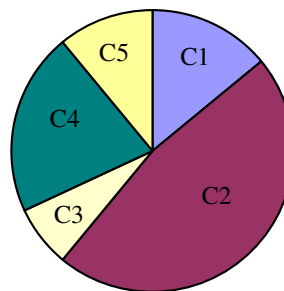


Figura 3.7. Exemplo de uma roleta para uma população de cinco cromossomos.

O método aleatório, como o próprio nome indica, seleciona cada cromossomo de forma aleatória. Por fim, o método misto combina os métodos roleta e aleatório.

3.1.4.1 Cruzamento e mutação

Na operação de cruzamento, pares de cromossomos são selecionados e cada cromossomo pai é dividido em pontos sorteados aleatoriamente, denominados pontos de corte. Novos cromossomos são então gerados permutando-se um conjunto de genes de um cromossomo pai com um conjunto de genes do outro cromossomo pai. Na Figura 3.8 é apresentado um exemplo gráfico de uma operação de recombinação de dois cromossomos com apenas um ponto de corte. Nesse caso, apenas dois cromossomos filhos diferentes podem ser gerados.

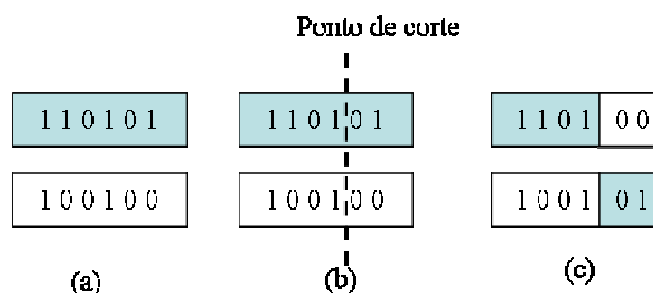


Figura 3.8. Exemplo gráfico de um procedimento de cruzamento com um ponto de corte. (a) Cromossomos pais; (b) Ponto de corte nos cromossomos pais; (c) Cromossomos filhos gerados pela recombinação genética dos cromossomos pais.

O operador de mutação altera aleatoriamente um ou mais genes de cada cromossomo. Sua utilização aumenta a diversificação da busca no espaço de soluções. A probabilidade de aplicação do operador de mutação, no entanto, é normalmente pequena.

3.2. Metaheurísticas para otimização contínua

Nesta seção descreveremos algumas metaheurísticas utilizadas para otimização global contínua. Na Seção 3.2.1 será apresentado a metaheurística *Continuous-GRASP* (C-GRASP), desenvolvido por Hirsch *et al.* [6,7] que é uma versão da metaheurística GRASP. Na Seção 3.2.2 será apresentado o método *Directed Tabu Search* (DTS),

desenvolvido por Hedar e Fukushima [9], que descreve uma versão da metaheurística Busca Tabu. Por fim, na Seção 3.2.3 será apresentada a metaheurística populacional *Particle Swarm Optimization* (PSO).

3.2.1 *Continuous-GRASP* (C-GRASP)

A versão da metaheurística GRASP adaptada à otimização contínua, denominada *Continuous-GRASP* (C-GRASP), foi uma alternativa proposta por Hirsch et al. [5-7] para resolução de problemas de otimização global contínua sujeitos a restrição nos valores limites das variáveis (*box constraints*). O C-GRASP é um método de busca local estocástico⁵, pode ser aplicado em vários tipos de problemas de otimização global contínua e não utiliza cálculos de derivada em sua busca. Esse trabalho explorará a versão do C-GRASP descrita em [6,7].

De forma análoga ao GRASP, o C-GRASP é um método de busca *multi-start*⁶ que utiliza uma fase de construção aleatória e gulosa para gerar as soluções iniciais e uma fase de busca local para tentar melhorar essas soluções iniciais. Uma característica particular do procedimento C-GRASP é que ele discretiza o domínio do problema em uma grade de pontos uniformemente distribuídos. Os procedimentos de construção e busca local realizam seus movimentos através dessa grade de pontos. À medida que o algoritmo progride, a grade de pontos vai adaptativamente se tornando mais densa.

A principal diferença para o GRASP, é que uma iteração do C-GRASP não é formada apenas por um procedimento de construção seguido por um procedimento de busca local. Ao invés disso, uma iteração do C-GRASP corresponde a uma série de ciclos construção-busca local, com a saída do procedimento de construção servindo como a entrada do procedimento de busca local e, a saída do procedimento de busca local servindo como a entrada do procedimento de construção.

O pseudocódigo descrito na Figura 3.9 ilustra o método C-GRASP.

De acordo com a Figura 3.9, podemos observar que o C-GRASP possui como parâmetros de entrada: a dimensão do problema n ; os vetores l e u que representam os limites inferiores e superiores, respectivamente; a função objetivo $f(\cdot)$; e os parâmetros

⁵ Estocásticos são padrões que surgem sobre eventos aleatórios.

⁶ Um procedimento *multi-start* é um procedimento iterativo que utiliza vários pontos de partida gerados de forma aleatória. Essa característica permite que o procedimento diversifique a busca no espaço de soluções, escapando de ótimos locais.

h_s , h_e e ρ_{lo} . Os parâmetros h_s e h_e representam a densidade inicial e final da grade de discretização, respectivamente, e ρ_{lo} representa o percentual da vizinhança da solução atual que será visitada durante a fase de busca local.

```

procedimento C-GRASP( $n, l, u, f(\cdot), h_s, h_e, \rho_{lo}$ )
1    $f^* \leftarrow \infty$ ;
2   enquanto Critério de parada não satisfeito faça
3        $x \leftarrow \text{RandomUniforme}(l, u)$ ;
4        $h \leftarrow h_s$ ;
5       enquanto  $h \geq h_e$  faça
6            $\text{Impr}_C \leftarrow \text{false}$ ;
7            $\text{Impr}_L \leftarrow \text{false}$ ;
8            $[x, \text{Impr}_C] \leftarrow \text{Construção}(x, f(\cdot), l, u, h, n, \text{Impr}_C)$ ;
9            $[x, \text{Impr}_L] \leftarrow \text{BuscaLocal}(x, f(\cdot), l, u, h, n, \rho_{lo}, \text{Impr}_L)$ ;
10          se  $f(x) < f^*$  então
11               $x^* \leftarrow x$ ;
12               $f^* \leftarrow f(x)$ ;
13          fim se;
14          se  $\text{Impr}_C = \text{false}$  and  $\text{Impr}_L = \text{false}$  então
15               $h \leftarrow h/2$ ; /* torna a grade mais densa */
16          fim se;
17      fim enquanto;
18  fim enquanto;
19  retorne  $x^*$ ;
fim C-GRASP

```

Figura 3.9. Pseudocódigo do C-GRASP.

O algoritmo processa as iterações enquanto o critério de parada não é satisfeito. Os critérios de parada podem ser baseados, por exemplo, em um número máximo de iterações, do tempo decorrido, do número de avaliações da função objetivo, etc. Quando algum dos critérios de parada é satisfeito, a melhor solução de todas as iterações é considerada a solução do problema.

Em cada iteração, inicialmente, um ponto aleatório uniformemente distribuído sobre os limites l e u é atribuído a solução atual x . O parâmetro h , que representa o tamanho atual do passo de discretização, é atualizado para o valor h_s . Em seguida, os procedimentos de construção e busca local são chamados sequencialmente, em ciclo, partindo da solução atual x . Após cada ciclo construção-busca local realiza-se uma comparação entre $f(x)$, o valor da função objetivo da solução x , e $f(x^*)$, o valor da função objetivo da melhor solução encontrada até o momento x^* . Se a solução atual x

for melhor que x^* (ou seja, $f(x) < f(x^*)$) então os valores de x^* e $f(x^*)$ são atualizados para os valores de x e $f(x)$, respectivamente.

Além de retornar a melhor solução visitada, os procedimentos de construção e busca local retornam as variáveis de estado Impr_C e Impr_L , respectivamente. Elas indicam se o procedimento conseguiu gerar uma solução melhor que sua solução de entrada. Se o procedimento conseguir gerar essa solução, a variável passa para o estado *true*, caso contrário ele fica no estado *false*. Se em um determinado ciclo construção-busca local, nenhum dos dois procedimentos melhorarem a solução de entrada, ou seja, Impr_C e Impr_L retornaram no estado *false*, o C-GRASP aumenta a densidade da grade de discretização atual. Aumenta-se essa densidade, reduzindo-se o valor do parâmetro h para metade do seu valor ($h \leftarrow h/2$). Uma iteração do C-GRASP termina quando $h < h_e$.

3.2.1.1 Procedimento de Construção

O procedimento de construção inicia permitindo que todas as coordenadas de x possam mudar de valor, criando um conjunto de coordenadas denominadas coordenadas *não-fixas*. A cada iteração, é efetuada uma busca linear na direção de cada coordenada *não-fixa* de x com as outras $(n-1)$ coordenadas mantidas com os seus valores atuais. Uma Lista de Candidatos Restrita (LCR) com tamanho determinado por $\alpha \in [0,1]$ é criada apenas com as coordenadas *não-fixas*. Um elemento da LCR é então escolhido aleatoriamente e essa coordenada é retirada do conjunto de coordenadas *não-fixas*. O procedimento é repetido até que o conjunto de coordenadas *não-fixas* esteja vazio. O pseudocódigo do procedimento de construção do C-GRASP é apresentado na Figura 3.10.

3.2.1.2 Procedimento de Busca Local

O procedimento de busca local do C-GRASP pode ser visto como uma aproximação da regra do gradiente da função objetivo. Para um dado ponto $x \in S \subset \mathfrak{R}^n$, o algoritmo gera uma vizinhança e determina que soluções na vizinhança, se existirem, são melhores que a solução atual. Se uma solução melhor for encontrada,

ela torna-se a solução atual e a busca continua a partir dessa solução. O pseudocódigo da busca local do C-GRASP é apresentado na Figura 3.11.

```

procedimento Construção ( $x, f(\cdot), l, u, h, n, Impr_C$ )
1   $n\grave{a}o\_fixas \leftarrow [1, 2, \dots, n];$ 
2   $\alpha \leftarrow \text{RandomUniforme}(0, 1);$ 
3   $Reuse \leftarrow \text{false};$ 
4  enquanto  $n\grave{a}o\_fixas \neq \emptyset$  faça
5       $gMin \leftarrow +\infty;$ 
6       $gMax \leftarrow -\infty;$ 
7      para  $i = 1, \dots, n$  faça
8          se  $i \in n\grave{a}o\_fixas$  então
9              se  $Reuse = \text{false}$  então
10                  $z_i \leftarrow \text{BuscaLinear}(x, f(\cdot), l, u, h, i);$ 
11                  $g_i \leftarrow f(x^i);$ 
12             fim se
13             se  $gMin > g_i$  então  $gMin \leftarrow g_i;$ 
14             se  $gMax < g_i$  então  $gMax \leftarrow g_i;$ 
15         fim se
16     fim para
17      $LCR \leftarrow \emptyset;$ 
18      $\text{Threshold} \leftarrow gMin + \alpha (gMax - gMin);$ 
19     para  $i = 1, \dots, n$  faça
20         se  $i \in n\grave{a}o\_fixas$  and  $g_i \leq \text{Threshold}$  então
21              $LCR \leftarrow LCR \cup \{i\};$ 
22         fim se
23     fim para
24      $j \leftarrow \text{Selecione aleatoriamente um elemento da LCR};$ 
25     se  $x_j = z_j$  então
26          $Reuse \leftarrow \text{true};$ 
27     senão
28          $x_j \leftarrow z_j;$ 
29          $Reuse \leftarrow \text{false};$ 
30          $Impr_C \leftarrow \text{true};$ 
31     fim se
32      $n\grave{a}o\_fixas \leftarrow n\grave{a}o\_fixas \setminus \{j\};$ 
33 fim enquanto;
34 retorne ( $x, Impr_C$ );
fim Construção

```

Figura 3.10. Pseudocódigo do procedimento de construção do C-GRASP.

Para definir a vizinhança no C-GRASP, seja $\bar{x} \in \mathfrak{R}^n$ a solução atual e h o parâmetro de discretização da grade de pontos. Definimos:

$$S_h(\bar{x}) = \{x \in S \mid l \leq x \leq u, x = \bar{x} + \tau \cdot h, \tau \in \mathbb{Z}^n\}$$

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

O conjunto $S_h(\bar{x})$ corresponde ao conjunto de pontos em S que são passos inteiros de comprimento h , a partir de x . O conjunto $B_h(\bar{x})$ é o conjunto formado pela

projeção dos pontos que pertencem a $S_h(\bar{x}) \setminus \{\bar{x}\}$ numa hiper-esfera de raio h centrada em \bar{x} .

```

procedimento BuscaLocal ( $x, f(\cdot), l, u, h, n, \rho_{io}, Impr_C$ )
1    $x^* \leftarrow x;$ 
2    $f^* \leftarrow f(x);$ 
3    $NumPontosGrade \leftarrow \prod_{i=1}^n \lceil (u_i - l_i) / h \rceil;$ 
4    $NumPontosParaExaminar \leftarrow \lceil \rho_{io} \cdot NumPontosGrade \rceil$ 
5    $NumPontosExaminados \leftarrow 0;$ 
6   enquanto  $NumPontosExaminados \leq NumPontosParaExaminar$  faça
7        $NumPontosExaminados \leftarrow NumPontosExaminados + 1;$ 
8        $x \leftarrow \text{SelecionaAleatoriamente}(B_h(x^*));$ 
9       se  $l \leq x \leq u$  and  $f(x) < f^*$  então
10           $x^* \leftarrow x;$ 
11           $f^* \leftarrow f(x);$ 
12           $Impr_L \leftarrow \text{true};$ 
13           $NumPontosExaminados \leftarrow 0;$ 
14       fim se
15   fim enquanto;
16   retorne ( $x, Impr_L$ );
fim Busca Local
  
```

Figura 3.11. Pseudocódigo do procedimento de busca local do C-GRASP.

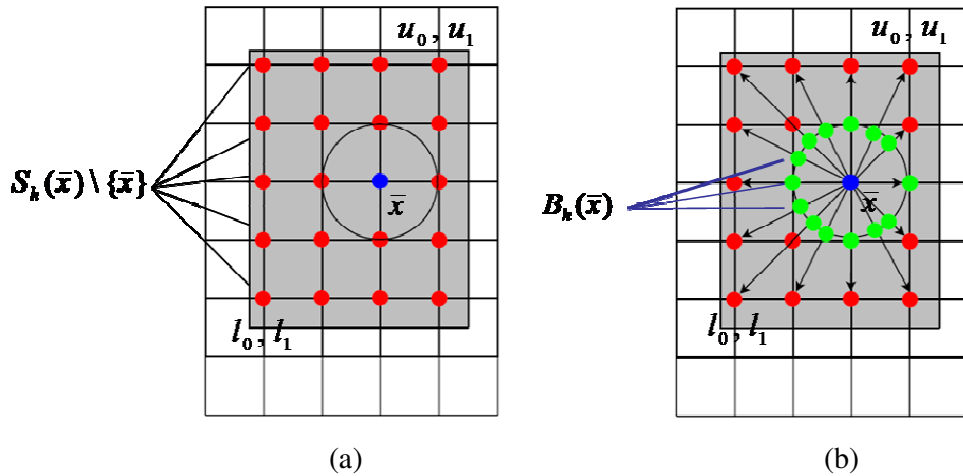


Figura 3.12. Busca Local do C-GRASP. (a) Representação do conjunto $S_h(\bar{x}) \setminus \{\bar{x}\}$. (b) Representação do conjunto $B_h(\bar{x})$. O ponto azul representa a solução atual \bar{x} , a área cinza representa os limites l e u e os pontos vermelhos representam os pontos do conjunto $S_h(\bar{x}) \setminus \{\bar{x}\}$. Os pontos verdes, (b), representam os pontos do conjunto $B_h(\bar{x})$, ou seja, as projeções dos $S_h(\bar{x}) \setminus \{\bar{x}\}$ na esfera de raio h centrada em \bar{x} .

A vizinhança da busca local do C-GRASP pode ser definida como o conjunto de pontos em $B_h(\bar{x})$. Na Figura 3.12 é ilustrado um exemplo gráfico dessa vizinhança representando os conjuntos $S_h(\bar{x}) \setminus \{\bar{x}\}$ e $B_h(\bar{x})$.

3.2.2 Directed Tabu Search (DTS)

Atualmente, na literatura, existem poucos algoritmos Busca Tabu adaptados para otimização contínua. Uma das contribuições mais recentes e relevantes é o algoritmo *Directed Tabu Search* (DTS). O DTS, desenvolvido por Hedar e Fukushima [9], é um algoritmo que utiliza de forma híbrida, a Busca Tabu e métodos de busca direcionada (*Direct Search Methods*). O papel dos métodos de busca direcionada é estabilizar a busca na vizinhança de um ótimo local. Mais especificamente, ao invés de utilizar métodos de busca aleatória na geração de movimentos na vizinhança, estratégias de busca direcionada são utilizadas para realizar esses movimentos.

O DTS é um método iterativo que utiliza três procedimentos de busca: Exploração, Diversificação e Intensificação. Além disso, ele introduz novos elementos de memória para evitar ciclagens, como as regiões tabu (*Tabu Regions* - TR), regiões semi-tabu (semi-TRs) e a lista tabu multi-ranqueada (*multi-ranked Tabu List* - TL). Outra estrutura de memória, denominada lista de regiões visitadas (*Visited Regions List* - VRL), é também introduzida. O objetivo da VRL é diversificar a busca em áreas não visitadas do espaço de soluções.

No procedimento de exploração, estratégias de busca local são utilizadas para explorar o espaço de soluções. Elementos de memória como a TL, TR e semi-TR são utilizados para evitar que soluções já visitadas recentemente sejam revisitadas ou fiquem presas em um mínimo local. O procedimento de Diversificação é utilizado para diversificar a busca em regiões do espaço de soluções ainda não visitadas pelo procedimento de exploração. A VRL é utilizada para gerenciar o procedimento de diversificação. Por fim, para refinar as melhores soluções visitadas até o momento, uma fase de intensificação é aplicada. A estrutura principal do método DTS é apresentada na Figuras 3.13.

De acordo com as Figura 3.13, o algoritmo DTS funciona da seguinte forma: o laço principal do DTS é composto pelos procedimentos de exploração e diversificação. Esses procedimentos são repetidos até que as condições de parada sejam satisfeitas. Na

fase final, o procedimento de intensificação é utilizado para refinar as melhores soluções.

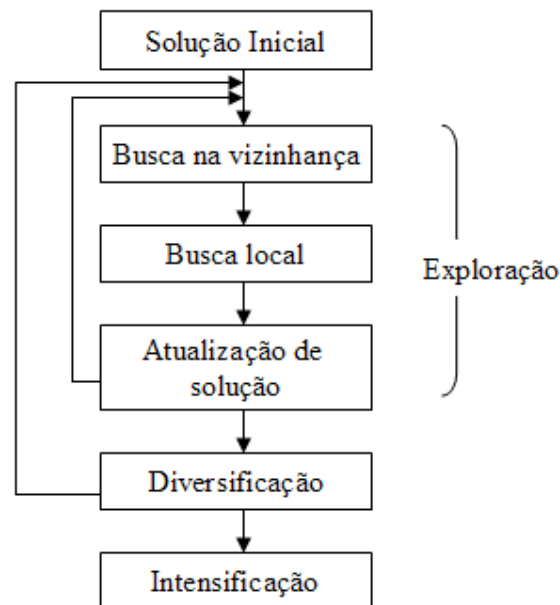


Figura 3.13. Estrutura do método DTS.

3.2.2.1 Elementos de memória

Nessa seção serão descritos alguns novos conceitos e implementações de elementos de memória do algoritmo DTS. O primeiro elemento de memória a ser apresentado é a lista tabu multi-ranqueada (TL) que armazena um conjunto de soluções visitadas. Os pontos na lista tabu são ranqueados e armazenados de acordo com o momento de visitação (*recency*) e de acordo com o valor da função objetivo. Com isso, algumas posições na lista tabu são utilizadas para armazenar as melhores soluções visitadas, o que auxilia o procedimento de intensificação.

Sobre cada solução armazenada na lista tabu, dois tipos de regiões são especificados no espaço de busca. A primeira é a região tabu (TR). Nenhum ponto que pertence à região tabu pode ser visitado. A outra região é a região semi-tabu (semi-TR), que corresponde a uma região ao redor da região tabu. Quando um ponto pertence a uma região semi-tabu, um procedimento especial é aplicado para evitar que os pontos de busca pertençam a alguma região tabu.

Outro elemento de memória introduzido é a lista de regiões visitadas (VRL). O centro das regiões visitadas e a frequência de visitação dessas regiões são armazenados na VRL, a fim de direcionar um procedimento de diversificação. As informações da VRL são, portanto, utilizadas com o objetivo de evitar a busca em regiões muito visitadas.

3.2.2.2 Procedimento de Exploração

O procedimento de exploração é responsável por varrer a região ao redor da solução atual. Para realizar essa tarefa o procedimento de exploração utiliza estratégias de busca local baseados em métodos de busca direcionada.

A cada iteração do procedimento de exploração, inicialmente, p pontos $\{y_i\}_{i=1}^p$ são gerados na vizinhança da solução atual x . Esse procedimento é denominado *BuscaVizinhança*. Em seguida, tenta-se melhorar os pontos de vizinhança $\{y_i\}_{i=1}^p$ através da execução de outro procedimento de busca, denominado *Busca local*. A busca local gera q pontos $\{y_{p+i}\}_{i=1}^q$ que são denominados pontos locais. Por fim, o melhor desses pontos é selecionado, a lista tabu e a lista de regiões visitadas são atualizadas. Na Figura 3.14 é ilustrado o pseudocódigo do procedimento de exploração.

```

procedimento Exploração ( $n, l, u, f(\cdot), \ell_{\text{inner}}, \ell'_{\text{inner}}, \text{IterSemMelhoras}_{\text{main}}$ )
1    $\text{IterSemMelhoras}_{\text{inner}} \leftarrow 0$ ;
2   enquanto  $k < \ell_{\text{inner}}$  or  $\text{IterSemMelhoras}_{\text{inner}} < \ell'_{\text{inner}}$  faça
3        $j \leftarrow j + 1$ ;
4        $\text{IterSemMelhoras}_{\text{inner}} \leftarrow \text{IterSemMelhoras}_{\text{inner}} + 1$ ;
5        $[y_1..y_p, d] \leftarrow \text{BuscaVizinhança}(x, f(\cdot), l, u, n)$ ;
6        $[y_{p+1}..y_{p+q}] \leftarrow \text{BuscaLocal}(y_1..y_p, f(\cdot), d, l, u, n)$ ;
7        $x \leftarrow \text{Melhor solução dos } \{y_i\}_{i=1}^{p+q}$ ;
8        $\text{AtualizaSolução}(x, f(\cdot), TL, VRL)$ ;
9       se  $f(x) < f^*$  então
10           $x^* \leftarrow x$ ;
11           $f^* \leftarrow f(x)$ ;
12           $\text{IterSemMelhoras}_{\text{inner}} \leftarrow 0$ ;
13           $\text{IterSemMelhoras}_{\text{main}} \leftarrow 0$ ;
14       fim se
15   fim enquanto
15   retorne  $x^*$ ;
fim Exploração

```

Figura 3.14. Pseudocódigo do procedimento de exploração.

O DTS utiliza duas estratégias de busca direcionada: a Busca Nelder-Mead (Nelder-Mead *Search* - NMS) e um método de busca por padrões, a Busca de Padrões Adaptativos (*Adaptive Pattern Search* - APS). Essas estratégias geram dois tipos de pontos: os pontos de vizinhança (*neighborhood trial points*) e os pontos locais (*local trial points*).

3.2.2.3 Procedimento de Diversificação

O procedimento de diversificação é responsável por explorar regiões ainda não visitadas no espaço de soluções. Esse procedimento utiliza a informação armazenada na lista de regiões visitadas (VRL). O pseudocódigo do procedimento de diversificação é apresentado na Figura 3.15.

```

procedimento Diversificação ( $n, l, u, TL, VRL, j$ )
1    $NumIters \leftarrow 0$ ;
2   repita
3        $x \leftarrow \text{RandomUniforme}(l, u)$ ;
4        $NumIters \leftarrow NumIters + 1$ ;
5   enquanto Região vizinha a  $x$  muito visitada na VRL or
        $NumIters > NumItersMax$ ;
6   Atualiza  $TL$  e  $VRL$ ;
7   retorne  $x$ ;
fim Diversificação

```

Figura 3.15. Pseudocódigo do procedimento de Diversificação.

3.2.2.4 Procedimento de Intensificação

Conforme já comentado na Seção 3.2.2.1, a lista tabu multi-ranqueada (TL) reserva algumas posições para as melhores soluções. Com o objetivo de refinar esses pontos, o método DTS aplica outro método de busca local partindo desses pontos, denominado Procedimento de Intensificação.

3.2.3 Particle Swarm Optimization (PSO)

A metaheurística *Particle Swarm Optimization* (PSO), proposta por Eberhart e Kennedy [29], é um método de otimização contínua inspirado no comportamento social

de um grupo de partículas, pássaros ou peixes.

O PSO é um método de busca populacional e estocástico. Além disso, ele possui muitas características semelhantes aos métodos de computação evolucionária, como os Algoritmos Genéticos (AGs). A população inicial do PSO é formada por um conjunto de soluções aleatórias. Além disso, no procedimento de busca as gerações são atualizadas em cada iteração.

Contudo, diferentemente, dos algoritmos genéticos, PSO não possui operadores de cruzamento e mutação. No PSO, as partículas (soluções) se movimentam pelo espaço de soluções seguindo as melhores partículas.

Para melhor compreensão do PSO, suponha o seguinte cenário: um grupo de pássaros está disposto aleatoriamente procurando alimento em uma determinada área. Nessa área o alimento está localizado em um único ponto e os pássaros não sabem onde ela se localiza. A única coisa que os pássaros sabem é quão distante o alimento está em cada iteração.

Considerando esse cenário, a melhor estratégia para se aproximar da comida é seguir o pássaro que está mais próximo dela. No PSO cada solução é um pássaro, também denominado partícula. Todas as partículas possuem valores de aptidão que são avaliados de acordo com uma função objetivo. As partículas se movimentam (voam) pelo espaço de soluções seguindo as atuais melhores partículas.

O PSO é inicializado com um grupo de partículas (soluções) aleatórias. Em cada iteração, cada partícula é atualizada de acordo com dois “melhores” valores. O primeiro valor, denominado *pbest*, é a melhor aptidão (solução) alcançada até o momento pela partícula. O segundo valor, denominado *gbest*, é a melhor aptidão obtida por qualquer partícula da população até o momento. Quando uma partícula pertence a uma população de vizinhos topológicos, o melhor valor local é denominado *lbest*.

Após encontrar os valores *pbest* e *gbest*, a partícula atualiza sua velocidade e posicionamento de acordo com as seguintes equações:

$$v = v + c_1 * rand() * (pbest - x) + c_2 * rand() * (gbest - x)$$

$$x = x + v$$

Nesse caso, v é a velocidade da partícula, x é a posição da partícula atual, $rand()$ é uma função de geração de números aleatórios no intervalo (0,1) e c_1 e c_2 são os fatores de aprendizagem. Usualmente $c_1 = c_2 = 2$.

O pseudocódigo do procedimento PSO é apresentado na Figura 3.16.

```

procedimento PSO
1   Inicialize o conjunto de partículas  $P(t)$ ;
2   repita
3       para Cada partícula  $p \in P(t)$  faça
4           Calcule valor aptidão;
5           se ( $f(p) < f(pBest)$ ) então
6                $pBest \leftarrow p$ ;
7           fim se;
8       fim para;
9        $gBest \leftarrow$  Valor de aptidão da melhor de todas as partículas;
10      para Cada partícula  $p \in P(t)$  faça
11          Calcule velocidade da partícula  $p$ ;
12          Atualize posição da partícula  $p$ ;
13      fim para;
14  enquanto Critério de parada não satisfeito;
fim PSO;

```

Figura 3.16. Pseudocódigo do PSO.

4. MÉTODOS PROPOSTOS

O C-GRASP, de forma similar as outras metaheurísticas, é um método de busca que consome pouco tempo computacional e possui a capacidade de escapar de mínimos locais. Contudo, conforme discutido no Capítulo 1, por utilizar construções aleatórias, o C-GRASP tem dificuldades em calcular direções de busca promissoras na vizinhança do mínimo local. Em consequência disso, o C-GRASP possui uma convergência lenta.

Com o objetivo de acelerar a convergência do C-GRASP e tornar esse método mais eficiente, esse trabalho propõe a hibridização do C-GRASP com métodos de busca local mais eficientes. Mais especificamente, dois métodos são propostos. O primeiro método, denominado *Enhanced Continuous-GRASP* (EC-GRASP), é uma hibridização do método C-GRASP, proposto em [6,7] com o método *Adaptive Pattern Search* (APS), um método de busca direcionada. O EC-CGRASP será apresentado na Seção 4.1.

O segundo método, denominado BFGS *Continuous-GRASP* (BC-GRASP), é uma extensão do método EC-GRASP. Trata-se de uma hibridização do método EC-GRASP com o método exato LBFGS (*Limited Memory BFGS*) [22,23]. O BC-GRASP será apresentado na Seção 4.2.

4.1 Método EC-GRASP

O EC-GRASP é uma combinação da metaheurística C-GRASP com um método de busca local denominado *Adaptive Pattern Search* (APS). Com essa hibridização, pretende-se projetar um método mais eficiente que o C-GRASP puro. O EC-GRASP é um método simples e não utiliza cálculos de derivada de primeira e segunda ordem em sua estrutura.

De forma similar ao C-GRASP, o EC-GRASP é um metaheurística utilizada para resolução de problemas de otimização global contínua sujeito a restrições nos limites das variáveis (*box constraints*). Sem perda de generalidade, definimos o domínio S como sendo um hiper-retângulo $S = \{x \in \mathbb{R}^n : l \leq x \leq u\}$, onde $l \in \mathbb{R}^n$, $u \in \mathbb{R}^n$ e $u_i \geq l_i$ para $i = 1, \dots, n$. O problema considerado, portanto, é:

Minimize $f(x)$,

Sujeito a $l \leq x \leq u$

Onde $f : \mathfrak{R}^n \rightarrow \mathfrak{R}$ e $l, x, u \in \mathfrak{R}^n$.

O EC-GRASP possui uma estrutura similar ao C-GRASP. Ele é um método *multi-start* composto por uma fase de construção gulosa e adaptativa e por uma fase de busca local. A melhor solução de todas as iterações é a solução final do método. Cada iteração do método é formada por um conjunto de ciclos construção-busca local. O domínio é discretizado em uma grade de pontos uniformemente distribuídos que vai adaptativamente se tornando mais densa à medida que o algoritmo progride.

O EC-GRASP utiliza o mesmo procedimento de construção do C-GRASP descrito na Seção 3.2.1.1. Contudo, um novo procedimento de busca local é definido, o *BuscaLocal_APS*. Esse procedimento de busca local é inspirado no método de busca direcionada *Adaptive Pattern Search* (APS), um método de Busca por Padrões. O pseudocódigo do procedimento EC-GRASP é apresentado na Figura 4.1. Na subseção 4.1.1 apresentaremos o método APS e na subseção 4.1.2 apresentaremos maiores detalhes sobre o método *BuscaLocal_APS*.

```

procedimento EC-GRASP( $n, l, u, f(\cdot), h_s, h_e, MaxIters$ )
1    $f^* \leftarrow \infty$ ;
2   enquanto Critério de parada não satisfeito faça
3        $x \leftarrow \text{RandomUniforme}(l, u)$ ;
4        $h \leftarrow h_s$ ;
5       enquanto  $h \geq h_e$  faça
6            $\text{Impr}_C \leftarrow \text{false}$ ;
7            $\text{Impr}_L \leftarrow \text{false}$ ;
8            $[x, \text{Impr}_C] \leftarrow \text{Construção}(x, f(\cdot), l, u, h, n, \text{Impr}_C)$ ;
9            $[x, \text{Impr}_L] \leftarrow \text{BuscaLocal\_APS}(x, f(\cdot), l, u, h, n, h_e, MaxIters, \text{Impr}_L)$ ;
10          se  $f(x) < f^*$  então
11               $x^* \leftarrow x$ ;
12               $f^* \leftarrow f(x)$ ;
13          fim se;
14          se  $\text{Impr}_C = \text{false}$  and  $\text{Impr}_L = \text{false}$  então
15               $h \leftarrow h/2$ ; /* torna a grade mais densa */
16          fim se;
17      fim enquanto;
18  fim enquanto;
19  retorne  $x^*$ ;
fim EC-GRASP

```

Figura 4.1. Pseudocódigo do EC-GRASP.

Os parâmetros de entrada do EC-GRASP são n , l , u , $f(\cdot)$, h_s e h_e e $MaxIters$. O parâmetro n representa a dimensão do problema; l e u representam os limites inferior e superior, respectivamente; $f(\cdot)$ representa a função objetivo; h_s e h_e representam a densidade inicial e final da grade de discretização, respectivamente. O parâmetro $MaxIters$ é um parâmetro específico do procedimento EC-GRASP e representa quantas iterações sem melhora o procedimento de busca local deve explorar.

4.1.1 Adaptive Pattern Search (APS)

A idéia principal do método *Adaptive Pattern Search* (APS) proposto por Hedar e Fukushima [9] é baseada no método direção de descida aproximada (*Approximate Descent Direction* - ADD) [10]. O método ADD é um procedimento livre de derivadas com uma grande habilidade de produzir direções de descidas utilizando pontos ao redor da solução atual.

Mais especificamente, o APS constrói n direções padrões paralelas aos eixos coordenados, partindo de uma solução atual x , e gera um conjunto $V = \{y_i\}_{i=1}^n$ de n pontos ao longo dessas direções com um determinado tamanho do passo. Uma direção adaptativa d é então computada utilizando esses n pontos, como segue.

$$d = \sum_{i=1}^n \omega_i u_i, \text{ onde,} \quad (4.1)$$

$$\omega_i = \frac{\Delta f_i}{\sum_{j=1}^n |\Delta f_j|}, \quad i = 1, 2, \dots, n, \quad (4.2)$$

$$u_i = -\frac{(y_i - x)}{\|(y_i - x)\|}, \quad i = 1, 2, \dots, n, \quad (4.3)$$

$$\Delta f_i = f(y_i) - f(x), \quad i = 1, 2, \dots, n \quad (4.4)$$

Na versão original do APS proposta em [9], após calcular a direção d , dois pontos y_{n+1} e y_{n+2} são gerados com tamanhos de passos diferentes. Esses pontos são utilizados com o objetivo de explorar a direção promissora d . O melhor dos $n+2$ pontos gerados é retornado como solução do procedimento APS.

O EC-GRASP, contudo, implementa algumas alterações no método APS com relação a exploração da direção de descida d . No EC-GRASP, após calcular a direção

de descida d , aplica-se um método de busca linear na direção d com comprimento inicial do intervalo h (o tamanho do passo de discretização atual do EC-GRASP) e comprimento final do intervalo h_e (tamanho do passo de discretização final do EC-GRASP). A solução retornada da busca linear na direção d é atribuída ao ponto y_{n+1} . O pseudocódigo do procedimento APS do EC-GRASP é apresentado na Figura 4.2.

```

procedimento APS ( $x, f(\cdot), n, h, h_e$ )
1   Gera conjunto  $V = \{y_i\}_{i=1}^n$ , a partir de  $x$  com tamanho de passo  $h$ ;
2   para  $i = 1, \dots, n$  faça
3       se  $f(y_i) < f(x)$  então
4           retorne  $y_i$ ;
5       fim se;
6   fim para;
7   Calcule direção  $d$  usando a equação (4.1);
8    $y_{n+1} \leftarrow \text{BuscaLinear}(x, f(\cdot), d, h, h_e)$ ;
9    $V \leftarrow V \cup \{y_{n+1}\}$ ;
10  retorne melhor elemento de  $V$ ;
fim APS

```

Figura 4.2. Pseudocódigo do procedimento APS.

De acordo com a Figura 4.2, os parâmetros de entrada do procedimento APS são a solução inicial x ; a função objetivo $f(\cdot)$; a dimensão do problema n ; os parâmetros h e h_e representam o comprimento inicial e final do intervalo da busca linear que será realizada na direção d , respectivamente.

O procedimento APS, na linha 1, visita um conjunto de n soluções $V = \{y_i\}_{i=1}^n$ na vizinhança da solução atual x . As soluções do conjunto V são construídas com direções paralelas aos eixos coordenados e com tamanho de passo h . Se alguma solução do conjunto V for melhor que a solução inicial x (linha 3), essa solução é retornada pelo procedimento (linha 4) e o procedimento termina. Caso contrário, na linha 7, calcula-se a direção de descida aproximada d utilizando a equação (4.1).

Em seguida, na linha 8, aplica-se uma busca linear na direção d com comprimento inicial do intervalo h e comprimento final h_e . A melhor solução retornada pela busca linear é atribuída ao ponto y_{n+1} . Na linha 9, adiciona-se a solução y_{n+1} ao conjunto V . Por fim, na linha 10, retorna-se o melhor elemento do conjunto V , ou seja, o melhor das $n+1$ soluções visitadas.

No exemplo apresentado na Figura 4.3, de duas dimensões ($n=2$), os dois pontos de vizinhança y_1 e y_2 são gerados na vizinhança da solução atual x , conforme Figura 4.3

(a). Uma direção de descida aproximada d é computada utilizando os pontos y_1 e y_2 . Se assumirmos que x é melhor que y_1 e y_2 , então o vetor d é composto em direção aos vetores $x - y_1$ e $x - y_2$ com pesos inversamente proporcionais a $|f(x) - f(y_1)|$ e $|f(x) - f(y_2)|$, conforme Figura 4.3 (b). Finalmente, aplica-se um método de busca linear sobre a direção d (Figura 4.3 (c)), gerando o ponto y_3 (Figura 4.3 (d)). O melhor ponto gerado (y_1 , y_2 ou y_3) será retornado pelo procedimento APS do EC-GRASP.

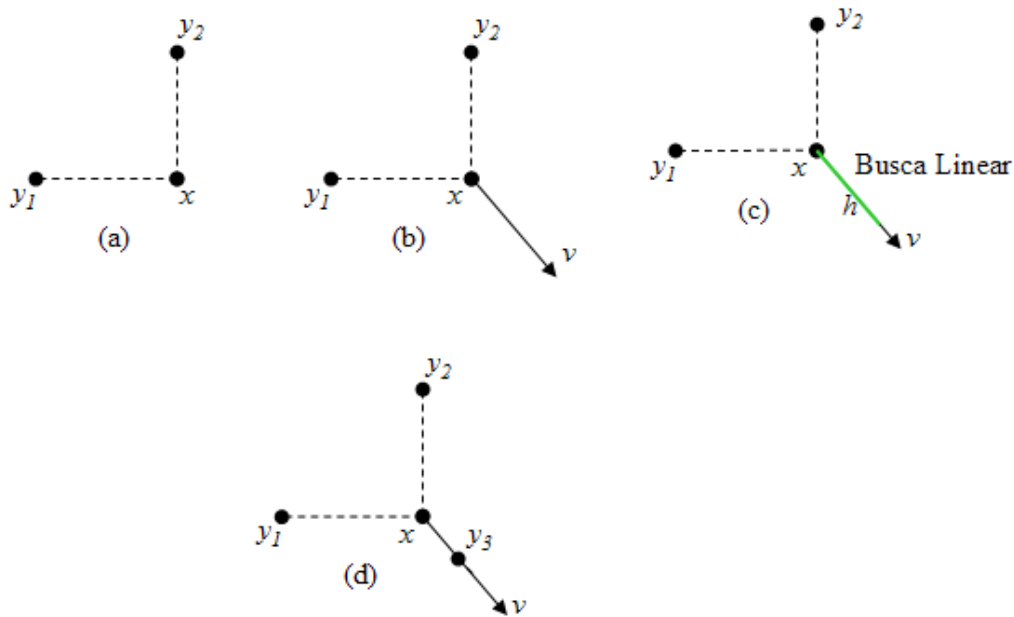


Figura 4.3. Estratégia APS em duas dimensões.

4.1.2 Procedimento de Busca Local

O procedimento de busca local do EC-GRASP, o *BuscaLocal_APS*, é um método iterativo de busca local que não utiliza cálculos de derivada e explora a vizinhança da solução atual aceitando apenas soluções melhores que ela. A vizinhança do procedimento de busca local é baseada no método de busca direcionada *Adaptive Pattern Search* (APS), descrito na Seção 4.1.1 e no conjunto $B_h(x)$, descrito na Seção 3.2.1.2. Nesse contexto, elementos do conjunto $B_h(x)$ são selecionados aleatoriamente para explorar a vizinhança da solução atual, enquanto o método APS tenta direcionar a busca nessa vizinhança para um mínimo local.

O procedimento *BuscaLocal_APS*, em cada iteração, calcula uma direção de

descida aproximada d , a partir da solução atual x e realiza uma busca linear sobre essa direção (método APS). Se a solução retornada pelo APS não for melhor que a solução x^* , a melhor solução visitada até o momento, o BuscaLocal_APS atualiza a solução corrente x com uma solução escolhida aleatoriamente do conjunto $B_h(x^*)$. O procedimento termina quando um determinado número de tentativas sem melhorias na solução x^* é extrapolado. O procedimento, então, retorna a solução x^* , a melhor solução explorada no procedimento de busca. O pseudocódigo do procedimento de busca local é apresentado na Figura 4.4.

```

procedimento BuscaLocal_APS ( $x, f(\cdot), l, u, h, n, h_e, MaxIters, Impr_L$ )
1    $x^* \leftarrow x$ ;
2    $f^* \leftarrow f(x)$ ;
3    $NumIters \leftarrow 0$ ;
4   enquanto  $NumIters \leq MaxIters$  faça
5        $NumIters \leftarrow NumIters + 1$ ;
6        $x \leftarrow APS(x, f(\cdot), n, h, h_e)$ ;
7       se  $l \leq x \leq u$  and  $f(x) < f^*$  então
8            $x^* \leftarrow x$ ;
9            $f^* \leftarrow f(x)$ ;
10           $Impr_L \leftarrow \text{true}$ ;
11           $NumIters \leftarrow 0$ ;
12      senão
13           $x \leftarrow \text{SelecioneAleatoriamente}(B_h(x^*))$ ;
14      fim se;
15  fim enquanto;
16  retorne ( $x, Impr_L$ );
fim BuscaLocal APS

```

Figura 4.4. Pseudocódigo do procedimento BuscaLocal_APS.

Os parâmetros de entrada do procedimento de busca local são: a solução de entrada x ; a função objetivo $f(\cdot)$; o tamanho do passo de discretização atual h ; a dimensão do problema n ; e os parâmetros h_e , $MaxIters$ e $Impr_L$. O parâmetro h_e representa o comprimento final do intervalo da busca linear do método APS. O parâmetro $MaxIters$ representa o número máximo de iterações sem melhoria na solução x^* . Quando esse valor é atingido o procedimento termina. O parâmetro $Impr_L$ indica se o procedimento encontrou ou não uma solução melhor que solução de entrada x .

Nas linhas 1 e 2, as variáveis x^* e f^* são inicializadas com os valores x e $f(x)$, respectivamente. As variáveis x^* e f^* representam a melhor solução visitada e o valor da função objetivo dessa solução, respectivamente. Na linha 3, a variável $NumIters$ é

inicializada com o valor zero. No laço das linhas 4-15, o procedimento de busca é executado enquanto a variável *NumIters* possui valor menor que o parâmetro *MaxIters*. Na linha 6, é invocado o método APS a partir da solução x , com os intervalos da busca linear, h e h_e .

Se a solução retornada pelo procedimento APS for melhor que x^* (linha 7), então as variáveis x^* e f^* são atualizadas com os valores x e $f(x)$, respectivamente (linhas 8 e 9), a variável *Impr_L* recebe o valor *true* (linha 10) e o parâmetro *NumIters* recebe o valor zero (linha 11). A variável *Impr_L* recebe o valor *true* indicando que o procedimento encontrou uma solução melhor que a solução de entrada. O parâmetro *NumIters* recebe o valor zero, indicando que o número de iterações sem melhorias, a partir da solução x^* (que acabou de ser atualizada), é zero.

Caso contrário (ou seja, se a solução retornada pelo APS não é melhor que x^*), na linha 13 atualiza-se a solução x com uma solução aleatoriamente escolhida do conjunto $B_h(x^*)$. Essa operação permitirá que o APS explore um conjunto diferente de soluções vizinhas na próxima iteração do laço das linhas 4-15.

4.2 Método BC-GRASP

O método BC-GRASP é uma combinação do método EC-GRASP apresentado na Seção 4.1 com o método exato BFGS com memória limitada (*Limited Memory BFGS* - LBFGS), apresentado na Seção 2.2.2.2.4. Com essa hibridização, pretende-se projetar um método de convergência mais eficiente que os métodos C-GRASP e EC-GRASP, uma vez que essa abordagem utiliza dois métodos de convergência local diferentes: o método de busca direcionada APS e o método quase-Newton LBFGS.

Em síntese, o método BC-GRASP estende o método EC-GRASP procurando acelerar a sua convergência com etapas de refinamento ou intensificação baseadas no método LBFGS, um método de busca multidimensional com derivadas. O método BC-GRASP é, portanto, um método de busca iterativo que utiliza cálculos de derivadas da função objetivo.

De acordo com Hirsch *et al.* [5], alguns dos problemas dos métodos baseados em derivadas é que a função pode não ser derivável em todo o domínio ou o cálculo de derivadas de primeira e segunda ordem pode consumir muito tempo computacional. Contudo, como o LBFGS é utilizado apenas como uma etapa de refinamento do BC-

GRASP e o método APS (Seção 4.1.1) é também utilizado como método de busca local, o BC-GRASP é suficientemente robusto para contornar esses problemas.

De forma similar ao EC-GRASP, o problema considerado no BC-GRASP é:

$$\begin{aligned} &\text{Minimize } f(x), \\ &\text{Sujeito a } l \leq x \leq u, \\ &\text{onde } f : \Re^n \rightarrow \Re \text{ e } l, x, u \in \Re^n. \end{aligned}$$

Os parâmetros de entrada do BC-GRASP são n , l , u , $f(\cdot)$, h_s e h_e , $MaxIters$ e m . Os parâmetros n , l , u , $f(\cdot)$, h_s e h_e e $MaxIters$ são os mesmos parâmetros de entrada do EC-GRASP descritos na Seção 4.1. O parâmetro m é um parâmetro de configuração do LBFGS. Conforme discutido na Seção 2.2.2.2.4, o parâmetro m representa a limitação de memória do LBFGS, ou seja, o número máximo de iterações utilizadas na atualização da matriz quasi-Newton. O pseudocódigo do procedimento BC-GRASP é apresentado na Figura 4.5.

```

procedimento BC-GRASP( $n, l, u, f(\cdot), h_s, h_e, MaxIters, m$ )
1    $f^* \leftarrow \infty$ ;
2   enquanto Critério de parada não satisfeito faça
3        $x \leftarrow \text{RandomUniforme}(l, u)$ ;
4        $h \leftarrow h_s$ ;
5       enquanto  $h \geq h_e$  faça
6            $\text{Impr}_C \leftarrow \text{false}$ ;
7            $\text{Impr}_L \leftarrow \text{false}$ ;
8            $[x, \text{Impr}_C] \leftarrow \text{Construção}(x, f(\cdot), h, n, \text{Impr}_C)$ ;
9            $[x, \text{Impr}_L] \leftarrow \text{BuscaLocal\_APS}(x, f(\cdot), h, n, h_e, MaxIters, \text{Impr}_L)$ ;
10          se  $\text{Impr}_C = \text{false}$  and  $\text{Impr}_L = \text{false}$  então
11               $x \leftarrow \text{LBFGS}(x, f(\cdot), n, m)$ ;
12               $h \leftarrow h/2$ ; /* torna a grade mais densa */
13          fim se;
14          se  $f(x) < f^*$  então
15               $x^* \leftarrow x$ ;
16               $f^* \leftarrow f(x)$ ;
17          fim se;
18      fim enquanto;
19  fim enquanto;
20  retorne  $x^*$ ;
fim BC-GRASP

```

Figura 4.5. Pseudocódigo do procedimento BC-GRASP.

Conforme podemos observar na Figura 4.5, o procedimento BC-GRASP apresenta uma estrutura muito similar ao EC-GRASP. Cada iteração é formada por um

conjunto de ciclos construção-busca local. Os mesmos procedimentos de construção e busca local do EC-GRASP são utilizados. O domínio é discretizado em uma grade de pontos uniformemente distribuídos que vai adaptativamente se tornando mais densa à medida que o algoritmo progride.

A diferença entre os métodos está na linha 11 do BC-GRASP, onde ele invoca o método LBFGS. Quando as variáveis Impr_C e Impr_L estão no estado *false* (linha 10), ou seja, os procedimentos de construção e busca local não conseguiram melhorar a solução de entrada x , então o BC-GRASP invoca o procedimento LBFGS (com parâmetro m) para intensificar a busca naquela região. Em seguida, o tamanho do passo de discretização h é reduzido pela metade (linha 12). Compara-se então a solução atual x com a melhor solução visitada até o momento x^* (linha 14). Se a solução x for melhor que x^* (ou seja, $f(x) < f(x^*)$), atualiza-se a solução x^* com a solução x (linhas 15 e 16). Por fim, inicia-se um novo ciclo construção-busca local.

5. RESULTADOS E DISCUSSÕES

Nesse capítulo serão apresentados os experimentos computacionais utilizados para validar os métodos propostos EC-GRASP e BC-GRASP (apresentados no Capítulo 4) e avaliar sua eficiência e robustez. Nesse contexto, os métodos propostos serão comparados com outras versões da metaheurística C-GRASP e com outras metaheurísticas propostas na literatura.

Na Seção 5.1 serão apresentados o ambiente de testes e os problemas testes utilizado nos experimentos. Na Seção 5.2, o método EC-GRASP será confrontado com as duas versões da metaheurística C-GRASP propostas em [5] e [6, 7], respectivamente, para um conjunto de funções de teste padrão. Por questões de simplicidade, a versão do C-GRASP proposta em [5] será denotada “Original-CGRASP” e a versão proposta em [6,7] será denotada “Novo-CGRASP”.

Na Seção 5.3, o método EC-GRASP será confrontado com outras metaheurísticas propostas na literatura, como as metaheurísticas *Directed Tabu Search* (DTS) [9], *Genetic Algorithm for Numerical Optimization of Constrained Problems* (Genocop III) [30] e *Scatter Search* (SS) [31]. Por fim, na Seção 5.4 confrontaremos os dois métodos propostos EC-GRASP e BC-GRASP.

5.1 Ambiente de Testes

Os experimentos computacionais foram executados em um *notebook* Dell Vostro 1000 com processador AMD-Athlon 64 X2 1,9 GHz e com 2 GB de memória RAM. O sistema operacional utilizado foi o Ubuntu Linux 8.0.4, kernel 2.6.24-16. Os códigos foram implementados utilizando a linguagem de programação C++ e o compilador GNU g++ 3.4.6.

O algoritmo utilizado para geração de números aleatórios foi o algoritmo Mersenne Twister [32]. O método de busca linear utilizado no procedimento APS foi o Método da Seção Áurea [12], descrito na Seção 2.2.1.2. O código-fonte disponível em [33] foi utilizado na implementação do método LBFGS.

As funções de teste utilizadas nesse trabalho são apresentadas na Tabela 5.1. Essas funções possuem mínimos globais conhecidos e de acordo com Hedar e

Fukushima [9], “essas funções possuem características suficientemente diversas para cobrir muitos tipos de dificuldades encontradas nos problemas de otimização global contínua”. Além disso, importantes trabalhos da literatura como C-GRASP [5, 6], DTS [9], Genocop III [30] e SS [31] utilizaram essas funções para testar e comparar seus algoritmos.

Tabela 5.1. 42 funções (ou problemas) testes.

Nome da função	Dimensões	Nome da função	Dimensões
Beale (BE)	2	Bohachevsky (B ₂)	2
Booth (BO)	2	Branin (BR)	2
Easom (EA)	2	Goldstein and Price (GP)	2
Matyas (M)	2	Rosenbrock (R ₂)	2
Schwefel (SC ₂)	2	Shubert (SH)	2
Six-Hump Camelback (CA)	2	Zakharov (Z ₂)	2
De Jong (SP ₃)	3	Hartmann (H _{3,4})	3
Colville (CV)	4	Perm (P _{4,1/2})	4
Perm ₀ (P _{4,10})	4	Power Sum (PS _{4,{ 8,18,44,114 }})	4
Shekel (S _{4,5})	4	Shekel (S _{4,7})	4
Shekel (S _{4,10})	4	Rosenbrock (R ₅)	5
Zakharov (Z ₅)	5	Hartmann (H _{6,4})	6
Schwefel (SC ₆)	6	Trid (T ₆)	6
Griewank (GR ₁₀)	10	Rastrigin (RA ₁₀)	10
Rosenbrock (R ₁₀)	10	Sum Squares (SS ₁₀)	10
Trid (T ₁₀)	10	Zakharov (Z ₁₀)	10
Griewank (GR ₂₀)	20	Rastrigin (RA ₂₀)	20
Rosenbrock (R ₂₀)	20	Sum Squares (SS ₂₀)	20
Zakharov (Z ₂₀)	20	Powell (PW ₂₄)	24
Dixon and Price (DP ₂₅)	25	Ackley (A ₃₀)	30
Levy (L ₃₀)	30	Sphere (SP ₃₀)	30

O Anexo I apresenta a definição de todas as funções da Tabela 5.1. As Figuras 5.1 e 5.2 apresentam uma representação geométrica dessas funções para duas variáveis ($n=2$). As funções Hartmann, Levy, Powell, Power Sum e Shekel não foram representadas nas Figuras 5.1 e 5.2.

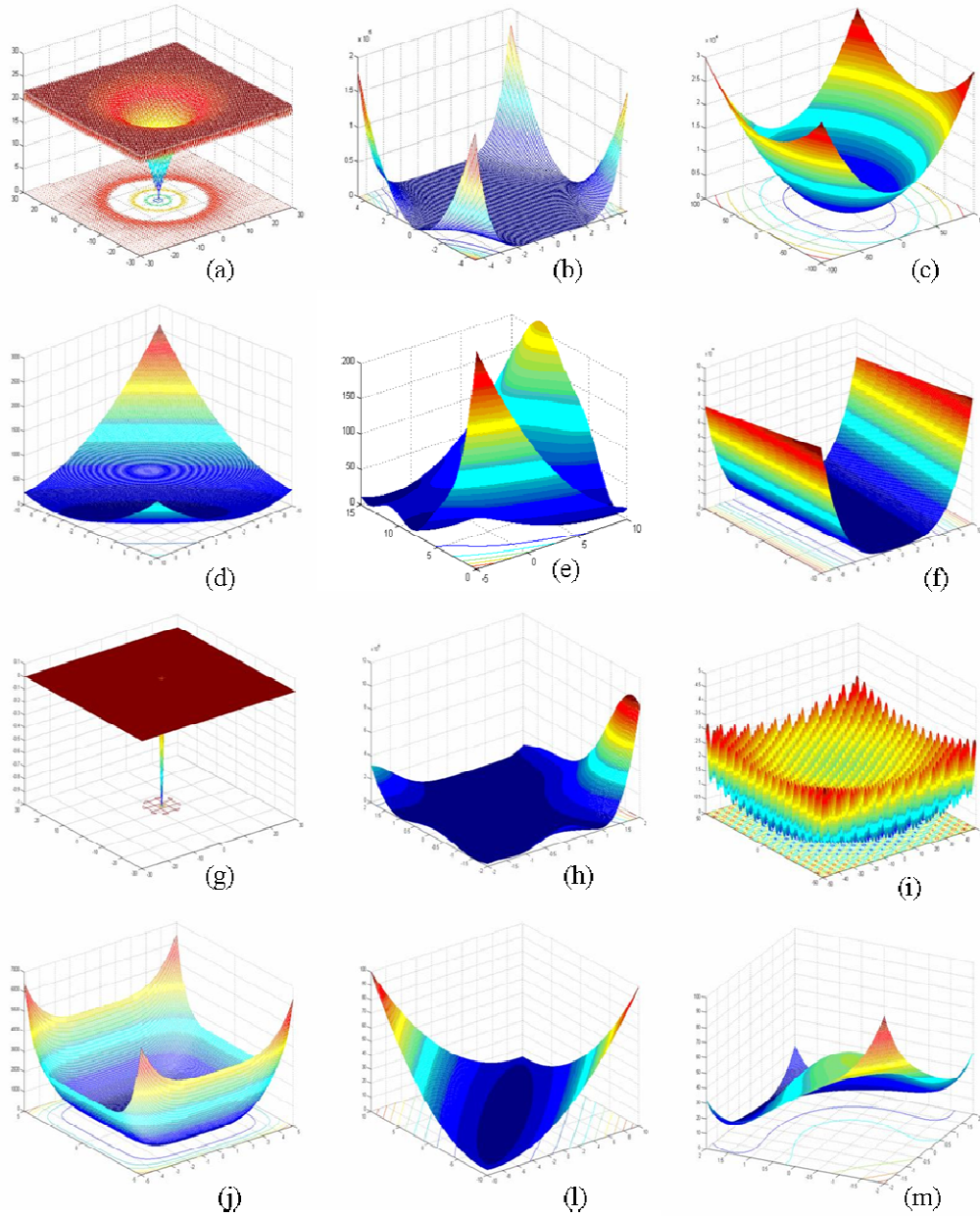


Figura 5.1. Representação geométrica de um conjunto de funções de teste para $n=2$. (a) Ackley; (b) Beale; (c) Bohachevsky; (d) Booth; (e) Branin; (f) Dixon and Price; (g) Easom; (h) Goldstein and Price; (i) Griewank; (j) Six-Hump Camelback; (l) Matyas; (m) Perm.

5.2 Comparando EC-GRASP com C-GRASP

Inicialmente, o método EC-GRASP foi comparado com as metaheurísticas “Original-CGRASP” [5] e “Novo-CGRASP” [6, 7] para um subconjunto de problemas (destacados em **negrito**) da Tabela 5.1. Esse subconjunto de problemas foi escolhido porque ele foi utilizado em [6] para testar e comparar os algoritmos Original-CGRASP

e Novo-CGRASP.

Como esses problemas possuem mínimos globais conhecidos, as metaheurísticas foram executadas até que o valor atual da função objetivo $f(x)$ se aproximasse significativamente do mínimo global $f(x^*)$. Assim como em [5-7, 9], a solução atual x é considerada *significativamente próxima* do ótimo global x^* se ela obedecer a seguinte inequação:

$$|f(x^*) - f(x)| \leq \varepsilon_1 |f(x^*)| + \varepsilon_2, \text{ onde}$$

$$\varepsilon_1 = 10^{-4} \text{ e } \varepsilon_2 = 10^{-6}$$

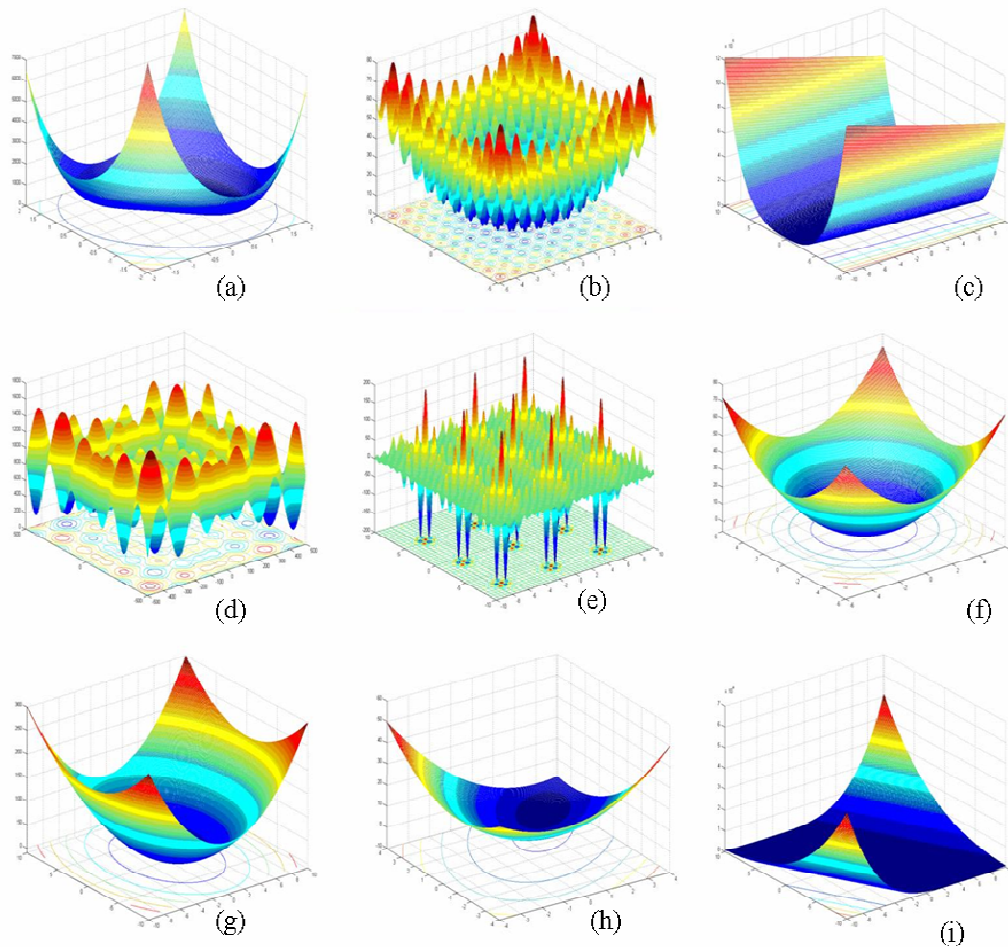


Figura 5.2. Representação geométrica de um conjunto de funções de teste para $n=2$. (a) Perm₀; (b) Rastrigin; (c) Rosenbrock; (d) Schwefel; (e) Shubert; (f) Sphere; (g) Sum Squares; (h) Trid; (i) Zakharov

De forma similar ao Original-CGRASP [5] e o Novo-CGRASP [6], o método EC-GRASP foi executados 100 vezes utilizando uma semente do gerador de números

aleatórios (*seed*) diferente para cada execução. Para cada um dos métodos, o critério de parada seria encontrar uma solução significativamente próximo do ótimo ou completar 20 iterações *multi-start*. O número de avaliações da função objetivo, o tempo decorrido para satisfazer um dos critérios de parada e o percentual de execuções em que a solução encontrada estava significativamente próxima do ótimo global foram armazenados.

Em todas as funções testadas, o parâmetro *MaxIters* foi configurado com o valor $2n$, onde n é o número de dimensões do problema. Os valores dos parâmetros h_s e h_e são apresentados na Tabela 5.2. Para comparação com o C-GRASP, assumiu-se que os resultados publicados em [5,6] são válidos (ou seja, esses algoritmos não foram implementados).

Tabela 5.2. Valor dos parâmetros h_s e h_e do EC-GRASP.

Problema Teste	h_s	h_e	Problema Teste	h_s	h_e
BR	1,0	0,001	R ₁₀	1,0	0,1
GP	1,0	1,0	S _{4,5}	1,0	0,5
SH	1,0	0,01	S _{4,7}	1,0	0,5
EA	1,0	0,1	S _{4,10}	1,0	0,5
H _{3,4}	0,5	0,001	Z ₅	1,0	0,5
R ₂	1,0	0,1	Z ₁₀	1,0	0,05
R ₅	1,0	0,1			

Na Tabela 5.3 são apresentados os resultados da comparação do método EC-GRASP com os métodos “Original-CGRASP” e “Novo-CGRASP” em relação ao número médio de avaliações da função objetivo e ao percentual de execuções em que o algoritmo encontrou uma solução significativamente próxima do mínimo global, ou seja, o percentual de convergência do algoritmo. Como essas metaheurísticas não utilizam cálculos de derivadas em sua estrutura, o número de avaliações da função objetivo é uma boa métrica para comparar o desempenho desses métodos.

Conforme destacado nos resultados em negrito da Tabela 5.3, na maioria dos problemas, houve uma redução significativa no número de avaliações da função objetivo pelo método EC-GRASP, comprovando sua eficiência computacional. Comparado com o Novo-CGRASP, por exemplo, observamos uma redução do número de avaliações de função de aproximadamente, 77,06%, 89,19% e 98,86%, para as funções S_{4,7}, R₁₀ e Z₁₀, respectivamente.

Além disso, não há nenhum impacto negativo no percentual de convergência do EC-GRASP. Em todos os problemas, o EC-GRASP obteve um percentual de convergência de 100%, comprovando a robustez do método.

Tabela 5.3. Comparação entre EC-GRASP e as metaheurísticas C-GRASP (“Original-CGRASP” [5] e “Novo_CGRASP” [6]).

Problema Teste	Original-CGRASP		Novo-CGRASP		EC-GRASP	
	Convergência (%)	Avaliações de Função	Convergência (%)	Avaliações de Função	Convergência (%)	Avaliações de Função
BR	100	59.857	100	10.090	100	8.735
GP	100	29	100	53	100	84
EA	100	89.630	100	5.093	100	96.627
SH	100	82.363	100	18.608	100	12.990
H _{3,4}	100	20.743	100	1.719	100	9.441
R ₂	100	1.158.350	100	23.544	100	5.038
R ₅	100	6.205.503	100	182.520	100	18.025
R ₁₀	99	20.282.529	100	725.281	100	78.375
S _{4,5}	100	5.545.982	100	9.274	100	1.286
S _{4,7}	100	4.052.800	100	11.766	100	1.739
S _{4,10}	100	4.701.358	100	17.612	100	4.040
Z ₅	100	959	100	12.467	100	924
Z ₁₀	100	3.607.653	100	2.297.937	100	26.159

5.3 Comparando EC-GRASP com outras metaheurísticas

Dando continuidade aos experimentos computacionais, comparamos o EC-GRASP com outras metaheurísticas propostas na literatura. São elas:

- *Directed Tabu Search* (DTS) [9];
- *Genetic Algorithm for Numerical Optimization of Constrained Problems* (Genocop III) [30];
- *Scatter Search* (SS) [31];

Essas metaheurísticas são adaptações das metaheurísticas Busca Tabu, Algoritmos Genéticos e *Scatter Search*, respectivamente. Os resultados da metaheurística Novo-CGRASP [6] também são considerados nessa seção.

As metaheurísticas são comparadas para 40 (quarenta) das funções testes da

Tabela 5.1. As funções R_5 e Z_5 não foram consideradas nesses experimentos, uma vez que elas não foram consideradas nos experimentos das metaheurísticas Novo-CGRASP[6], DTS [9], Genocop III [30] e SS [31]. Para comparar as metaheurísticas, consideramos o gap de otimalidade $GAP = \|f(x) - f(x^*)\|$, onde x é melhor solução atual encontrada pela metaheurística e x^* é o mínimo global do problema.

Em vários pontos do programa durante cada execução do EC-GRASP, valores de GAP foram calculados. Os valores do GAP médio para as 40 funções teste em função do número de avaliações da função objetivo estão listados na Tabela 5.4. De forma similar ao Novo-CGRASP [6], para cada função testada, o EC-GRASP foi executado 100 vezes. Os resultados do Novo-CGRASP, DTS, Genocop III e SS foram extraídos de [6]. Nas Tabelas 5.5 e 5.6 são apresentados, respectivamente, os valores de GAP médio e os parâmetros utilizados pelo EC-GRASP para cada uma das 40 funções de teste consideradas.

Tabela 5.4. Valores do GAP médio para os 40 problemas testes.

Metaheurística	Avaliações de Função						
	100	500	1000	5000	10000	20000	50000
Genocop	$5,37 e^{25}$	$2,39 e^{17}$	$1,13 e^{14}$	636,37	399,52	320,84	313,34
Scatter Search	134,45	26,34	14,66	4,96	3,60	3,52	3,46
DTS	50400,00	43,06	24,26	4,22	1,80	1,70	1,29
Novo-CGRASP	23610,61	10185,84	1341,70	6,20	4,73	3,92	3,02
EC-GRASP	632,80	11,88	9,65	4,40	3,37	1,75	0,89

Os valores do GAP médio (em função do número de avaliações da função objetivo) correspondem a outra boa métrica para comparar a eficiência dos algoritmos. Considerando-se que quanto menor o valor desse GAP , mais próximo o algoritmo estará do mínimo global, conseqüentemente, maior será a capacidade dele produzir melhores soluções com o mesmo esforço computacional.

Analisando os dados da Tabela 5.4, observa-se que o EC-GRASP apresenta um excelente desempenho quando comparado as metaheurísticas Genocop III, DTS, SS e Novo-CGRASP. Considerando todos os problemas testes, o EC-GRASP obteve o menor GAP médio entre todas as metaheurísticas para 500, 1000 e 50000 avaliações da função objetivo. Para os outros valores avaliados, apenas uma das metaheurísticas superou o EC-GRASP (a metaheurística SS para 100 avaliações da função objetivo e a

metaheurística DTS para 5000, 10000 e 20000 avaliações da função objetivo).

Tabela 5.5. Valores de GAP médio para cada um dos problemas testes.

Problemas Testes	Avaliações de Função						
	100	500	1000	5000	10000	20000	50000
BE	0,8238	0,1970	0,0606	0,0000	0,0000	0,0000	0,0000
B ₂	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
BO	0,2880	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
BR	0,1807	0,0261	0,0128	0,0001	0,0000	0,0000	0,0000
EA	0,8399	0,7739	0,7699	0,7516	0,5429	0,2244	0,0505
GP	3,8220	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
M	0,0155	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
R ₂	1,8125	0,8638	0,5470	0,0399	0,0097	0,0000	0,0000
SC ₂	0,2368	0,1012	0,1012	0,0098	0,0028	0,0012	0,0004
SH	42,1901	8,1957	2,5068	0,0778	0,0238	0,0049	0,0000
CA	0,2707	0,0463	0,0164	0,0004	0,0000	0,0000	0,0000
Z ₂	0,0459	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
SP ₃	0,2462	0,0517	0,0287	0,0042	0,0021	0,0011	0,0006
H _{3,4}	0,8618	0,1452	0,0563	0,0012	0,0002	0,0000	0,0000
CV	189,2070	8,7632	4,4758	1,7239	1,4793	1,4022	1,3551
P _{4,1/2}	51,8402	23,4249	20,3649	10,0239	7,1521	5,0159	2,8553
P _{4,10}	236,2939	31,6021	8,3086	1,7947	1,2555	0,8519	0,5671
PS _{4,{8,18,44,114}}	4,2230	0,0081	0,0000	0,0000	0,0000	0,0000	0,0000
S _{4,5}	6,5135	3,6387	2,1736	0,2021	0,0000	0,0000	0,0000
S _{4,7}	6,8525	4,5586	3,1873	0,3387	0,0000	0,0000	0,0000
S _{4,10}	7,8216	5,3158	4,1550	1,0790	0,4025	0,0000	0,0000
H _{6,4}	1,0460	0,7740	0,5586	0,2572	0,1931	0,1618	0,1360
SC ₆	322,8288	80,4994	59,8433	14,9723	8,4573	4,6578	0,9323
T ₆	187,2500	0,4691	0,4005	0,1269	0,0220	0,0000	0,0000
GR ₁₀	0,6841	0,5739	0,5231	0,5137	0,4151	0,2994	0,1673
RA ₁₀	7,0584	6,8550	6,8550	3,7429	2,7711	2,2057	1,7071
R ₁₀	7458,8100	107,8212	87,1108	29,2855	18,4175	8,4771	2,3323
SS ₁₀	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
T ₁₀	2171,6500	34,2317	33,2116	20,9272	14,3342	6,3132	3,1471
Z ₁₀	184,6938	8,4395	7,9900	2,0471	0,7803	0,1675	0,0135
GR ₂₀	0,6204	0,5923	0,5546	0,5490	0,4780	0,4658	0,4130
RA ₂₀	14,1167	13,8456	13,8456	13,5149	8,3836	6,1788	4,7887
R ₂₀	12506,4420	8,4005	7,1061	4,6924	4,5224	4,3041	4,0140
SS ₂₀	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
Z ₂₀	512,7381	108,1708	107,7074	61,6530	59,4142	25,9154	10,6575
PW ₂₄	141,6700	0,0763	0,0052	0,0010	0,0000	0,0000	0,0000
DP ₂₅	605,7000	3,0769	2,0000	1,9987	1,1874	0,7618	0,5340
A ₃₀	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
L ₃₀	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
SP ₃₀	9,4080	1,8843	1,8843	1,3942	1,3942	1,0919	0,8605

Tabela 5.6. Valor dos parâmetros do EC-GRASP para cada um dos problemas testes.

Problema Teste	h_s	h_e	Problema Teste	h_s	h_e	Problema Teste	h_s	h_e	Problema Teste	h_s	h_e
BE	1,0	0,5	CA	1,0	0,01	$S_{4,10}$	1,0	0,5	GR_{20}	10	0,25
B_2	1,0	0,5	Z_2	1,0	0,5	$H_{6,4}$	0,5	0,0001	RA_{20}	0,5	0,1
BO	1,0	0,5	SP_3	2,0	0,05	SC_6	50	0,25	R_{20}	0,1	0,05
BR	1,0	0,001	$H_{3,4}$	0,5	0,001	T_6	1,0	0,1	SS_{20}	1,0	0,1
EA	1,0	0,1	CV	1,0	0,1	GR_{10}	10	0,25	Z_{20}	1,0	0,005
GP	1,0	1,0	$P_{4,1/2}$	0,1	0,0125	RA_{10}	0,5	0,1	PW_{24}	1,0	0,1
MA	1,0	0,1	$P_{4,10}$	0,1	0,1	R_{10}	1,0	0,1	DP_{25}	5,0	0,25
R_2	1,0	0,1	$PS_{4,\{8,18,44,114\}}$	1,0	0,1	SS_{10}	1,0	0,5	A_{30}	5,0	0,25
SC_2	5,0	0,25	$S_{4,5}$	1,0	0,5	T_{10}	5,0	0,1	L_{30}	1,0	0,1
SH	1,0	0,01	$S_{4,7}$	1,0	0,5	Z_{10}	1,0	0,005	SP_{30}	2,0	0,05

Além de mostrar sua eficiência quando comparado a outras metaheurísticas, os dados da Tabela 5.4 também comprovam a capacidade do EC-GRASP de acelerar a convergência do C-GRASP. Utilizando pequenos esforços computacionais como, por exemplo, 100, 500 e 1.000 avaliações da função objetivo, o EC-GRASP obteve GAPs médios equivalentes a 632,80, 11,80 e 9,65, respectivamente, enquanto que Novo-CGRASP obteve GAPs médios de 23.610,61, 10.185,84 e 1.341,70, respectivamente, uma redução de aproximadamente 97,31%, 99,88% e 99,28%, respectivamente.

5.4 Comparando EC-GRASP com BC-GRASP

Comprovada a eficiência e robustez do método EC-GRASP, avaliaremos a eficiência e robustez do método proposto BC-GRASP, comparando os dois métodos. O método BC-GRASP, diferentemente de todas as metaheurísticas comparadas (C-GRASP, DTS, Genocop III e SS), utiliza cálculos de derivada de primeira ordem em sua estrutura. Como esses cálculos, em geral, consomem muito tempo computacionalmente, não é conveniente comparar a eficiência do BC-GRASP com as outras metaheurísticas em função do número de avaliações da função objetivo.

Contudo, como o método EC-GRASP foi comparado com as metaheurísticas C-GRASP, DTS, Genocop III e SS e mostrou-se eficiente e robusto, analisaremos a eficiência e robustez do BC-GRASP, por analogia, comparando-o com o EC-GRASP.

Para comparação dos dois métodos propostos, inicialmente, o BC-GRASP foi executado para o subconjunto de problemas testes da Seção 5.2. De forma similar ao EC-GRASP, o BC-GRASP foi executado 100 vezes utilizando uma semente do gerador

de números aleatórios (*seed*) diferente para cada execução. O critério de parada seria encontrar uma solução significativamente próxima do ótimo ou completar 20 iterações *multi-start*.

O número médio de avaliações da função objetivo, o número médio de avaliações do gradiente da função objetivo, o tempo médio decorrido e o percentual de execuções em que a solução encontrada estava significativamente próxima do ótimo global foram armazenados. O parâmetro *MaxIters* também foi configurado com o valor $2n$ e os parâmetros h_s e h_e foram configurados de acordo com a Tabela 5.2. O parâmetro m , um parâmetro específico do BC-GRASP, foi configurado com o valor 2 para todos os problemas testes.

Tabela 5.7. Comparação entre os métodos EC-GRASP e BC-GRASP.

Problema Teste	EC-GRASP			BC-GRASP			
	Convergência (%)	Avaliações Função	Tempo (s)	Convergência (%)	Avaliações Função	Avaliações Gradiente	Tempo (s)
BR	100	8.735	0,005	100	7.931	130	0,005
GP	100	84	> 0,001	100	79	0	> 0,001
EA	100	96.627	0,035	100	17.545	30	0,011
SH	100	12.990	0,011	100	373	11	0,001
H _{3,4}	100	9.441	0,015	100	8.912	64	0,014
R ₂	100	5.038	0,001	100	218	40	> 0,001
R ₅	100	18.025	0,007	100	15.077	106	0,004
R ₁₀	100	78.375	0,037	100	37.266	262	0,019
S _{4,5}	100	1.286	0,005	100	1.367	106	0,005
S _{4,7}	100	1.739	0,007	100	1.896	150	0,007
S _{4,10}	100	4.040	0,011	100	2.859	165	0,011
Z ₅	100	924	0,001	100	917	0	0,001
Z ₁₀	100	26.159	0,013	100	1.225	7	0,010

Os resultados são apresentados na Tabela 5.7. Como o número médio de avaliações da função objetivo não é uma métrica conveniente, o tempo médio decorrido foi utilizado para comparar o desempenho dos algoritmos, uma vez que ambos foram executados no mesmo ambiente de teste (descrito na Seção 5.1).

Com o objetivo de minimizar erros nos cálculos do tempo decorrido, uma vez que o sistema operacional utilizado (Seção 5.1) é multi-tarefa, os procedimentos BC-

GRASP e EC-GRASP foram executados de forma dedicada no ambiente de testes utilizado. Além disso, os procedimentos foram executados de forma intercalada para cada um dos problemas testes da Tabela 5.7.

De acordo com os dados da Tabela 5.7, o BC-GRASP conseguiu reduzir ou equiparar o tempo médio decorrido para encontrar uma solução significativamente próxima do mínimo global pelo EC-GRASP para a maioria dos problemas testados. Em consequência disso, podemos afirmar que o método BC-GRASP é mais eficiente ou tão eficiente quanto o EC-GRASP para o conjunto de problemas das Tabelas 5.4 e 5.5. Além disso, de forma similar ao EC-GRASP, o BC-GRASP obteve 100% de convergência em todos os problemas, comprovando a robustez do método.

Com o objetivo de avaliar o efeito de alta dimensionalidade nos métodos propostos, um conjunto de experimentos foi aplicado sobre os métodos EC-GRASP e o BC-GRASP. Esses experimentos utilizaram altos valores de n (dimensões) sobre as funções Rosenbrock (R_n) e Zakharov (Z_n). A função Rosenbrock foi avaliada com 20, 50, 100 e 500 dimensões (R_{20} , R_{50} , R_{100} e R_{500} , respectivamente) e a função Zakharov com 20, 50 e 100 dimensões (Z_{20} , Z_{50} e Z_{100} , respectivamente).

De forma similar aos experimentos anteriores, cada método foi executado 100 vezes para cada função testada utilizando uma semente diferente para cada execução e os mesmos critérios de parada (20 iterações *multi-start* ou encontrar uma solução significativamente próxima da solução ótima). O parâmetro *MaxIters* foi configurado com o valor $2n$ para os dois métodos (EC-GRASP e BC-GRASP). O parâmetro m (do BC-GRASP) foi configurado com o valor 7 para todos os problemas testes. Os dois métodos propostos utilizaram os parâmetros de discretização h_s e h_e apresentados na Tabela 5.8.

Tabela 5.8. Valor dos parâmetros do EC-GRASP e BC-GRASP.

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

Os resultados em relação ao número médio de avaliações da função objetivo,

número médio de avaliações do vetor gradiente da função objetivo, tempo médio decorrido e percentual de convergência são apresentados na Tabela 5.9.

De acordo com os resultados da Tabela 5.9, observa-se que, mesmo em condições de alta dimensionalidade, os métodos EC-GRASP e BC-GRASP conseguiram manter a robustez obtendo 100% de convergência em todos os problemas. Além disso, observa-se que, em todos os problemas, o BC-GRASP reduziu significativamente o tempo médio necessário para encontrar uma solução próxima do mínimo global, se comportando, portanto, de forma mais eficiente.

Tabela 5.9. Comparação entre os métodos propostos com alta dimensionalidade.

Problema Teste	EC-GRASP			BC-GRASP			
	Convergência (%)	Avaliações Função	Tempo (s)	Convergência (%)	Avaliações Função	Avaliações Gradiente	Tempo (s)
R ₂₀	100	170.053	0,105	100	127.646	480	0,076
R ₅₀	100	619.392	0,917	100	463.609	494	0,605
R ₁₀₀	100	2.236.963	6,823	100	1.383.466	460	3,801
R ₅₀₀	100	32.134.487	661,714	100	10.169.780	194	204,083
Z ₂₀	100	4.932.048	2,297	100	285.739	35	0,216
Z ₅₀	100	9.659.774	8,730	100	637.629	13	1,444
Z ₁₀₀	100	20.826.130	73,518	100	10.988.031	2	39,649

Os resultados apresentados nessa seção comprovam que mesmo utilizando cálculos de derivada de primeira ordem da função objetivo, o método BC-GRASP é eficiente e robusto. Além disso, o BC-GRASP é mais eficiente que o EC-GRASP, especialmente, para funções com alta dimensionalidade.

6. CONSIDERAÇÕES FINAIS

Esse trabalho propôs a implementação de métodos híbridos baseados em *Continuous-GRASP* (C-GRASP) [5-7] para resolução de problemas de otimização global contínua. Por utilizar construções e direções aleatórias, o C-GRASP, em geral, apresenta uma convergência lenta. Com o objetivo de acelerar a convergência do C-GRASP foram propostos dois métodos híbridos denominados *Enhanced Continuous-GRASP* (EC-GRASP) e *BFGS Continuous-GRASP* (BC-GRASP).

Esses métodos utilizam o C-GRASP como base combinados com etapas de refinamento compostas por métodos de descidas como o método *Adaptive Pattern Search* (APS), um método de busca direcionada e o *BFGS* com memória limitada (*Limited Memory BFGS* - LBFGS), um método de busca multidimensional com derivadas. O EC-GRASP é formado pela hibridização do C-GRASP com o método de busca direcionada APS e o BC-GRASP é formado pela hibridização com o APS e o LBFGS.

Para validar os métodos propostos e avaliar sua eficiência e robustez, os métodos foram implementados e comparados com o C-GRASP e algumas importantes metaheurísticas, como a *Directed Tabu Search* (DTS) e *Scatter Search* (SS). Essa comparação foi realizada para um conjunto de problemas testes da literatura com mínimo global conhecido. Os resultados computacionais comprovaram a capacidade dos métodos de acelerar a convergência do C-GRASP, aliadas a sua eficiência e robustez quando comparado a outras metaheurísticas.

Em relação ao número médio de avaliações da função objetivo, por exemplo, observou-se uma redução significativa do EC-GRASP em relação ao C-GRASP para um subconjunto desses problemas. Para as funções $S_{4,7}$, R_{10} e Z_{10} , por exemplo, a redução no número médio de avaliações da função objetivo foi de aproximadamente 77,06%, 89,19% e 98,86, respectivamente. Além disso, não houve nenhum impacto negativo em relação ao percentual de convergência.

Os valores do GAP médio também foram significativamente reduzidos. Utilizando pequenos esforços computacionais como, por exemplo, 100, 500 e 1.000 avaliações da função objetivo, o EC-GRASP obteve GAPs médios equivalentes a

632,80, 11,80 e 9,65, respectivamente, enquanto que C-GRASP obteve GAPs médios de 23.610,61, 10.185,84 e 1.341,70, respectivamente.

Comparando com outras metaheurísticas, o EC-GRASP também se mostrou eficiente. Considerando todos os problemas testes, o EC-GRASP obteve o menor *GAP* médio entre todas as metaheurísticas para 500, 1000 e 50000 avaliações da função objetivo. Para os outros valores avaliados, apenas uma das metaheurísticas comparadas superou o EC-GRASP (a metaheurística SS para 100 avaliações da função objetivo e a metaheurística DTS para 5000, 10000 e 20000 avaliações da função objetivo).

O método proposto BC-GRASP provou ser robusto mesmo utilizando cálculos de derivadas de primeira ordem, pois obteve 100% de convergência para todas as funções testadas. Além disso, o BC-GRASP mostrou-se mais eficiente que o EC-GRASP, reduzindo significativamente o esforço computacional para encontrar uma solução próxima ao mínimo global, especialmente, para problemas com alta dimensionalidade.

6.1 Trabalhos Futuros

Como propostas de trabalhos futuros, destacamos a aplicação dos métodos propostos EC-GRASP e BC-GRASP em problemas reais de grande porte e a adaptação desses métodos para resolução de problemas de otimização com restrições. Como os principais problemas reais possuem restrições, essa adaptação permitiria aplicar o EC-GRASP e o BC-GRASP para resolver muitos problemas associados a tomadas de decisão.

Outra proposta de trabalho interessante seria estudar a hibridização do C-GRASP com outras metaheurísticas propostas na literatura como o *Particle Swarm Optimization* (PSO), *Directed Tabu Search* (DTS), ou *Scatter Search* (SS). Outros métodos de descida também poderiam ser utilizados como, por exemplo, o Método Nelder-Mead [20] ou o Método dos Gradientes Conjugados [12-15].

7. REFERÊNCIAS BIBLIOGRÁFICAS

- [1] FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v.6, n.2, p. 109-133, 1995.
- [2] GLOVER, F. Tabu Search-Part I. *ORSA Journal on Computing*, v. 1, n.3, p. 190-206, 1989.
- [3] GLOVER, F. Tabu Search-Part II. *ORSA Journal on Computing*, v. 2, n.1, p. 4-32, 1990.
- [4] KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671-680, 1983.
- [5] HIRSCH, M. J. et al. Global optimization by continuous GRASP. *Optimization Letters*, v. 1, n.2, p. 201-212, 2007.
- [6] HIRSCH, M. J.; PARDALOS, P. M.; RESENDE, M. G. C. Speeding up continuous GRASP. *European Journal of Operational Research*, submitted. (disponível em: <http://www.research.att.com/~mgcr/papers.html>, acessado em janeiro de 2009)
- [7] HIRSCH, M.J. et al. A continuous GRASP to determine the relationship between drugs and adverse reactions. In: *Data Mining, Systems Analysis e Optimization in Medicine*. AIP Conference Proceedings, v. 952, p. 106-121, 2008.
- [8] HEDAR, A. R. *Studies on Metaheuristics for Continuous Global Optimization Problems*. 2004. 148 f. Tese (Doutorado em Informática) - Kyoto University, Kyoto, 2004.
- [9] 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.
- [10] 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.

- [11] CORMEN, T. et al. *Algoritmos: Teoria e Prática*. Trad. da 2ª. ed. americana sob direção de Vandberg D. Souza. Rio de Janeiro: Editora Campus, 2002.
- [12] BAZARAA, M. S.; SHERALI, H. D.; SHETTY, C. M. *Nonlinear Programming: Theory and Algorithms*. 2. ed. Estados Unidos: John Wiley & Sons, Inc., 1989. 638 p.
- [13] IZMAILOV, A.; SOLODOV, M. *Otimização - volume 2: Métodos computacionais*. Rio de Janeiro: Impa, 2007. 458 p.
- [14] PRESS, W.H. et al. *Numerical Recipes (Fortran version): The art of scientific computing*. Estados Unidos: Cambridge University Press, 1989. 702 p.
- [15] PINTÉR, J.D.. *Global Optimization in Action: Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications*. Dordrecht – Boston – London: Kluwer Academic Publishers, 1996. 512 p. (Edição Nonconvex Optimization and Its Applications , v. 6).
- [16] KOLDA, T. G.; LEWIS, R. M.; TORCZON, V. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, v. 45, n. 3, p.385-482, 2004.
- [17] ROSENBROCK, H. H. An automatic method for finding the greatest or least value of a function, *The Computer Journal*, v. 3, n. 3, p.175-184, 1960.
- [18] GILL, P. E.; MURRAY, W.; WRIGHT, M. H. *Practical Optimization*. London: Academic Press, 1981. 402 p.
- [19] TORCZON, V. On the convergence of the multidirectional search algorithm. *SIAM Journal on Optimization*, v.1, n. 1, p. 123-145, 1991.
- [20] NELDER, J.A.; MEAD, R. A simplex method for function minimization. *The Computer Journal*, v. 7, n. 4, p. 308-313, 1965.
- [21] HOOKE, R.; JEEVES, T.A. Direct search solution of numerical and statistical problems. *Journal of the ACM*, v. 8, n. 2, p. 212-229, 1961.
- [22] NOCEDAL, J. Updating Quasi-Newton Matrices with Limited Storage. *Mathematics of Computation*, v. 35, n. 151, p. 773-782, 1980.

- [23] LIU, D.C.; NOCEDAL, J. On the Limited Memory Method for Large Scale Optimization. *Mathematical Programming*, v. 45, n. 3, p. 503-528, 1989.
- [24] DIAS, T. *Algoritmos heurísticos e metaheurísticas híbridas aplicadas ao planejamento de uma rede de telecomunicações com topologia anel-estrela*. 100 f. Dissertação (Mestrado em Sistemas de Computação - UFRJ, Rio de Janeiro, 2006.
- [25] SCHAEFER, A. Tabu search techniques for large high-school timetabling problems. In: Thirteen National Conference on Artificial Intelligence (AAAI-96), 13., 1996, Portland. *Proceedings of the 13th National Conference on Artificial Intelligence*, Portland, 1996, p. 363-368.
- [26] DAMMEYER, F.; VOB, S.. Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, Amsterdã, v. 41, n. 2, p. 29-46, 1993.
- [27] DOWSLAND, K. A. Simulated Annealing. In: REEVES, C. *Modern heuristic techniques for combinatorial problems*. London: Blackwell Scientific Publications, 1993. p. 20-63.
- [28] GOLDBERG, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1. ed. Berkley: Addison-Wesley, 1989. 432 p.
- [29] KENNEDY, J.; EBEHART, R.C.. Particle swarm optimization. In: IEEE International Conference on Artificial Neural Networks, 1995, Cambridge. *Proceedings of the IEEE International Conference on Neural Networks*. 1995, v. 4, p. 1942-1948.
- [30] MICHALEWICZ, Z.; NAZHIYATH, G.. Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In: IEEE International Conference on Artificial on Evolutionary Computation, 1995, Australia. *Proceedings of IEEE International Conference on Evolutionary Computation*. IEEE Press, 1995, v. , p. 647-651.
- [31] LAGUNA, M.; MARTÍ, R. Experimental testing of advanced Scatter Search designs for global optimization of multimodal functions. *Journal of Global Optimization*, v. 33, n. 2, p. 235-255, 2005.
- [32] MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, v.8, n. 1, p. 3-30, 1998.

- [33] L-BFGS algorithm for multivariate optimization. Disponível em: <<http://www.alglib.net/optimization/lbfgs.php>>. Acesso em: 01 set. 2008.

ANEXO I - DEFINIÇÃO DOS PROBLEMAS TESTES

(A_n) Função Ackley:

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

Domínio: $[-15; 30]^n$

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

(BE) Função Beale

Definição: $BE(x) = (1,5 - x_1 - x_1 x_2)^2 + (2,25 - x_1 - x_1 x_2^2)^2 + (2,65 - x_1 - x_1 x_2^3)^2$

Domínio: $[-4,5; 4,5]^2$

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

(B₂) Função Bohachevsky

Definição: $B_2(x) = x_1^2 + 2x_2^2 - 0,3\cos(3\pi x_1) - 0,4\cos(4\pi x_2) + 0,7$

Domínio: $[-50; 100]^2$

Mínimo Global: $x^* = (0; 0)$; $B_2(x^*) = 0$

(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: $BR(x) = (x_2 - \frac{5}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10$

Domínio: $[-5; 15]^2$

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

(CV) Função Colville (também denominada Função Wood)

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: $DP_n(x) = (x_1 - 1)^2 + \sum_{i=2}^n i (2x_i^2 - x_{i-1})^2$

Domínio: $[-10; 10]^n$

Mínimo Global: $x_i^* = 2^{-\frac{2^i-2}{2^i}}, \forall i = 1, \dots, n$; $DP_n(x^*) = 0$

(EA) Função Easom

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

Domínio: $[-100; 100]^2$

Mínimo Global: $x^* = (\pi; \pi)$; $EA(x^*) = -1$

(GP) Função Goldstein and Price

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

Domínio: $[-2; 2]^2$

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

(GR_n) Função Griewank

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

Domínio: $[-300; 600]^n$;

Mínimo Global: $x^* = (0; \dots; 0)$; $GR_n(x^*) = 0$

(H_{n,m}) Função Hartmann

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

Domínio: $[0; 1]^n$

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

$$H_{3,4}(x^*) = -3,86278.$$

Mínimo Global ($n=6, m=4$): $x^* = (0,20169; 0,150011; 0,476874; 0,2785332;$

$$0,311652; 0,657300) ; H_{6,4}(x^*) = -3,32237.$$

Parâmetros:

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

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

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

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

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

$$\alpha = [1; 1,2; 3; 3,2]$$

(L_n) Função Levy

Definição: $L_n(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1} [(y_i - 1)^2 (1 + 10 \sin^2(\pi y_i + 1))] + (y_n - 1)^2 (10 \sin^2(2\pi y_n))$

Domínio: $[-10; 10]^n$

Mínimo Global: $x^* = (1; \dots; 1)$; $L_n(x^*) = 0$

Parâmetros: $y_i = 1 + (x_i - 1)/4, \forall i = 1, \dots, n$

(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,β}) Função Perm

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

Domínio: $[-n; n]^n$

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

(P_{n,β}⁰) Função Perm₀

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

Domínio: $[-n; n]^n$

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

(PW_n) Função Powell

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

Domínio: $[-4; 5]^n$

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

(PS_{n,b}) Função Power Sum

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

Domínio: $[0; n]^n$

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

(RA_n) Função Rastrigin

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

Domínio: $[-2,56; 5,12]^n$

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

(R_n) Função Rosenbrock

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

Domínio: $[-10; 10]^n$

Mínimo Global: $x^* = (1, \dots, 1); R_n(x^*) = 0$.

(SC_n) Função Schwefel

$$\text{Definição: } SC_n(x) = 418,9829 - \sum_{i=1}^n x_i \sin(\sqrt{|x_i|})$$

Domínio: $[-500; 500]^n$

Mínimo Global: $x^* = (420,9687; \dots; 420,9687); SC_n(x^*) = 0$

(S_{4,m}) Função Shekel

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

Domínio: $[0; 0]^4$

Mínimo Global: $x^* = (4, 4, 4, 4); S_{4,5}(x^*) = -10,15319538,$

$$S_{4,7}(x^*) = -10,40281868, S_{4,10}(x^*) = -10,53628349.$$

Parâmetros:

$$a = \begin{bmatrix} 4,0 & 4,0 & 4,0 & 4,0 \\ 1,0 & 1,0 & 1,0 & 1,0 \\ 8,0 & 8,0 & 8,0 & 8,0 \\ 6,0 & 6,0 & 6,0 & 6,0 \\ 7,0 & 3,0 & 7,0 & 3,0 \\ 2,0 & 9,0 & 2,0 & 9,0 \\ 5,0 & 5,0 & 3,0 & 3,0 \\ 8,0 & 1,0 & 8,0 & 1,0 \\ 6,0 & 2,0 & 6,0 & 2,0 \\ 7,0 & 2,6 & 7,0 & 3,6 \end{bmatrix}$$

$$c = [0,1; 0,2; 0,2; 0,4; 0,4; 0,6; 0,3; 0,7; 0,5; 0,5]$$

(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): $x^* = (5,48242188; 4,84742188)$; $SH(x^*) = -186,7309$.

(CA) Função Six-Hump CamelBack

Definição: $CA(x) = 4x_1^2 - 2,1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$

Domínio: $[-5; 5]^2$

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

(SP_n) Função Sphere

Definição: $SP_n(x) = \sum_{i=1}^n x_i^2$

Domínio: $[-2,56; 5,12]^n$

Mínimo Global: $x^* = (0; \dots; 0)$; $SP_n(x^*) = 0$

(SS_n) Função Sum of Squares

Definição: $SS_n(x) = \sum_{i=1}^n ix_i^2$

Domínio: $[-5; 10]^n$

Mínimo Global: $x^* = (0; \dots; 0)$; $SS_n(x^*) = 0$

(T_n) Função Trid

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

Domínio: $[-n^2; n^2]^n$

Mínimo Global: $x_i^* = i(n+1-i)$, $\forall i = 1, \dots, n$; $T_6(x^*) = -50$ e $T_{10}(x^*) = -210$

(Z_n) Função Zakharov

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

Domínio: $[-5; 10]^n$

Mínimo Global: $x^* = (0, 0, \dots, 0)$; $Z_n(x^*) = 0$.

ANEXO II – ARTIGO PUBLICADO

Como um dos resultados preliminares desse trabalho, o artigo **“Hibridizando a metaheurística C-GRASP com o método de busca por padrões adaptativos para resolução de problemas de otimização global contínua”** foi publicado nos *Anais do XV Simpósio Brasileiro de Engenharia de Produção (XV SIMPEP)*, realizado no período de 10 a 12 de Novembro de 2008 na cidade de Bauru – SP.

Esse artigo foi selecionado como **melhor artigo da área de Pesquisa Operacional do XV SIMPEP** e convidado para publicação na **Revista GEPROS: Gestão da Produção, Operações e Sistemas, ISSN 1984-2430**.

A versão publicada do artigo, o comprovante de publicação do artigo, a seleção como melhor artigo da área de Pesquisa Operacional e o convite para publicação do artigo na Revista GEPROS são apresentados nesse anexo.