UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RAPHAEL BEZERRA XAVIER

ABORDAGEM C-GRASP COM ADAPTAÇÃO AUTOMÁTICA PARA OTIMIZAÇÃO GLOBAL CONTÍNUA

JOÃO PESSOA
2017

RAPHAEL BEZERRA XAVIER

ABORDAGEM C-GRASP COM ADAPTAÇÃO AUTOMÁTICA PARA OTIMIZAÇÃO GLOBAL CONTÍNUA

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro de Informática da Universidade Federal da Paraíba, como requisito parcial para obtenção do grau de Mestre em Informática

Orientador: Prof. Dr. Lucídio dos Anjos Formiga Cabral

JOÃO PESSOA

2017

Catalogação na Publicação
Seção de Catalogação e Classificação

X3a Xavier, Raphael Bezerra.
Abordagem C-GRASP com adaptação automática para otimização global contínua / Raphael Bezerra Xavier. - João Pessoa, 2017.
68 f. : il.

Orientador: Dr. Lucídio dos Anjos Formiga Cabral.
Dissertação (Mestrado) - UFPB/CI/PPGI

1. Informática. 2. GRASP. 3. Meta-heurística. I. Título.

UFPB/BC CDU - 004(043)

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Raphael Bezerra Xavier, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 24 de agosto de 2017.

Aos vinte e quatro dias do mês de agosto, do ano de dois mil e dezessete, às quatorze e trinta, no Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. Raphael Bezerra Xavier, vinculado a esta Universidade sob a matrícula nº 2015104195, candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Computação distribuída", do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores: Lucidio dos Anjos Formiga Cabral (PPGI-UFPB), Orientador e Presidente da Banca, Anand Subramanian (PPGI-UFPB), Examinador interno, Roberto Quirino do Nascimento (UFPB), Examinador externo ao programa, e Marcone Jamilson Freitas Souza (UFOP), Examinador externo à instituição. Dando início aos trabalhos, o Presidente da Banca, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado "Abordagem C-GRASP com adaptação automática para otimização global contínua". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Clauirton de Albuquerque Siebra, Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 24 de agosto de 2017.

Prof. Dr. Clauirton de Albuquerque Siebra

Prof. Dr. Lucidio dos Anjos Formiga Cabral
Orientador (PPGI-UFPB)

Prof. Dr. Anand Subramanian
Examinador interno (PPGI-UFPB)

Prof. Dr. Roberto Quirino do Nascimento
Examinador externo ao programa (UFPB)

Prof. Dr. Marcone Jamilson Freitas Souza
Examinador externo à instituição (UFOP)

Resumo

Meta-heurísticas são normalmente utilizadas para resolver problemas de otimização combinatória, em que as variáveis envolvidas são discretas. No entanto, também podem ser aplicadas para resolver problemas de otimização global contínua. O *Greedy Randomized Adaptive Search Procedure* (GRASP) têm sido adaptadas para resolver problemas de otimização global contínua. A meta-heurística *Continuous* GRASP está incluída na classe das que mais sofreram adaptações para a resolução desses problemas. Alguns trabalhos possuem relevância por implementarem estas adaptações do *Continuous* GRASP (C-GRASP).

O *Directed Continuous* GRASP (DC-GRASP) é um melhoramento proposto para acelerar a convergência do método C-GRASP através da geração de direções de descida, sem cálculos de derivada, utilizando uma busca local baseada no método *Adaptive Pattern Search* (APS). Uma adaptação automática é inserida no DC-GRASP para otimizar a definição dos parâmetros em funções de alta dimensão, com um método utilizando a meta-heurística *Particle Swarm Optimization* (PSO). Nas funções com dimensões menores, um mecanismo de ampliação do tamanho do passo na busca local (APS) e o uso da busca linear inexata, foram propostas para melhorar o aproveitamento das iterações dos métodos.

Para validar o método implementado foram usadas as últimas implementações adaptativas do *Continuous* GRASP encontradas na literatura recente, como também em versões originais publicadas em artigos que usam a meta-heurística. Alguns experimentos computacionais foram realizados em um benchmark de funções de teste com mínimo global conhecido, comprovando assim a eficácia do método para o auxílio na convergência.

Palavras-chave: Otimização Contínua, *Continuous* GRASP, Adaptação automática, Meta-heurísticas, *Particle Swarm Optimization*.

Abstract

The use of meta-heuristic is strongly recommended for solving optimization problems. They are usually used to solve discrete optimization problems. The meta-heuristics aim to achieve good approximated solutions, and adaptive methods are implemented to achieve higher performance, thus improv ing the way the goal-heuristic works. Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS) and Variable Neighborhood Descent (VND) have been used to solve continuous global optimization problems along with its implemented adaptations. The meta-heuristic Continuous GRASP is included in the class that have most been adapted in order to solve these problems. Some works are relevant to implement these adaptations Continuous GRASP.

The Directed Continuous GRASP (DC-GRASP) is a proposed improvement to accelerate the convergence of the C-GRASP method by generating downward directions without derivative calculations using a local search based on the Adaptive Search Pattern (APS) method. An automatic adjustment is inserted into the DC-GRASP in order to define the parameters in high-dimensional functions, using other meta-heuristic, the Particle Swarm Optimization (PSO). For functions with few dimensions, a step size ampliation mechanism on APS and the use of inexact linear search has been proposed to improve the use of iteration methods.

To validate the method implemented, the last adaptive implementations Continuous GRASP found in recent literature were used, as well as in original versions published in early articles using meta-heuristic. Some computational experiments were performed in a benchmark test of functions in an known global minimum, thus proving the effectiveness of the method for aiding in convergence.

Keywords: Continuous Optimization, Continuous GRASP, automatic adaptation, meta-heuristics, Particle Swarm Optimization.

Agradecimentos

Durante os dois anos de Mestrado todas as conquistas alcançadas são devidas ao estímulo e motivações que recebi. Por isto venho agradecer:

Ao meu grandioso Deus, o maior responsável por tudo que aconteceu até hoje na minha vida, o Autor da minha fé e que me dá forças para caminhar e continuar firme em suas promessas.

A minha esposa Ada Karine que tanto me apoiou, me suportando nos momentos difíceis, me consolando e levantando a minha cabeça quando um dia pensei em desistir de tudo. A ela retribuo todo o meu amor e carinho por esta conquista.

A família da minha esposa que foi de extrema importância, como alicerce na minha vida.

Ao meu tão querido orientador e amigo Lucídio Cabral, por todas as oportunidades a mim ofertadas e o tempo dedicado a minha trajetória acadêmica, fazendo com que todas as dificuldades fossem superadas da melhor forma possível. Sou eternamente grato a Ele.

Ao Eduardo Queiroga que foi uma peça fundamental para que esse trabalho fosse concluído com sucesso, e pelas várias horas em laboratório e *Hangouts* que fizemos para evoluir a pesquisa.

A Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), por me ajudar financeiramente durante o período do Mestrado.

A todos que por algo me ajudaram, meus sinceros agradecimentos.

Conteúdo

Lista de Abreviaturas

APS	:	Adaptive Pattern Search
C-GRASP	:	Continuous GRASP
C-GRASP-DE	:	Continuous GRASP Differencial Evolution
CLUS	:	Collective Local Unimodal Search
CPU	:	Central Processing Unit
DC-GRASP	:	Directed Continuous GRASP
DC-GRASP SA	:	Self-Adaptive Directed Continuous GRASP
DE	:	Differencial Evolution
GNU	:	GNU is not Unix
GPU	:	Graphics Processing Unit
GRASP	:	Greedy Randomized Adaptive Search Procedure
GSO	:	Galactic Swarm Optimization
ITS	:	Iterated Tabu Search
NAPS	:	New Adaptive Pattern Search
POGC	:	Problemas de Otimização Global Contínua
PSO	:	Particle Swarm Optimization
RCL	:	Restricted Candidate List
SA	:	Simulated Annealing
SADE	:	Self-Adaptive Differential Evolution
VNS	:	Variable Neighborhood Search
VND	:	Variable Neighborhood Descent

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

INTRODUÇÃO

1.1 Motivação

Comumente pesquisadores têm a necessidade de resolver problemas de otimização de alto grau de complexidade, como o de escalonamento de tarefas, alocação de pessoas e transporte. A solução desses problemas requer o desenvolvimento de algoritmos que produzam soluções com qualidade e que sejam obtidas com custo computacional reduzido.

A otimização é uma área da computação, da engenharia e da matemática, que tem o objetivo de encontrar valores ótimos, minimizando ou maximizando uma função objetivo definida sobre um domínio (PATRIKSSON; ANDREASSON; EVGRAFOV, 2007). Podemos enfatizar que a otimização discreta trata de variáveis de decisão discretas e otimização contínua trata de variáveis de decisão contínuas.

A dificuldade existente no tratamento dos problemas de otimização encontra-se na escolha do método mais adequado para resolvê-lo. Alguns métodos mostram-se ineficientes pois não convergem adequadamente para a obtenção de uma solução ótima. Outros tipos de dificuldades são apresentadas em métodos que utilizam cálculo de gradiente de função, para definir direções adequadas de busca que demandam alto esforço computacional.

A partir daí, pesquisadores têm intensificado o desenvolvimento de vários métodos, bem como de ferramentas baseadas em inteligência computacional denominadas meta-heurísticas, que no geral são aplicadas à problemas de natureza discreta e que têm

ganhado adaptações para tratar problemas de natureza contínua.

Algumas meta-heurísticas se destacam na resolução de problemas de natureza contínua, tais como: *Simulated Annealing*(SA), *Greedy Randomized Adaptive Search Procedure*(GRASP), o *Variable Neighborhood Search*(VNS), as quais não utilizam cálculo de gradiente, mesmo possuindo uma convergência lenta quando comparadas a métodos clássicos não lineares, são eficientes por terem a capacidade de irem além da otimalidade local, não ficando presas em ótimos locais aparentemente satisfatórios.

O estudo de métodos híbridos tem se fortificado no intuito de minimizar o esforço computacional. Desta forma este trabalho apresenta uma adaptação automática inserida em um abordagem híbrida do trabalho *Continuous* GRASP (ARAÚJO et al., 2015) para definição dos parâmetros em funções de média e alta dimensionalidade. A ideia é utilizar um tempo computacional relativamente baixo para encontrar parâmetros que ocasionaram o menor gap, a fim de acelerar a execução principal do método. Para funções com poucas variáveis, é proposto um mecanismo de variação no tamanho do passo, para melhorar o aproveitamento das iterações do método, consequentemente, acelerando a convergência do algoritmo.

Por fim, a utilização de outros procedimentos de busca local e adaptações com meta-heurísticas podem prover maior capacidade de diversificação e intensificação ao método atual. Experimentos envolvendo as adaptações para o problema foram conduzidos, comparando com o método original.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem como objetivo geral aprimorar o algoritmo *Continuous* GRASP (C-GRASP) para resolver problemas de otimização global contínua.

1.2.2 Objetivos Especificos

- Implementar uma estratégia dinâmica de atualização de parâmetros em uma abordagem híbrida do *Continuous* GRASP para definição dos parâmetros em

funções de média e alta dimensionalidade;

- Implementar uma busca linear inexata na construção do C-GRASP, para substituir o método da seção áurea;

- Avaliar o impacto da estratégia dinâmica de atualização de parâmetros, em nível de esforço computacional, medido em termos de número de chamadas, ao procedimento de avaliação da função, na busca pela solução ótima e na média do valor da função objetivo ao estabelecer um número máximo de iterações;

- Realizar uma melhoria busca local APS e adaptar a escolha de parâmetros com a meta-heurística *Particle Swarm Optimization* (PSO), para prover maior capacidade de diversificação e intensificação ao método atual aplicado à POGCs.

1.3 Estrutura da Dissertação

- O capítulo 2 apresenta toda a fundamentação teórica do trabalho, com a descrição dos conceitos básicos de todos os métodos necessários utilizados;

- O capítulo 3 contém a revisão de literatura dos métodos híbridos recentes que são utilizados para resolução de problemas de otimização global contínua;

- O capítulo 4 apresenta o mecanismo de adaptação automática de parâmetros do C-GRASP aplicado à otimização global contínua, as novas abordagens utilizadas na busca local e a utilização da busca linear inexata;

- O capítulo 5 apresenta os experimentos computacionais e os resultados obtidos;

- O capítulo 6 apresenta as considerações finais.

Capítulo 2

Fundamentação Teórica

2.1 Otimização Global Contínua

Nos dias de hoje vários problemas podem ser formulados como problemas de otimização originando modelos matemáticos que visam minimizar ou maximizar uma função objetivo, através de valores encontrados para as variáveis que otimizem o objetivo. As variáveis que estão em domínios contínuos referem-se aos problemas de otimização contínua (POGC), os quais podem ser formulados genericamente, na versão de maximização, da seguinte maneira (SALEH, 2014):

$$\text{maximize } f(x) \tag{2.1}$$

$$\text{sujeito a: } r_i(x) = 0,\ i \in I \tag{2.2}$$

$$g_j(x) \geq 0,\ j \in J \tag{2.3}$$

$$x \in S \tag{2.4}$$

em que as funções f, r_i e g_j, $\forall i \in I, \forall j \in J$ são funções reais no $\mathbb{R}^n$. Os conjuntos (2.2) e (2.3) são as restrições de igualdade e desigualdade, respectivamente. A restrição (2.4) limita x ao conjunto $S \subset \mathbb{R}^n$. Um vetor x que satisfaz a todas as restrições é chamado de *solução viável*;

Podemos afirmar que uma solução x' é um máximo local se $f(x') \geq f(x)$ para $\forall x \in V \subset S$, em que V é um conjunto de vizinhos de x'. Uma solução x^* é um *ótimo*

global (ou solução ótima) se $f(x^*) \geq f(x)$, $\forall x \in S \subseteq \mathbb{R}^n$. A Figura 2.1 ilustra ambos os casos em uma superfície de uma função não-linear.

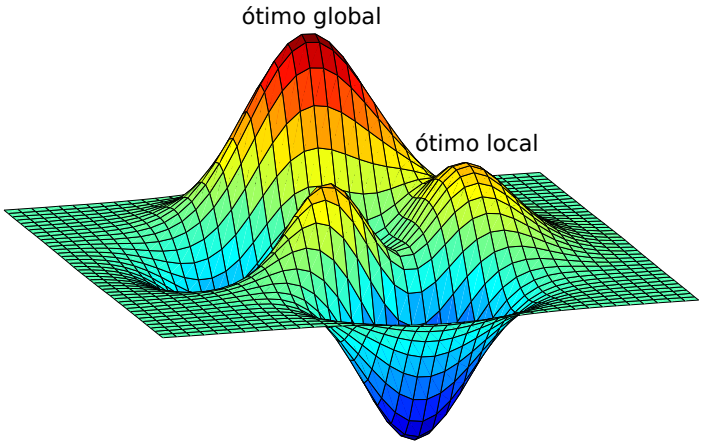


Figura 2.1: Ótimo local e global para um problema de maximização. Adaptado de (QUEIROGA, 2016)

Problemas podem ser de natureza discreta, onde neste caso se usam modelos matemáticos com programação inteira (KRAMER, 2014). Neste trabalho serão abordados problemas sem restrições, com natureza contínua através de métodos clássicos e meta-heurísticas que serão usadas para resolver problemas desta natureza.

Algumas meta-heurísticas foram inicialmente propostas para os problemas de natureza contínua, sendo depois adaptadas para os problemas de otimização de natureza discreta. O *Continuous Greedy Randomized Adaptive Search* (C-GRASP) e o *Variable Neighborhood Search* (VNS) são duas técnicas de otimização que possuem várias adaptações e novas abordagens.

2.2 Heurísticas

As heurísticas são procedimentos que servem para reduzir o espaço de busca de problemas que demandam alto custo computacional (GRONER; GRONER; BISCHOF, 2014). Grande parte desses problemas são combinatórios, sendo classificados na literatura como NP-difíceis, dado que, ainda não existem algoritmos conhecidos que os resolvam em tempo polinomial (SOUZA, 2008).

Os pesquisadores têm concentrado esforços em utilizar técnicas heurísticas para solucionar problemas com alto nível de complexidade, pois através de métodos intuitivos se consegue encontrar soluções boas a um custo computacional aceitável, entretanto sem garantir sua otimalidade, bem como garantir quão próximo está da solução ótima. O maior desafio é tentar produzir soluções tão próximas quanto possível da solução ótima, em tempo aceitável. Podemos citar como meta-heurísticas aplicadas à otimização combinatória: *Tabu Search* (GLOVER; LAGUNA, 2013), *Simulated Annealing* (DOWSLAND; THOMPSON, 2012), *Variable Neighborhood Search* (VNS) (HANSEN; MLADENOVIĆ, 2014), dentre muitas outras.

A literatura cita dois principais tipos de heurísticas, segundo (SOUZA, 2008), as heurísticas de construção e as heurísticas de refinamento (busca local):

2.2.1 Heurísticas de Construção

As heurísticas de construção são métodos que têm por objetivo claro construir uma solução, elemento por elemento, variando cada passo de acordo com a função de avaliação adotada, que por sua vez depende do problema abordado. A forma mais usada dentro do procedimento de construção é o da *escolha gulosa*, que estima o benefício da inserção de cada elemento, e somente o melhor elemento é inserido a cada passo. Outra forma é a construção aleatória, que é a forma mais fácil de se gerar uma solução inicial, a qual se escolhe os elementos candidatos aleatoriamente, isto é, o elemento a ser inserido na solução é selecionado de forma aleatória, dentre o conjunto de elementos candidatos ainda não selecionados. Nesta metodologia a implementação pode ser realizada de forma simples.

2.2.2 Heurísticas de Refinamento

As heurísticas de refinamento que também podem ser chamadas de busca local, são consideradas uma família de técnicas baseadas na noção de vizinhança. Seja $s \in S$, onde S representa um espaço de soluções possíveis, temos que $N(s)$ representa conjunto de vizinhos de s, tal que $N(s) \subseteq S$. Cada solução $s' \in N(s)$ é chamada de vizinho de s, onde cada *movimento* é denominado mudança m que transforma uma solução s em

s', pertencente a sua vizinhança. A definição geral desta classe de heurística, parte de uma solução inicial qualquer (a qual pode ser obtida por uma heurística construtiva ou gerada aleatoriamente) e prossegue, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança adotada.

2.3 Meta-heurísticas

As meta-heurísticas são consideradas um subcampo primário da otimização, as quais possuem algoritmos e técnicas que utilizam algum grau de aleatoriedade a fim de encontrar soluções tão boas quanto possível em problemas difíceis(LUKE, 2013).

De acordo com o conceito original, as meta-heurísticas são métodos de solução que coordenam procedimentos de buscas locais, com estratégias de alto nível, que criam processos capazes de ir além da otimalidade local, realizando uma busca robusta no espaço de soluções de um problema(GLOVER; KOCHENBERGER, 2006).

As meta-heurísticas também são combinações de métodos básicos heurísticos onde os conjuntos de soluções são compostos por um conjunto finito de valores discretos, assim como os métodos heurísticos(ALBA, 2005).

A seguir, serão conceituadas três meta-heurísticas bastatnte utilizadas em problemas de otimização global contínua, (veja a seção 2.1): O *Greedy Randomized Adaptive Search Procedure* (GRASP), O *Particle Swarm Optimization* (PSO) e o *Variable Neighborhood Search* (VNS).

2.3.1 *Greedy Randomized Adaptive Search Procedure* - GRASP

O *Greedy Randomized Adaptive Search Procedure* (GRASP) é um procedimento proposto por Feo (1995) e Resende (1995), onde é caracterizado como um método *multi-start* (RESENDE, 1995). Este procedimento possui duas fases, a de construção e a de busca local. A fase construtiva é realizada de forma pseudo aleatória, iterativa, analisando cada elemento gerado de forma adaptiva e gulosa, sendo responsável pela escolha destes elementos que serão usados na solução corrente. A cada iteração realizada o método construtivo gera várias soluções diferentes (FEO; RESENDE, 1995).

A função de construção é definida em g : $C \rightarrow \mathbb{R}$, onde C, corresponde a uma lista que possui elementos com características que beneficiam a função objetivo do problema tratado.

Após a fase de construção, as soluções obtidas ainda não são consideradas ótimos locais, devendo ser melhoradas ao passar pela fase de busca local. Este método também é iterativo buscando a melhor solução em sua vizinhança, sendo sua eficiência dependente das construções de cada solução, representado de forma em que partindo de uma

Algoritmo 1: Pseudocódigo GRASP

```
1 procedimento GRASP(GRASPMax, s)
2     para cada i ← 1...GRASPMax fazer
3         ConstrucaoSolucao(s);
4         BuscaLocal(s);
5         AtualizandoSolucao(s, s*);
6     retorne (s*);
```

solução inicial s, pertencente ao espaço de soluções S, é realizado um procedimento de busca local na vizinhança, que busca melhorar a solução corrente do problema, preenchendo a lista de candidatos, que, ao final da iteração, é conhecida como solução s^1. O procedimento é reiniciado na próxima iteração e se obtém a solução s^2. Este processo é realizado *GRASPMax* até que seja encontrada uma solução s^* que será considerada como a melhor solução obtida em todas as iterações.

O método é adapativo devido a escolha do grau de aleatoriedade de cada iteração. O GRASP conjuga bons aspectos dos algoritmos puramente gulosos, com aqueles dos procedimentos aleatórios de construção de soluções(SOUZA, 2008).

2.3.2 *Particle Swarm Optimization* - PSO

Otimização por Nuvem de Partículas (PSO, em inglês, *Particle Swarm Optimization*) é uma técnica de otimização estocástica, proposta por (EBERHART; KENNEDY, 1995), motivada pelo comportamento social de organismos, tais como pássaros em busca de comida. No PSO, cada partícula representa uma solução para o problema de otimização subjacente e a nuvem de partículas, de forma análoga à população de indivíduos em Algoritmos Genéticos (MELANIE, 1996), constitui um conjunto de so-

luções candidatas. Na busca pelo ótimo global, o enxame de partículas (veja a Figura 2.2) se movimenta no hiperespaço de acordo com equações que definem a próxima posição de cada partícula baseado em estratégias que consideram a melhor posição obtida por uma partícula ou *pbest* (do inglês, *personal best*) e a melhor solução do enxame *gbest* (do inglês, *global best*) encontrada até a iteração corrente. Em alguns casos, as partículas vizinhas também podem influenciar a movimentação.

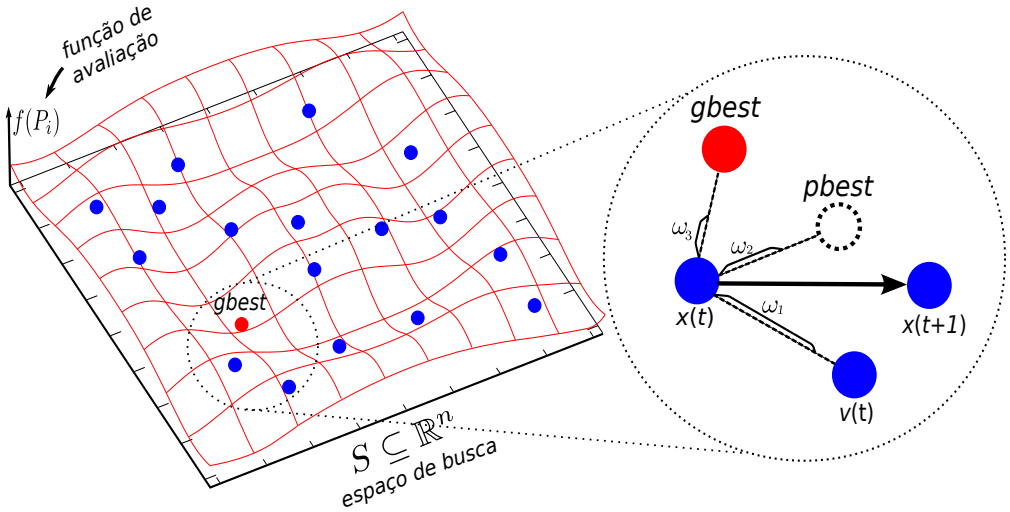


Figura 2.2: Enxame de partículas para um problema bidimensional. Adaptado de (QUEIROGA, 2016)

No PSO básico, o cálculo das componentes do vetor velocidade de cada partícula i em cada dimensão j na iteração $t+1$ é dado pela equação (2.5). O escalar $w \in [0,1]$ consiste no peso inercial que possui o papel de freio gradual do enxame em sintonia com a convergência do método. As constantes c_1 e c_2 são coeficientes de aceleração que ponderam os termos cognitivo (que utiliza o *pbest*) e social (que utiliza o *gbest*) da partícula. Os pesos $r_1, r_2 \sim U([0,1])$ são valores aleatórios distribuídos uniformemente e podem prover diversificação na movimentação do enxame. A posição de uma partícula i na iteração $t+1$ é calculada pela equação 2.6.

$$V_{ij}^{t+1} = w \times V_{ij}^{t} + c_1 \times r_1 \times (pbest_{ij} - X_{ij}^{t}) + c_2 \times r_2 \times (gbest_{ij} - X_{ij}^{t}) \tag{2.5}$$

$$X_{ij}^{t+1} = X_{ij}^{t} + V_{ij}^{t+1} \tag{2.6}$$

O PSO é descrito em alto nível pelo pseudocódigo do Algoritmo 2 e pelo diagrama de fluxo na Figura 2.3. Na versão pura, o PSO inicia com vetores de velocidade nulos. Por isso, nesta iteração, o *gbest* possui maior poder de influência do que nas iterações seguintes.

Algoritmo 2: Pseudocódigo do PSO (EBERHART; KENNEDY, 1995)

```
1  procedimento PSO
2      Inicialize o conjunto de partícula P(t);
3      repita
4          para cada partícula p ∈ P(t) fazer
5              Calcule valor aptidão;
6              se (f(p) < f(pBest)) então
7                  pBest ← p;
8          gBest ← Valor de Aptidão da melhor de todas as partículas;
9          para cada partícula p ∈ P(t) fazer
10             Calcule velocidade da partícula p;
11             Atualize posição da partícula p;
12     até Critério de parada seja satisfeito;
```

O funcionamento do PSO se inicializa com o enxame de partículas em posições aleatórias, para calcular a função objetivo de cada partícula (*pbest* inicial) e encontrar o *gbest* inicial. Feito isso, são calculados os vetores velocidade e as posições, através das equações 2.5 e 2.6, citadas anteriormente. Caso ocorra melhora no procedimento, o *pbest* e *gbest* são atualizados. Finalmente, se os critérios de término são satisfeitos, é retornado o *gbest*. Caso contrário, os vetores velocidade e posições são calculados novamente até o critério de término ser satisfeito.

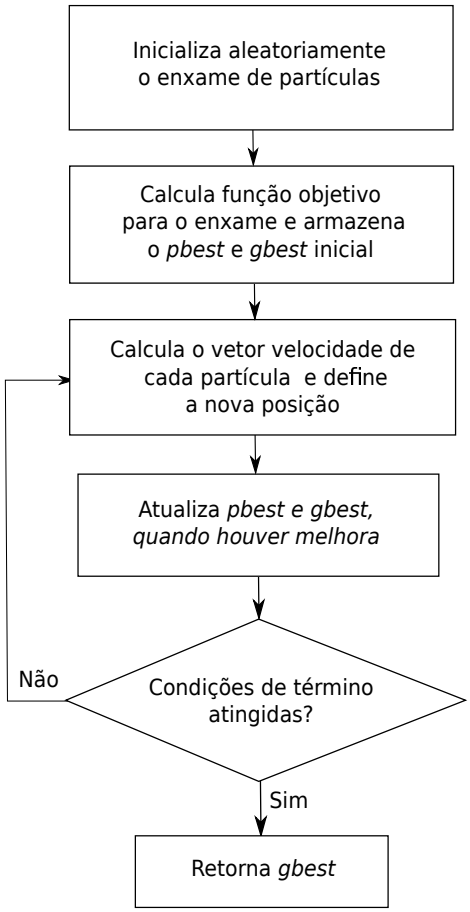


Figura 2.3: Diagrama de fluxo do PSO básico. Adaptado de (QUEIROGA, 2016)

2.3.3 *Variable Neighborhood Search* - VNS

O *Variable Neighborhood Search* (VNS) é um método que diferencia das demais meta-heurísticas pois não utiliza uma busca em um único espaço de vizinhança. A estratégia consiste em realizar variações sistemáticas da vizinhança, explorando assim regiões distintas (MLADENOVIĆ; HANSEN, 1997).

O VNS possui um conjunto finito de estruturas de vizinhanças N_k com $k = 1...n$, sendo $s^{'} \in N_k(s)$ uma solução na *k-ésima* vizinhança de s. O procedimento inicia com $k = 1$ e até que k seja diferente de n, procurando uma solução aleatória s^1, pertencente a *k-ésima* vizinhança de s, que após a busca local traga uma melhoria na função objetivo. Caso não seja obtida, k será atualizado ($k = k + 1$), caso contrário o procedimento continuará com $k = 1$, de acordo com o pseudocódigo do Algoritmo 3.

Algoritmo 3: Pseudocódigo do VNS (MLADENOVIĆ; HANSEN, 1997)

```
1  procedimento VNS(MaxItersILS, s)
2      Selecionar vizinhanças N_k, k = 1..n;
3      s ← SoluçãoInicial();
4      enquanto Critério de parada não satisfeito faça
5          k ← 1;
6          enquanto k ≤ n faça
7              s^1 ← Perturbação(N_k(s));
8              s^2 ← BuscaLocal(s);
9              se (f(s^2) < f(s)) então
10                 s ← s^2;
11                 k ← 1;
12             senão
13                 k ← k + 1;
```

Partindo de uma solução $x \in X$ é iniciado um processo de busca na região ao redor de um espaço com tamanho H, por uma nova solução x^1. Esta solução é gerada a partir de uma perturbação, onde o valor da função é verificado para as duas, sendo $f(x^1) < f(x)$. Se o valor de x^1 for considerado o melhor valor da função, o procedimento é retornado tendo como solução atual x^1. A partir daí o processo de busca local é realizado novamente, agora na próxima vizinhança H^1, em que a solução x^2 é encontrada da mesma maneira que x^1 e tem seu valor da função objetivo validado, onde $f(x^1) < f(x^1)$.

Se durante o procedimento de busca acontecer da solução corrente não ser considerada a melhor, o VNS reinicia sua operação aumentando o espaço de vizinhança $k = 1$, e continua o procedimento de busca, anteriormente visto, a fim de encontrar uma nova solução melhor. O processo continua apenas se um movimento de melhora for realizado, se não, serão realizadas trocas sistemáticas de estruturas de vizinhança, até que o critério de parada seja satisfeito, ou o número total de iterações seja alcançado, $k = n$.

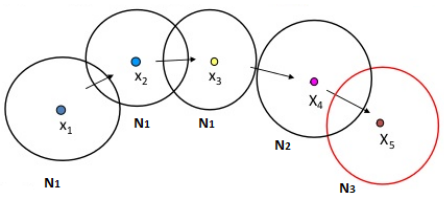


Figura 2.4: Procedimento VNS na exploração do espaço de soluções S.

O método possui um procedimento de perturbação, também conhecido como *shaking*, que é realizado durante a etapa de diversificação da busca, responsável por realizar movimentos exploratórios maiores e mais intensos em regiões distintas (CARRIZOSA et al., 2012). Desta forma, o procedimento pode realizar uma piora na qualidade da solução, porém provê um bom mecanismo de fuga de ótimos locais e uma exploração mais abrangente do espaço de soluções.

2.3.4 *Variable Neighborhood Descent* - VND

O método de descida VND também foi proposta por Mladenović (1997) e Hansen (1997) e tem como principal objetivo escapar de ótimos locais, pois realiza uma exploração sistemática de diferentes vizinhanças em uma solução. Ao considerar k_{max} estruturas de vizinhanças $N^{(k)}(k = 1 \dots k_{max})$ e uma solução corrente s, o procedimento é iniciado com $k = 1$ e um *loop* é executado enquanto $k \neq k_{max}$. Para cada k, os possíveis movimentos são avaliados. Caso haja melhora, a solução obtida passa a ser a solução corrente e k é reiniciado em 1. Assim que houver toda a exploração de uma vizinhança k e não encontrar nenhum movimento de melhora na solução corrente, k é incrementado. Abaixo o pseudocódigo do procedimento VND para um problema de maximização é apresentado no algoritmo 4.

Algoritmo 4: Pseudocódigo do VND (MLADENOVIĆ; HANSEN, 1997)

```
1  procedimento VND
2      s_0 = Solução Inicial
3      s ← s_0
4      k ← 1
5      enquanto k ≤ k_max faça
6          Melhor vizinho s' ∈ N^(k)
7          se (f^(s') > f(s)) então
8              s ← s'
9              k ← 1
10         senão
11             k ← k + 1
12     retorne s
```

Capítulo 3

Revisão da Literatura

A literatura em problemas de otimização global contínua é extremamente ampla, devido a alta variabilidade do problema em termos de restrições e critérios de qualidade. Portanto, nesta seção será apresentada uma breve revisão da literatura para métodos de otimização aplicados à otimização global contínua, (veja a seção 2.1), citando novas abordagens heurísticas propostas na literatura no uso extenso de meta-heurísticas adaptativas.

3.1 *Continuous* GRASP

O *Continuous Greedy Randomized Adaptive Search Procedure* (C-GRASP), proposto por Hirsch (2010), Pardalos (2010) e Resemnde (2010), é uma adaptação da meta-heurística GRASP (FEO; RESENDE, 1995) para resolução de problemas de otimização contínua, uma vez que o método convencional é utilizado para resolver problemas de otimização discreta. A ideia principal do algoritmo consiste em realizar uma estratégia *multi-start* sobre uma fase de construção e busca local. A primeira fase combina aleatoriedade com um critério guloso para prover diversificação e gerar boas soluções de partida que devem ser refinadas pela fase posterior. Para o C-GRASP, a tarefa de otimização associada pode ser definida como encontrar o ponto ótimo x^* com relação a uma função $f : \mathbb{R}^n \mapsto \mathbb{R}$ em um hiperetângulo $S = \{x \in \mathbb{R}^n : l \leq x \leq u\}$, onde $l, u \in \mathbb{R}^n$ tal que $u \geq l$. O algoritmo explora a discretização da região em uma grade com densidade crescente sem realizar cálculos de gradiente, que custam caro

computacionalmente para algumas funções.

O pseudocódigo do C-GRASP para um problema de minimização é descrito no Algoritmo 5, com os seguintes parâmetros de entrada: n é o número de dimensões do problema, l e u são os vetores *lower bound* e *upper bound* que determinam as restrições em caixa (*box constraints*) do espaço de busca, $f(.)$ é a função objetivo, *MaxIters* consiste no número de iterações no procedimento *multi-start*, *MaxNumIterNoImprov* determina o número de iterações em que o método busca uma melhor solução sem reduzir o tamanho do passo h, *NumTimesToRun* determina o número de execuções do procedimento, *MaxDirToTry* o número máximo de direções que a busca local irá analisar, e o α determina o balanceamento entre aleatoriedade e escolha gulosa na fase de construção. Para cada iteração do lado na linha 3, o método é realizado para um ponto inicializado aleatoriamente com distribuição uniforme (linha 4) e tamanho do passo h inicialmente igual a 1, sendo este último responsável pela densidade da grade a ser explorada. No *multi-start* interno (linha 5), a fase de construção *Construção* é executada sobre o x seguido pelo procedimento *BuscaLocal* que recebe o x resultante da fase anterior. Em seguida a melhor solução encontrada x^* é atualizada quando houver melhora (linhas 8 e 9). O tamanho do passo h é reduzido pela metade (aumentando a densidade da grade), sempre que houver *NumIterNoImprov* iterações sem melhora (linhas 12 e 13).

Algoritmo 5: C-GRASP Original (HIRSCH; PARDALOS; RESENDE, 2010)

```
1  procedimento C-GRASP(n, l, u, f(.), MaxIters, MaxNumIterNoImprov,
   NumTimesToRun, MaxDirToTry, α)
2      x* ← ∅;
3      f* ← ∞;
4      para cada j = 1, ..., NumTimesToRun fazer
5          x ← UnifRand(l, u);
6          h ← 1;
7          NumIterNoImprov ← 0;
8          para cada Iter = 1, ..., MaxIters fazer
9              x ← Construção(x, f(.), n, h, l, u, α);
10             x ← BuscaLocal(x, f(.), n, h, l, u, MaxDirToTry);
11             se f(x) < f* então
12                 x* ← x;
13                 f* ← f(x);
14                 MaxNumIterNoImprov ← 0;
15             senão
16                 NumIterNoImprov ← NumIterNoImprov + 1;
17             se NumIterNoImprov ≥ MaxNumIterNoImprov então
18                 h ← h/2;
19                 NumIterNoImprov ← 0;
20     retorne x*;
```

3.1.1 Etapa de Construção

Na fase de construção do C-GRASP, como mostra o Algoritmo 6, um ponto x é tomado como entrada e uma busca linear (linha 8) é realizada para cada coordenada não fixada (conjunto C). Inicialmente todas as coordenadas não estão fixadas e para cada coordenada i é realizada uma busca linear mantendo-se as outras coordenadas inalteradas. As soluções resultantes z_i das buscas lineares são armazenadas, bem como a melhor (min) e pior (max) soluções decorrentes dessa etapa (linhas 4-11). A Lista Restrita de Candidatos (RCL, do inglês *Restricted Candidate List*) consiste em um conjunto de soluções que respeitam um limiar definido pelo parâmetro α e as soluções min e max obtidas pela etapa anterior (linhas 12-15). Finalmente uma solução é escolhida aleatoriamente da RCL para atualizar a j-ésima coordenada de x, sendo esta coordenada fixada para as próximas iterações (linha 16). O procedimento se repete até que todas as coordenadas sejam fixadas ($C = \emptyset$).

Algoritmo 6: Procedimento de construção do C-GRASP original (HIRSCH et al., 2007)

```
1  procedimento Construção(x, f(.), n, h, l, u, α)
2      C ← {1, 2, ..., n};
3      enquanto C ≠ ∅ faça
4          min ← +∞;
5          max ← −∞;
6          para cada i = 1, ..., n fazer
7              se i ∈ C então
8                  z_i ← BuscaLinear(x, h, i, n, f(.), l, u);
9                  g_i ← f(z_i);
10                 se min > g_i então min ← g_i ;
11                 se max < g_i então max ← g_i ;
12         RCL ← ∅;
13         para cada i = 1, ..., n fazer
14             se i ∈ C and g_i ≤ (1 − α) ∗ min + α ∗ max então
15                 RCL ← RCL ∪{i};
16         j ← SelecionaElementoAleatoriamente(RCL);
17         x_j ← z_j;
18         C ← C \ {j};
19     retorne x*;
```

3.1.2 Etapa de Busca Local

Na fase de busca local, ilustrada no algoritmo 7, busca-se melhorar a solução por explorar o conjunto de direções $\Gamma = \{d \in \mathbb{R}^n : d_i = -1 \vee d_i = 0 \vee d_i = 1,\ i = 1, ..., n\} \setminus \{d \in \mathbb{R}^n : d_i = 0, i = 1, ..., n\}$ que consiste em todas as possibilidades de vetores com $\{-1, 0, 1\}$ em cada coordenada, exceto o vetor nulo. Por exemplo, para $n = 2$, temos que $\Gamma = \{(0,1), (0,-1), (1,1), (1,0), (1,-1), (-1,1), (-1,0), (-1,-1)\}$, como ilustrado no exemplo da Figura 3.1. A cardinalidade do conjunto de direções é dada por $3^n - 1$ para n dimensões, podendo-se realizar um mapeamento bijetivo $T : \{1, ..., 3^n - 1\} \mapsto \Gamma$. Dado um ponto x de entrada, o procedimento basicamente gera *NumDirToTry* (linha 4) direções aleatórias através do mapeamento T e movimenta x com um passo h na direção candidata (linha 10), repetindo o mecanismo sempre que houver melhora.

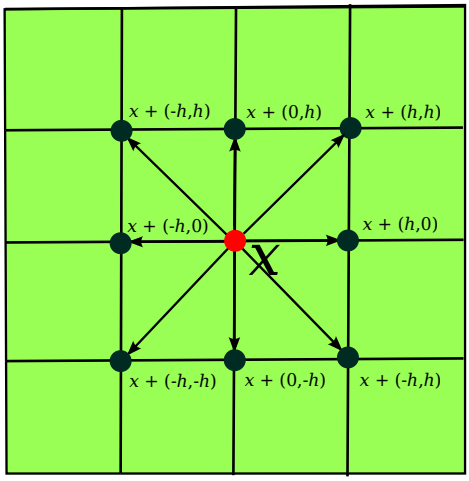


Figura 3.1: Direções de busca possíveis para um problema bidimensional. Adaptado de (QUEIROGA, 2016)

Algoritmo 7: Procedimento de busca local do C-GRASP original (HIRSCH et al., 2007)

```
1  procedimento BuscaLocal(x, n, l, u, f(.), MaxDirToTry)
2      Improved ← true;
3      D ← ∅;
4      x* ← x;
5      f* ← f(x);
6      NumDirToTry ← min{3^n − 1, MaxDirToTry};
7      enquanto Improved faça
8          Improved ← false;
9          enquanto |D| ≤ NumDirToTry E not Improved faça
10             Gere r ← ⌈UnifRand(1, 3^n − 1)⌉ ∉ D;
11             D ← D ∪ {r};
12             d ← T(r);
13             x ← x* + h ∗ d;
14             se l ≤ x ≤ u então
15                 se f(x) ≤ f* então
16                     x* ← x;
17                     f* ← f(x);
18                     D ← ∅;
19                     Improved ← true;
20     retorne x*;
```

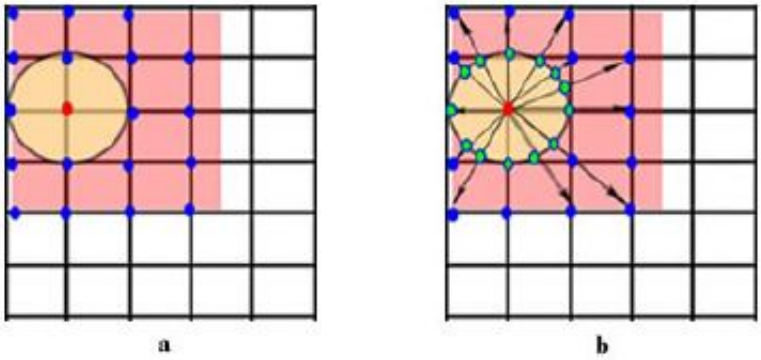


Figura 3.2: Representação da vizinhança gerada na busca local do C-GRASP. Adaptado de (ANDRADE, 2013).

3.2 C-GRASP com construção paralela

Muitas tecnologias têm evoluído e alcançado novos níveis de aplicabilidade e poder computacional, que ajudam a solucionar problemas de Otimização, problemas de arquiteturas paralelas, assim como o processamento de vídeos e processamento vetorial.

Observando o C-GRASP original proposto por Hirsch et al. (2007) na seção 3.1, é verificado que a meta-heurística expõe certa dificuldade de convergência, à medida que a obtenção de uma solução, que concederia à função objetivo uma melhora, não ocorre, pois a grade projetada com limites inicial e final torna-se mais densa a cada passo, e esta característica torna-se evidente a medida em que as funções utilizadas possuem alta dimensionalidade. Com o objetivo de minimizar este esforço computacional, foi proposto uma nova abordagem para a construção (ANDRADE et al., 2014) do procedimento C-GRASP.

De acordo com Andrade et al. (2014), utilizando funções bidimensionais, constatou-se que menos de 10 % do tempo total gasto foi atribuído ao procedimento de construção, enquanto que para funções com n=30 dimensões, mais de 40 % do tempo total é dedicado. Desta maneira, a fase de construção é revelada como uma parte crítica da meta-heurística C-GRASP para resolução de problemas com grandes dimensões.

As GPUs apresentam uma solução de hardware de baixo de custo e alto poder computacional disponível para solucionar problemas. A paralelização do procedimento de construção do C-GRASP padrão foi proposta devido a seu baixo desempenho pro-

movido à medida que ocorre um aumento na dimensionalidade das funções utilizadas. No entanto, poucos algoritmos podem ser diretamente traduzidos para execuções em GPUs sem exigirem mudança em sua estrutura. Deste modo, são necessários cuidados para obtenção de qualquer ganho de eficiência quando utilizado este tipo de arquitetura paralela. Então, para garantir boa eficácia em GPUs, foi necessário organizar as estruturas de dados do algoritmo, adaptando sua arquitetura para GPU.

A parte de alto custo na fase de construção C-GRASP padrão, Algoritmo 6, é o laço entre as linhas 5 e 11, onde uma pesquisa linear é executada em cada direção sobre as coordenadas não-fixas. O algoritmo paralelo poderia ser apenas uma versão paralela deste loop, mas a gestão das coordenadas não-fixas de CPU para GPU promoveria grande sobrecarga (*overhead*).

Desta maneira, a maior parte do processo de construção é feito na GPU. O procedimento *ConstrucaoParalela*, presente no algoritmo 8, promove primeiro uma cópia dos vetores importantes da CPU para a memória da GPU, que inicializa a lista de coordenadas não-fixas na memória da GPU (chamando um núcleo de inicialização) e, finalmente, chama o procedimento de construção em GPU, algoritmo 9, com n *threads*. Ao final, o vetor resultante x é copiado para a memória de CPU que continuará o processo do C-GRASP padrão em CPU. Nota-se que os vetores dos limites l e u não mudam durante o processo de otimização C-GRASP, por isso pode ser copiado uma única vez para a memória da GPU, como na linha 2 do algoritmo 8.

Algoritmo 8: Construção Paralela C-GRASP (ANDRADE et al., 2014)

```
1 procedimento ConstrucaoParalela(x, f(.), n, h, l, u, Impr_c)
2     Copia x, l and u para memória GPU;
3     Inicialização nFixa em GPU;
4     ConstrucaoKernel ≪n≫ x, f(.), n, h, l, u;
5     Copia resultados x da memória GPU para memória principal;
6     retorne (x, Impr_c);
```

O procedimento *ConstrucaoKernel* chamado pelo algoritmo *ContrucaoParalela* é mostrado no Algoritmo 9. O *kernel* gerencia as coordenadas não-fixas e chama o *BuscaLinearKernel*, para cada thread, utilizando seu índice como parâmetro. O procedimento *BuscaLinearKernel* verifica as coordenadas não-fixas, realizando uma busca linear à procura de uma solução que ofereça o mínimo valor para a função

objetivo; como um detalhe de implementação, a lista de coordenadas não-fixas é armazenada como um vetor de booleanos.

Algoritmo 9: Procedimento de Construção para C-GRASP no kernel (ANDRADE et al., 2014)

```
1  procedimento ConstrucaoKernel ≪n≫ (x, f(.), n, h, l, u)
2      ReUse ← falso;
3      enquanto nFIxa ≠ ∅ faça
4          g ← +∞;
5          ḡ ← +∞;
6          BuscaLinearKernel(threadIdx, x, h, i, n, f(.), l, u, nFixa);
7          LCR ← ∅;
8          threshold ← g + α(ḡ − g);
9          para cada i = 1 ··· n fazer
10             se i ∈ nFixa E g_i < threshold então
11                 LCR ← LCR ∪ i;
12         j ← ElementoAleatorio(LCR);
13         se x_j = z_j então
14             ReUse ← falso;
15         senão
16             x_j ← z_j;
17             ReUse ← falso;
18             impr_c ← verdadeiro;
19         nFixa ← nFixa j;
20     retorne (x, Impr_c);
21
```

3.3 *Directed* C-GRASP

Para tratar os problemas de intensificação do C-GRASP, foi proposta uma hibridização dele com uma fase de busca local baseada em *Adaptive Pattern Search* (APS) (HEDAR; FUKUSHIMA, 2006), e essa abordagem foi chamada de *Directed Continuous* GRASP (DC-GRASP).

3.3.1 *Adaptive pattern search*

O método APS gera direções de descida de uma solução de referência por meio de n direções paralelas aos eixos coordenados, gerando um conjunto $T = \{y\}_{i=1}^{n}$ de pontos

com tamanho de passo h. Através de T a direção é obtida da seguinte forma:

$$d = \sum_{i=1}^{n} w_i v_i, \ onde \tag{3.1}$$

$$w_i = \frac{\Delta f_i}{\sum\limits_{i=1}^{n} |\Delta f_i|}, \ i = 1, 2, ..., n, \tag{3.2}$$

$$v_i = -\frac{y_i - x}{||y_i - x||}, i = 1, 2, .., n, \tag{3.3}$$

$$\Delta f_i = f(y_i) - f(x) \tag{3.4}$$

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

A Figura 3.3 ilustra o procedimento APS em duas dimensões, em que, dados os pontos y_1 e y_2 responsáveis por gerar uma direção de descida, assume-se que x é a melhor solução entre y_1 e y_2 (Figura 3.3_a). O procedimento realiza uma busca com passos discretizados em h (Figura 3.3_c), gerando o ponto y_3 (Figura 3.3_d), onde o melhor ponto entre (y_1, y_2, y_3) será retornado.

Diferentemente do APS, que gera dois pontos na direção d para buscar o melhor vizinho, o *New Adaptive Pattern Search* (NAPS), proposto por Araújo et al. (2015), realiza uma busca linear em d e adiciona o melhor ponto encontrado ao conjunto T, retornando por fim, o melhor ponto em T.

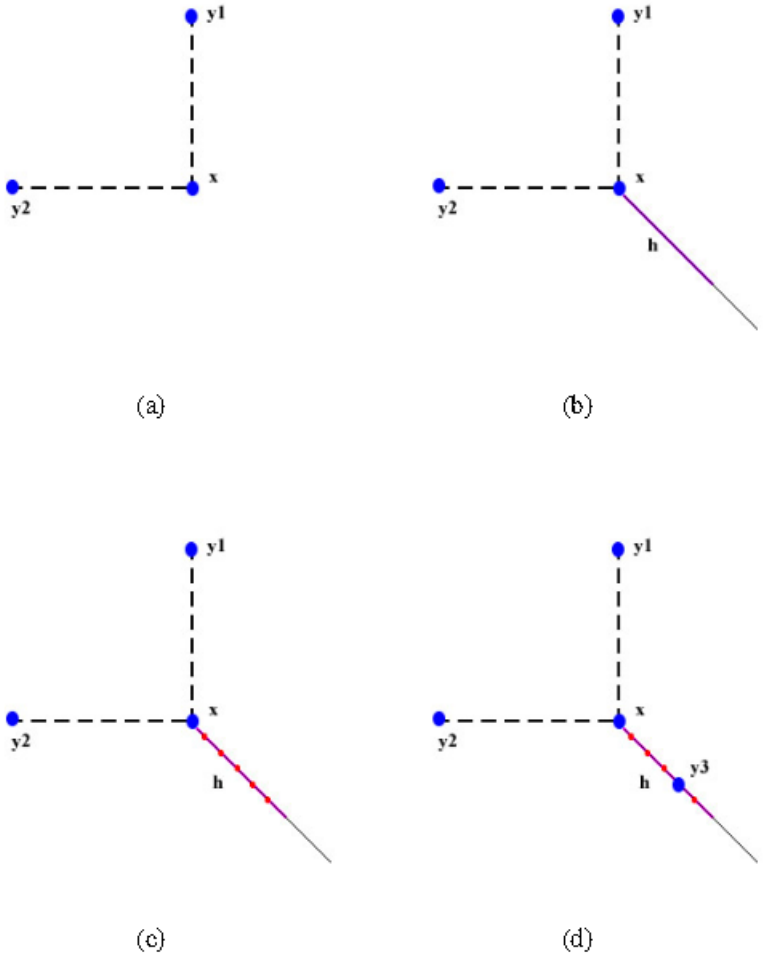


Figura 3.3: Representação do procedimento APS em duas dimensões. Adaptado de (ANDRADE, 2013).

3.3.2 Procedimento de Busca Local NAPS

O procedimento BuscaLocal_NAPS é apresentado, em pseudocódigo, no Algoritmo 10. Enquanto o número máximo de iterações não é alcançado, o NAPS é chamado e caso não ocorra melhora, uma perturbação é realizada por amostrar um vizinho com esfera de raio h do ponto x^*, como descrito na seção anterior. O DC-GRASP se caracteriza por substituir a busca local do C-GRASP pelo BuscaLocal_NAPS.

Algoritmo 10: Busca local do DC-GRASP (ARAÚJO et al., 2015)

```
1  procedimento BuscaLocal_NAPS(x, f(.), n, h, h_e, l, u, MaxLocalIters, ImprL)
2      x* ← x;
3      f* ← f(x);
4      NumIters ← 0;
5      enquanto NumIters ≤ MaxLocalIters faça
6          NumIters ← NumIters + 1;
7          x ← NAPS(x, f(.), n, h, h_e);
8          se l ≤ x ≤ u ∧ f(x) < f* então
9              x* ← x;
10             f* ← f(x);
11             ImprL ← true;
12             NumIters ← 0;
13         senão
14             x ← RandomSelectElement(B_h(x*));
15     retorne [x, ImprL];
```

3.4 C-GRASP *Differential Evolution*

Storn (1997) e Price (1997) propõem trabalhar na resolução de problemas de despacho de energia elétrica, ou seja, na otimização da quantidade de energia utilizada em setores de distribuição elétrica, utilizando métodos de otimização para minimizar o custo da distribuição e encontrar soluções ótimas. Para isso se utilizou o *continuous* GRASP, o qual sofreu uma hibridização com o *Differencial Evolution* (DE), que é uma técnica de otimização também aplicada a problemas de natureza contínua, no intuito de minimizar possíveis espaços não lineares e não diferenciáveis.

A técnica proposta possui um controle de parâmetros e uma combinação de estratégias DE. Esta gama de estratégias é utilizada na fase de busca local, Algoritmo 11, chamada de C-GRASP-DE.

Algoritmo 11: Procedimento de busca local do C-GRASP original utilizado com a escolha de parâmetros do DE (HIRSCH et al., 2007)

```
1  procedimento BuscaLocal(x, n, l, u, f(.), MaxDirToTry)
2      Improved ← true;
3      D ← ∅;
4      x* ← x;
5      f* ← f(x);
6      NumDirToTry ← min{3^n − 1, MaxDirToTry};
7      enquanto Improved faça
8          Improved ← false;
9          enquanto |D| ≤ NumDirToTry E not Improved faça
10             Gere r ← ⌈UnifRand(1, 3^n − 1)⌉ ∉ D;
11             D ← D ∪ {r};
12             d ← T(r);
13             x ← x* + h * d;
14             se l ≤ x ≤ u então
15                 se f(x) ≤ f* então
16                     x* ← x;
17                     f* ← f(x);
18                     D ← ∅;
19                     Improved ← true;
20     retorne x*;
```

A escolha apropriada de uma estratégia de mutação como o fator de mutação e a taxa de *crossover*/probabilidade requer evidências empíricas e alguma experiência. No algoritmo 11, a taxa é gerada com uma distribuição uniforme no intervalo $[0.2, 0.8]$. A seguir são adotadas duas estratégias com probabilidade igual a 50 %, cada, durante cada geração:

(1) : $DE/best/1/bin$ onde um valor é gerado com uma distribuição uniforme no intervalo $[0.3, 0.6]$ em cada geração;

(2) : $DE/best/1/bin$ onde uma matriz de valores é adotada para cada variável e gerada com uma distribuição uniforme no intervalo $[0.005, 0.4]$.

Enfim o C-GRASP-DE, proposto por (NETO et al., 2017), propõe melhorar a capacidade de pesquisa global e impedir a convergência para mínimos locais, sendo aplicado em vez da busca local do C-GRASP, Algoritmo 7.

Outra adaptação foi realizada no DE, denominada de *Self-Adaptive Differential Evolution* (SaDE) (QIN; SUGANTHAN, 2005), onde há um controle na parametrização

dos valores e na taxa *crossover*/probabilidade, onde estes parâmetros são gradualmente adaptáveis de acordo com a forma de aprendizado na performance anterior. Em outras palavras o SaDE memoriza o conhecimento da busca anterior a fim de adaptar o controle de parâmetros. Esta adaptação também é combinada com o C-GRASP e denominada de C-GRASP-SaDE.

3.5 DE-VNS *Self-Adaptive Differential Evolution*

O método DE-VNS é uma combinação de duas meta-heurísticas aplicadas a problemas de otimização contínua, baseada na junção do *Differential Evolution*, proposta por Storn (1997) e Price (1997) e o *Variable Neighborhood Search*, proposto por Mladenović (1997) e Hansen (1997). A ideia básica dessa hibridização é utilizar o mecanismo de mudança de vizinhança a fim de estimar o parâmetro *Crossover* do DE (KOVAČEVIĆ et al., 2014).

Além disso é introduzida uma nova família de distribuições adaptativas para controlar as distâncias entre as soluções no espaço de busca, bem como evidências experimentais para encontrar a melhor probabilidade da função de distribuição para o parâmetro do VNS suportado pela estimativa estatística. O pseudocódigo do DE-VNS está representado no Algoritmo 12.

Essa heurística hibridizada tem mostrado excelentes características, se tornando mais favorável do que as características do estado da arte do DE, quando testada em instâncias padrões da literatura. Todas as instâncias testadas resultaram em 100 % de convergência, onde o esforço computacional antes de encontrar o ótimo global provou ser consistentemente menor que o esforço obtido em outros métodos do estado da arte, destacando-se em problemas de alta dimensão.

Algoritmo 12: Pseudocódigo do DE-VNS (KOVAČEVIĆ et al., 2014)

procedimento DE-VNS($n, F = [0.4, 0.6, 0.8, 1], h = 4, par_{max} = 0.7, par^{(0)} = par_{min}, n_0 = 2, \delta = 0.05,$ *stepfactor* $= 1/(10Dlog_2(D))$)
  Inicializa randomicamente uma população ou n indivíduos $P_g = \{X_1^{(0)} \ldots, X_n^{(0)}\}$
  Avalia a população, encontra-se $f(x_i^{(0)})$, $i = 1, \ldots, n$
  Configurando a probabilidade do roleta inicial para o parâmetro de mutação $f^{(0)}$
  $g \leftarrow 1$
  enquanto *Os critérios de parada não são cumpridos* **faça**
    para cada $i = 1 : n$ **fazer**
      Calcule a probabilidade da roleta $p_h = n_h + n_0/\Sigma(n_j + n_0)$,
      $h = 1.2, \ldots, h$ *e obtém* $f_i^{(g)}$
      Amostrando $CR_i^{(g)}$ *da distribuição TSP adaptativa como*
      $CR_i^{(g)} \leftarrow 1 - (1 - x)^{1/par(g)}, x \sim$ *Unif* (0,1)
      Aplicando a estratégia "DE/Rand - Local - Best/1/bin"com o parâmetro obtido
      $f_i^{(g)}$
      Aplicando o crossover com o parâmetro obtido $CR_i^{(g)}$
      Avalia o vetor $f(u_i^{(g)})$
      se $f(u_i^{(g)}) < f(x_i^{(g)})$ **então**
        $par^{(g+1)} \leftarrow max(par_{min}, par^{(g)} - (f(u_i^{(g)}) - f(x_i^{(g)})))$
        $x_i^{(g+1)} \leftarrow u_i^{(g)}$;
      senão
        $par^{(g+1)} \leftarrow min(par^{(g)} +$ *stepfactor*, $par_{max})$
    $g \leftarrow g + 1$;

3.6 Hibridização de um Algoritmo Baseado em Computação Evolucionária

Bashir (2012) e Neville (2012) propõem uma novo algoritmo hibridizado para resolver problemas de otimização global contínua de baixa dimensão, e também de grande escala. Este método compreende um algoritmo genético (GOLDBERG; JOHN, 1988),

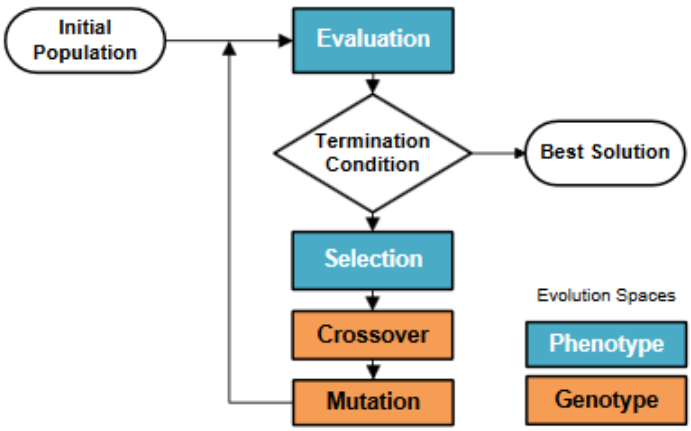


Figura 3.4: Fluxo de um algoritmo genético típico. (BASHIR; NEVILLE, 2012).

caracterizando uma estratégia elitista adaptativa e um algoritmo de programação sequencial e quadrático. Este algoritmo combina um portfolio colaborativo com uma fase de validação.

O algoritmo de programação quadrática sequencial padrão (SQP) é um algoritmo de otimização linear baseada em Newton, que pode lidar com problemas não-lineares restritos. Tipicamente, os algoritmos SQP compreendem uma linearização externa e um interior Loop de otimização.

Considerando a expressão geral, na equação 3.5, de um problema de otimização contínua:

$$\text{minimize}(x) : f \in \mathbb{R}^n; n \geq 1, \tag{3.5}$$

Resultados experimentais justificam que a hibridização de algoritmos evolucionários com um método de busca local, pode ser robusto e eficiente para poder produzir meios de resolução para problemas de otimização contínua.

O pseudocódigo 13 mostra a hibridização de um algoritmo genético e um algoritmo quadrático sequencial.

Algoritmo 13: Hibridização EC/SQP (BASHIR; NEVILLE, 2012)

```
1 procedimento EC/SQP()
2     t ← 0;
3     inicialize P_1(t) : size = N
4     Busca Global : x_EC ← invoke EC(P_1(t))
5     Busca Local : x_SQP ← invoke SQP(x_EC)
6     re-inicialize P_2(t) : size = N − 2
7     valide x* ← invoke EC(P_2(t) ∪ x_SQP ∪ x̄_SQP)
```

3.7 *Galactic Swarm Optimization*

Muthiah-Nakarajan (2016) e Noel (2016) propõem uma nova meta-heurística chamada *Galactic Swarm Optmization*, que é inspirada no movimento das estrelas, galáxias e *superclusters* das galáxias, sob a influência da gravidade. O GSO emprega ciclos exploratórios múltiplos explorando a existência de soluções, fazendo trocas entre a as novas soluções e as soluções já existentes.

O GSO é baseado no PSO (EBERHART; KENNEDY, 1995), meta-heurística que é inspirada no comportamento do sobrevôo de pássaros.

A fase de busca do GSO traz as melhores soluções de diferentes subpopulações, semelhante ao comportamento das super partículas que são atualizadas no algoritmo do PSO. Resultados estatísticos indicam que o algoritmo GSO propõe performances significantes, comparadas a quatro versões encontradas no estado da arte do algoritmo PSO, também a quatro super partículas do PSO aplicadas em 15 problemas *benchmark* de otimização, utilizando mais de cinquenta ensaios independentes em problemas acima de cinquenta dimensões. Todas as simulações mostram que o algoritmo GSO, neste trabalho, converge rapidamente numa variedade de problemas de alta dimensão.

Algoritmo 14: *Galactic Swarm Optimization* (MUTHIAH-NAKARAJAN; NOEL, 2016)

procedimento GSO(f)
  Inicialização nível 1:
  $x_j^{(i)},\ v_j^{(i)},\ p_j^{(i)},\ g_j^{(i)}$ *em* $[x_{min}, x_{max}]^D$ *randomicamente*
  Inicialização nível 2:
  $v^{(i)},\ p^{(i)},\ g$, *em* $[x_{min}, x_{max}]^D$ *randomicamente*
  para cada $EP \leftarrow$ ***para*** EP_{max} **fazer**
    Iniciar PSO: nível 1
    para cada $i \leftarrow$ ***para*** M **fazer**
      para cada $k \leftarrow 0$ ***para*** L_1 **fazer**
        para cada $j \leftarrow 1$ ***para*** N **fazer**
          $v_j^{(i)} \leftarrow \omega_1 v_j^{(i)} + c_1 r_1 (p_j^{(i)} - x_j^{(i)}) + c_2 r_2 (g^{(i)} - x_j^{(i)})$;
          $\mathrm{x}_j^{(i)} \leftarrow x_j^{(i)} + v_j^{(i)}$;
          se $f(x_j^{(i)}) < f(p_j^{(i)})$ **então**
            $p_j^{(i)} \leftarrow x_j^{(i)}$;
            se $f(p_j^{(i)}) < f(g^{(i)})$ **então**
              $g^{(i)} \leftarrow p_j^{(i)}$;
              se $f(g_j^{(i)}) < f(g)$ **então**
                $g \leftarrow g^{(i)}$;
    Iniciar PSO: nível 2
    Inicialize Swarm $y^{(i)} = g^{(i)} : i = 1, 2, \ldots, M$;
    para cada $k \leftarrow 0$ ***para*** L_2 **fazer**
      para cada $i \leftarrow 1$ ***para*** M **fazer**
        $v^{(i)} \leftarrow \omega_2 v^{(i)} + c_3 r_3 (p^{(i)} - y^{(i)}) + c_4 r_4 (g - y^{(i)})$;
        $\mathrm{y}^{(i)} \leftarrow y^{(i)} + v(i)$;
        se $f(y^{(i)}) < f(p^{(i)})$ **então**
          $(p^{(i)}) \leftarrow (y^{(i)})$;
          se $f(p^{(i)}) < f(g)$ **então**
            $g \leftarrow p^{(i)}$;
  retorne $g,\ f(g)$

3.8 *Iterated Tabu Search and Variable Neighborhood Descent* (ITS-VND)

Zeng et al. (2016) propõem uma hibridização entre a meta-heurística *Iterated Tabu Search* e o *Variable Neighborhood Descent*, (ITS-VND) para um problema contínuo de empacotamento de círculos desiguais, em um contêiner circular (PUCC). O algoritmo adapta o método de Busca Tabu dos algoritmos de Busca Tabu Iterativa e propõe um procedimento de Busca Tabu com VND (TS-VND).

O procedimento de Busca Tabu tradicional é então transformado em um método híbrido composto de duas partes alternativas, chamado de VND e Busca Tabu, respectivamente. Além deste processo remodulado, o ITS-VND também incorpora outras características novas, tais como uma função de avaliação adaptável, o critério de "acidentes de colisão",

Algoritmo 15: Hibridização proposta com ITS-VND, Busca Tabu Iterativa e VND (ZENG et al., 2016)

```
1  procedimento ITS-VND()
2      Entrada: the numbers of circles n; the radio of the n circles
3      Saída: A melhor solução encontrada durante o processo;
4      R ← sqrt(2 Σ_{i=1}^{N} r_i^2);
5      X ← Repor aleatoriamente os círculos n no contêiner;
6      TempoDecolisão ← 0;
7      do
8      ITS-VND (Início do processo)
9      X' ← Perturbe X com uma força de perturbação escolhida aleatoriamente do
       intervalo, [1, n/6];
10     Estratégia de perturbação, Fase 3
11     (R'', X'') ← TS-VND(R, X');
12     se (R'', X'') é melhor que (R, X) então
13         (R, X) ← (R'', X'');
14         TemposDeColisão ← 0;
15         senão
16             se (R'', X'') é o mesmo que (R, X) então
17                 TemposDeColisão ← TemposDeColisão +1;
18         enquanto TemposDeColisão < LimiteDeColisão faça
19             retorne (R, X);
```

e um novo método para acelerar a exploração da vizinhança, e avaliar quão intensivamente a área perto da solução atual tem sido explorada.

Os resultados computacionais em três conjuntos de referência bem estabelecidos, mostram que o algoritmo proposto não só tem uma boa capacidade de descoberta mas também pode fornecer bons resultados dentro de um prazo razoável, de um total de 84 instâncias *benchmark* executadas.

3.9 VNS Gaussiano para otimização contínua

Carrizosa et. al (2012) propõem uma abordagem para resolução de problemas contínuos baseado na meta-heurística VNS, desenvolvida por Mladenović (1997) e Hansen (1997). O VNS mostra ser uma ferramenta poderosa para resolver problemas de otimização contínua discretos e de caixa. Esta metodologia também é aplicada para resolver problemas de otimização contínua sem restrições.

O método se inicia com perturbação da solução incumbente, adicionando um pouco de ruído e seguindo uma distribuição Gaussiana em vez de perturbar a solução incumbente, gerando aleatoriamente um ponto em uma esfera de um determinado tamanho. A geração de pontos é feita de forma intuitiva, de preferência para alguma direção no espaço de busca, ou de forma contrária, para tratar uniformemente todas as direções. Em outras palavras, no processo de perturbação o ponto é gerado através de um vetor y n-dimensional aleatório com função de densidade na forma, equação 3.6.

$$\varphi(y) = \frac{1}{(2\pi)^{n/2}|\Sigma_k|^{1/2}} \varrho^{-1/2(y-x)^T \Sigma_k^{-1}(y-x)} \quad (3.6)$$

Os experimento realizados comprovam através de resultados computacionais, que existem vantagens para a nova abordagem. O funcionamento do método, chamado de GAUSS-VNS, está representado no pseudocódigo 16.

Algoritmo 16: VNS Gaussiano (CARRIZOSA et al., 2012)

procedimento `Gauss-VNS()`
  Selecione o conjunto de matrizes de covariância $\Sigma_k, k = 1, \ldots, k_{max}$
  Escolha um ponto inicial arbitrário $x \in S$
  set $x^* \leftarrow x, f^* \leftarrow f(x)$
  repeat
    os seguintes passos
  until;
  A condição de parada seja satisfeita
  set $k \leftarrow 1$
  repeat
    os seguintes passos
  until;
  $k > k_{max}$
  Perturbe: *Gerar y de uma distribuição gaussiana com média* x^* *e Covariância matricial* Σ_k
  Aplicar busca local de y
  se $f(y') < f^*$ **então**
    set $x^* \leftarrow y^{'}$
    f* $\leftarrow f(y)$
    and goto *linha 5*
  set $k \leftarrow k + 1$
  stop. x^* *é uma solução aproximada do problema.*

3.10 *Improved Particle Swarm Optimization with collective Local Unimodal Search*(PSO-CLUS)

Arasomwan (2014) e Adewumi (2014) propõem uma nova técnica de busca local usada na melhoria da *performance* dos algoritmos PSO, resolvendo problemas de convergência.

Na técnica de busca local proposta, uma posição de uma partícula com alto potencial no espaço de busca é construída de forma coletiva por um número de partículas selecionadas randomicamente no enxame. O número de vezes que é feita a seleção varia de acordo com a dimensão do problema de otimização e cada partícula doa o valor na localização de sua dimensão selecionada aleatoriamente de seu melhor *Pbest*. Depois de construir a posição da partícula de melhor potencial, algumas buscas locais

são realizadas em torno da sua vizinhança, comparando com a melhor posição global, *Gbest*, do enxame. A partícula é usada para substituir a posição do *Gbest* caso seja melhor, caso contrário, nenhuma substituição é feita.

Vários experimentos em problemas de benchmark conhecidos, utilizando baixa e alta dimensões, foram usadas para validar o desempenho dos melhores algoritmos. As comparações foram realizadas com quatro versões de PSO diferentes. Duas dessas versões implementam técnicas de busca local diferentes, enquanto as outras não. Os resultados apontam que os melhores algoritmos podem obter uma melhor solução de qualidade ao mesmo tempo que demonstra uma melhor convergência, velocidade, precisão, estabilidade, robustez e capacidade de busca global/local que as outras versões.

3.11 *Adaptive Large Neighborhood Search*(ALNS)

Martins (2017) propõe uma abordagem da heurística *Adaptive Large Neighborhood Search* proposta por Ropke (2006) e Pisinger (2006), que consiste numa extensão da Busca em Vizinhança de Grande Porte (*Large Neighborhood Search* - LNS), inicialmente criada por Shaw (1997). O mecanismo de busca das duas heurísticas iniciam a partir de uma solução inicial viável e operam com diferentes métodos que destroem e reparam a solução corrente. Na busca LNS um par dessas heurísticas de remoção e inserção é adotada durante toda a busca, já na ALNS várias heurísticas de remoção e inserção são utilizadas durante uma mesma busca. Um peso é usado inicialmente para cada heurística com intuito de encontrar o par de heurísticas mais eficiente para o problema. Todos os pesos são ajustados dinamicamente, baseados na eficiência de cada método ao longo da busca.

A cada iteração do pseudocódigo 16, um par de heurísticas (h_r, h_i), a de remoção e a de inserção, é selecionado de acordo com os desempenhos em iterações anteriores, dados pelas pontuações e pesos das heurísticas. Ao aplicar o par escolhido, as tarefas são removidas pela heurística de remoção h_r e reinseridas na solução pela heurística de inserção h_i. Se caso a nova solução $s^{'}$ satisfaz o critério de aceitação, a mesma substitui a solução s. Se $s^{'}$ é melhor que a melhor solução encontrada até o momento, a mesma também substitui s_{melhor}. Em seguida, as pontuações e pesos de (h_r, h_i) são

atualizados e a busca continua na próxima iteração. Esse processo é repetido até que algum critério de parada seja satisfeito.

Algoritmo 17: ALNS (MARTINS, 2017)

```
1  procedimento ALNS()
2      Gere uma solução inicial s;
3      s_melhor ← s;
4      enquanto algum critério de parada não seja satisfeito faça
5          Selecione um par de heurísticas de parada de remoção e de inserção
           (h_r, h_i) baseadas em seus pesos adaptativos;
6          Aplique (h_r, h_i) em s, produzindo s';
7          se s' é melhor que s_melhor então
8              s_melhor ← s';
9              s ← s';
10             senão
11                 se s' satisfaz o critério de aceitação então
12                     s ← s';
13         Atualize as pontuações e os pesos das heurísticas;
14     retorna s_melhor;
```

Com o intuito de detalhar os trabalhos que foram apresentados na revisão da literatura, foi criada a Tabela 3.1, para resumir as mais recentes e principais abordagens de meta-heurísticas encontradas na literatura, para resolver Problemas de Otimização Global Contínua. As meta-heurísticas apresentadas na tabela abaixo sofreram algum tipo de hibridização com outra meta-heurística.

Tabela 3.1: Novas abordagens referenciadas da literatura para POGC

Trabalho / Ano	**Abordagem**
(RESENDE; RIBEIRO, 2010)	C-GRASP
(BASHIR; NEVILLE, 2012)	EC + SQP
(CARRIZOSA et al., 2012)	VNS GAUSSIAN
(ANDRADE et al., 2014)	C-GRASP + GPU
(KOVAČEVIĆ et al., 2014)	DE + VNS
(ARASOMWAN; ADEWUMI, 2014)	PSO-CLUS
(ARAÚJO et al., 2015)	DC-GRASP
(MUTHIAH-NAKARAJAN; NOEL, 2016)	GSO
(ZENG et al., 2016)	ITS + VND
(NETO et al., 2017)	C-GRASP + DE
(MARTINS, 2017)	ALNS

Capítulo 4

Métodos Propostos

Nesta seção será apresentada a abordagem com adaptação automática baseada em DC-GRASP, proposta para otimização global contínua. De forma geral, será explicada a motivação para o desenvolvimento deste mecanismo que auxilia na escolha de parâmetros de utilização nas funções, a adaptação e a fase de calibração dos parâmetros via algoritmo PSO, o conceito da Busca Linear *Armijo Search*, e uma nova adaptação no método de descida APS, o qual foi proposto para auxiliar na busca local.

4.1 Abordagem com adaptação automática baseada em DC-GRASP

No C-GRASP(HIRSCH; PARDALOS; RESENDE, 2010) e DC-GRASP(ARAÚJO et al., 2015), o parâmetro h_s pode ser insuficiente para a convergência inicial adequada em cada iteração, podendo gerar um desperdício computacional em uma sub-região que caracteriza um ótimo local. Além disso, em funções com alto número de variáveis, encontrar os parâmetros h_s e h_e pode ser uma tarefa ainda mais complexa, sendo útil um mecanismo de definição automática desses parâmetros em tempo de execução.

Para tratar este problema, é proposta uma abordagem chamada *Self-Adaptive DC-GRASP* (DC-GRASP_SA), que incorpora no DC-GRASP mecanismos de adaptação automática para um melhor tratamento dos parâmetros h_s e h_e. Estes parâmetros na sua grande maioria são aplicados de forma manual, combinando vários valores para

poder utilizá-los nas entradas dos métodos. Sendo assim, houve a motivação para construir este mecanismo de adaptação automática.

No Algoritmo 18, é apresentado o pseudocódigo do DC-GRASP_SA. Para funções com variáveis acima de dez dimensões, uma etapa chamada **Adaptação** é realizada no início (veja o Algoritmo 19).

Algoritmo 18: Self-Adaptive DC-GRASP (DC-GRASP_SA)

```
1  procedimento DC-GRASP_SA(n, l, u, f(.), h_s, h_e, MaxLocalIters,
   MaxIterNoConv, MaxAvals)
2      se n > 10 então
3          paramList ← Adaptação(n, l, u, f(.), MaxLocalIters, MaxIterNoConv,
           MaxAvals);
4      f* ← ∞; NumIterNoConv = 0;
5      enquanto Critérios de parada não satisfeitos faça
6          x ← UnifRand(l, u);
7          se n > 10 então [h_s  h_e] ← random(paramList) ;
8          h ← h_s; f_ant ← ∞;
9          enquanto h > h_e faça
10             ImprC ← false;
11             ImprL ← false;
12             [x, ImprC] ← Construção(x, f(.), n, h, l, u, ImprC);
13             [x, ImprL] ← BuscaLocal_NAPS(x, f(.), n, h, h_e, l, u,
               MaxLocalIters, ImprL);
14             se f(x) < f* então x* ← x; f* ← f(x); ;
15             f_ant ← f(x);
16             se ImprC = false ∧ ImprL = false ∨ f(x) = f_ant então
17                 h ← h/2
18         se n ≤ 10 ∧ NumIterNoConv ≥ MaxIterNoConv então
19             h_s ← h_s + 1;
20             NumIterNoConv ← 0;
21         senão
22             NumIterNoConv ← NumIterNoConv + 1;
23     retorne x*;
```

Nessa etapa, o método é executado para todas as combinações de parâmetros entre $h_s = \{1, 2, ..., 10\}$ e $h_e = \{1, 0.1, 0.001\}$, permitindo-se apenas que $n \times MaxAvals$ avaliações da função sejam realizadas em cada possibilidade. A versão DC-GRASP_SA*, utilizada na adaptação, não possui as linhas 2, 3, 7, 18-22; além de retornar o *gap* para o valor mínimo conhecido da função. A função *atualizaLista* gerencia a lista de

parâmetros *paramList*, mantendo sempre os três melhores pares de h_s e h_e ordenados pelo menor *gap*.

Após essa etapa, em cada iteração do método principal (linha 5), um dos três pares de parâmetros é escolhido aleatoriamente. Para funções menores, a cada *MaxIterNoConv* iterações sem convergência para a solução ótima, o parâmetro h_s é incrementado. Outra alteração consiste na redução pela metade de h sempre que o valor da função objetivo corrente for o mesmo gerado na iteração anterior f_{ant} (veja as Linhas 15 e 16).

Algoritmo 19: Procedimento de adaptação de parâmetros

```
1  procedimento Adaptação(n, l, u, f(.), MaxLocalIters, MaxIterNoConv,
   MaxAvals)
2      h_s ← 1.0;
3      h_e ← {1, 0.1, 0.001};
4      paramList ← nil;
5      enquanto h_s ≤ 10 faça
6          para cada i = 1, ..., 3 fazer
7              minGap ← DC-GRASP_SA*(n, l, u, f(.), h_s, h_e[i], MaxLocalIters,
                   MaxIterNoConv, n × MaxAvals);
8              atualizaLista(paramList, h_s, h_e[i], minGap);
9          h_s ← h_s + 1;
10     retorne paramList;
```

4.1.1 Adaptação e calibração de parâmetros via PSO

Devido a alta variabilidade entre as regiões definidas pelas funções, obter resultados satisfatórios pode requerer informações prévias sobre essas regiões ou um tratamento individual. Os trabalhos de Hirsch (2010), Pardalos (2010) e Resende (2010) e Araújo et al. (2015) definiram os valores dos parâmetros h_s e h_e para cada função de teste, conseguindo resultados satisfatórios para todo o *benchmark* de funções. Seguindo esta prática, é proposto encontrar a parametrização adequada através da meta-heurística *Particle Swarm Optimization* (EBERHART; KENNEDY, 1995).

4.2 Busca Linear com *Armijo Search*

O *Armijo Search* é um tipo de Busca Linear que também pode ser classificada como um método de busca inexata. Apesar da grande necessidade de valores da função em diversos pontos como valores de suas respectivas derivadas, objetiva a determinação aproximada do valor do parâmetro α, situando-o em um intervalo aceitável de forma que este valor não seja considerado nem muito grande nem muito pequeno. Por não muito grande entende-se um α para o qual, definindo-se segundo a igualdade (LUENBERGER; YE, 2015):

$$\varphi(\alpha) = f(x_k + ad_k) \tag{4.1}$$

tem-se que:

$$\varphi(\eta\alpha) \leq \varphi(0) + \epsilon\varphi'(0)\alpha, \ para\ 0 < \epsilon < 1 \tag{4.2}$$

Por outro lado, α não é muito pequeno se:

$$\varphi(\eta\alpha) > \varphi(0) + \epsilon\varphi'(0)\eta\alpha, \ sendo\ \eta > 1 \tag{4.3}$$

A Regra de Armijo, ao sacrificar a precisão do cálculo, acarreta no aumento do número de iterações necessárias à convergência. Um fator benéfico, é oferecer uma redução bastante significativa no tempo gasto em cada iteração, relativamente a processos de busca exata.

4.3 Adaptação da busca na descida do APS

O Desenvolvimento de uma nova adaptação no método de descida APS originalmente proposto por Araújo et al. (2015), possibilita novas estratégias inteligentes de busca local para melhorar a convergência do método. Uma delas, é expandir a exploração de uma direção escolhida pelo APS, de forma que se possa achar um ponto melhor na direção explorada.

O que acontece é que o método de descida explora os pontos até o limite da direção, e se não achar um ponto melhor ele perturba e escolhe outro ponto.

O mecanismo tem contribuição por possibilitar que a busca local amplie a exploração e vá mais adiante na direção escolhida.

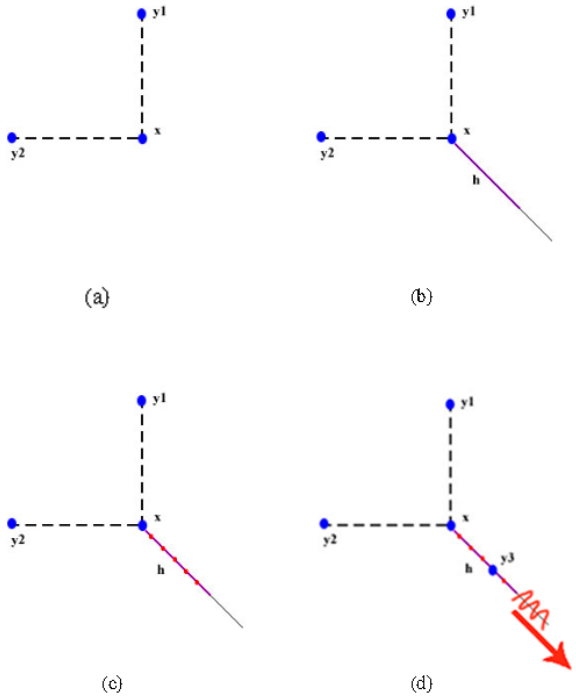


Figura 4.1: Representação da estratégia aplicada no APS em duas dimensões. Adaptado de (ANDRADE, 2013).

A Figura 4.1 mostra em d a estratégia aplicada no APS para exploração da direção promissora, visto que o ponto y^3 é considerado como o mínimo local da direção. O algoritmo faz uma validação para saber se existe algum valor de mínimo, para que este mecanismo seja acionado ou não.

Capítulo 5

Resultados

Nesta seção, as configurações do ambiente de teste e análise dos resultados obtidos nos experimentos realizados são apresentados e discutidos. O método proposto DC-GRASP_SA foi comparado com as versões anteriores C-GRASP e DC-GRASP para um conjunto de funções conhecidas.

5.1 Abordagem com adaptação automática baseada em DC-GRASP

5.1.1 Ferramentas e Tecnologia

Os algoritmos DC-GRASP e DC-GRASP_SA foram implementados em linguagem C++ e compilados através da ferramenta GNU g++ 4.8.2. O computador utilizado possui processador *Intel Core i5-3210M* (64-bits) dual-core com 2.50GHz e 8GB de memória RAM com o sistema operacional Linux Mint 17 (64 bits).

5.1.2 Execução do Experimento

A busca linear utilizada na BuscaLocal_NAPS foi a busca inexata de *Armijo* proposta por Luenberger (2015) e Ye (2015). As funções de testes utilizadas, definidas em Hirsch (2010), Pardalos (2010) e Resende (2010) e apresentadas na Tabela 5.1, possuem mínimo global conhecido e foram as mesmas utilizadas nos experimentos dos

trabalhos de Hirsch et al. (2007) e Hirsch (2010), Pardalos (2010) e Resende (2010).

A condição de otimalidade ou convergência é dada quando $f(x)$ for significantemente próximo do mínimo global, satisfazendo a inequação 5.1 com $\varepsilon_1 = 10^{-4}$ e $\varepsilon_2 = 10^{-6}$, considerando o ponto encontrado x e a solução ótima $f(x^*)$ conhecida.

$$|f(x^*) - f(x)| \leq \varepsilon_1 |f(x^*)| + \varepsilon_2 \tag{5.1}$$

De forma análoga ao trabalho de Hirsch (2010), Pardalos (2010) e Resende (2010) e Araújo et al. (2015), 100 execuções independentes do DC-GRASP e DC-GRASP_SA foram realizadas para cada função de teste. O critério de parada de ambos os métodos foi respeitar a condição de otimalidade ou realizar 20 iterações no procedimento multi-start. Para ambos os métodos, o parâmetro *MaxLocalIters* foi definido como $2n$. No DC-GRASP_SA foi utilizado $MaxIterNoConv = 3$ e $MaxAvals = 500$.

Tabela 5.1: Funções usadas na calibração por PSO.

Função	Dimensão	Função	Dimensão
Branin (BR)	2	Rosenbrock (R_5)	5
Goldstein and Price (GP)	2	Rosenbrock (R_{10})	10
Easom (EA)	2	Shekel ($S_{4,5}$)	4
Shubert (SH)	2	Shekel ($S_{4,7}$)	4
Hartmann ($H_{3,4}$)	3	Shekel ($S_{4,10}$)	4
Hartmann ($H_{6,4}$)	6	Zakharov (Z_5)	5
Rosenbrock (R_2)	2	Zakharov (Z_{10})	10

5.1.3 Análise dos Resultados

Devido a alta complexidade no tratamento de funções com naturezas distintas, obter resultados satisfatórios pode requerer informações prévias ou um tratamento individual. Os trabalhos de Hirsch (2010), Pardalos (2010) e Resende (2010) e Araújo et al. (2015) definiram os valores dos parâmetros h_s e h_e para cada função de teste, conseguindo resultados satisfatórios para todo o *benchmark* de funções. Dessa forma, além de utilizar os mesmos parâmetros propostos, para garantir uma comparação justa, utilizamos uma calibração via meta-heurística *Particle Swarm Optimization* (PSO) (EBERHART; KENNEDY, 1995) para encontrar novos parâmetros para as funções

BR, GP, EA, SH, $H_{3,4}$ e $H_{6,4}$. Para as demais funções os parâmetros foram definidos manualmente, uma vez que o PSO não conseguiu encontrar bons parâmetros em tempo aceitável. Os valores encontrados são apresentados na Tabela 5.2.

Na Tabela 5.3, são reportados o percentual de convergência, entre as 100 execuções, para uma solução significantemente próxima do ótimo global; e o número médio de avaliações da função, considerado apenas as execuções em que houve sucesso. Os resultados do C-GRASP, foram obtidos de Hirsch (2010), Pardalos (2010) e Resende (2010), para os parâmetros propostos no mesmo trabalho. O DC-GRASP e o DC-GRASP_SA[1] foram executados com os parâmetros propostos em Araújo et al. (2015). No DC-GRASP_SA[2], os parâmetros propostos na Tabela 5.2 foram utilizados.

Os resultados demonstram vantagens da abordagem proposta com relação aos outros algoritmos. Com a mesma parametrização do DC-GRASP, a versão DC-GRASP_SA[1], demonstrou que o mecanismo de incremento do passo h_s provocou melhorias no DC-GRASP em 8 funções, apesar de piorar os resultados para duas funções e empatar em 4 funções. A maior redução de avaliações média da função ocorreu nas funções EA e R_{10}, além da convergência de 100% obtida para a função $S_{4,10}$. Utilizando os novos parâmetros propostos, é notável a vantagem obtida em 12 das 14 funções, onde apenas para as funções GP e EA o algoritmo C-GRASP consegue melhores resultados.

Na Tabela 5.4, são reportados os resultados para as funções de Rosenbrock (R), Figura 5.1 e Zakharov (Z), Figura 5.2 com média e alta dimensionalidade. Os parâmetros do DC-GRASP foram os mesmos utilizados em Araújo et al. (2015). A função Z_{200}, considerada apenas neste trabalho, recebeu os mesmos valores utilizados em Z_{100}. Para a função de Rosenbrock, observa-se uma melhoria significativa, em todas as métricas, devido ao mecanismo de adaptação de parâmetros proposto. Para a função Zakharov, o método proposto conseguiu resultados superiores em Z_{20}. Porém, nas funções Z_{50} e Z_{100}, o DC-GRASP conseguiu convergir em 100% das iterações e o DC-GRASP_SA apenas em 92% para Z_{50}, além de não concluir a execução em tempo aceitável da função Z_{100}. Deve-se destacar que o DC-GRASP utiliza uma parametrização individual para cada função, diferentemente do DC-GRASP_SA, que encontra os valores adequados de h_s e h_e automaticamente. Se o parâmetro *MaxAvals*, definido como 500 para todo o conjunto, receber o valor de 15000 para Z_{50}, o DC-GRASP_SA consegue 100% de

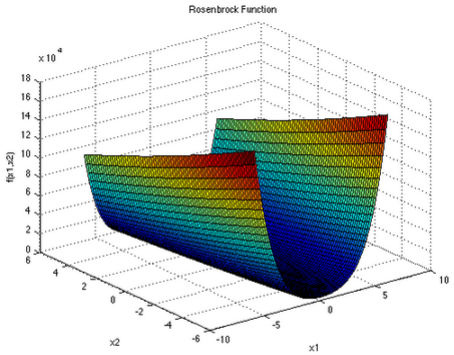
convergência com 10879802 avaliações médias e 6623 ms. Para $MaxAvals = 50000$, na função Z_{100}, o DC-GRASP_SA consegue 98% de convergência com 161391527 avaliações médias em tempo médio de 141811 ms. Esses resultados indicam que o número de avaliações necessárias na fase de adaptação para funções distintas possui grande variação. uma vez que a abordagem proposta possui melhores resultados em todos os aspectos, com exceção da função Z_{100}, onde o DC-GRASP_SA não conseguiu convergir em todas as iterações, apesar de obter menor tempo computacional e número médio de avaliações.

Tabela 5.2: Valores dos parâmetros utilizados nos experimentos

Função	h_s	h_e	Função	h_s	h_e
BR	0.5514	0.0002	R_5	3	1.2
GP	0.9999	0.9999	R_{10}	6	0.7
EA	0.6767	0.03	$S_{4,5}$	1	1
SH	0.4182	0.0001	$S_{4,7}$	1	0.9
$H_{3,4}$	0.804	0.0002	$S_{4,10}$	4	3
$H_{6,4}$	1.6235	0.0002	Z_5	2.5	0.5
R_2	2	1.5	Z_{10}	2.5	0.5

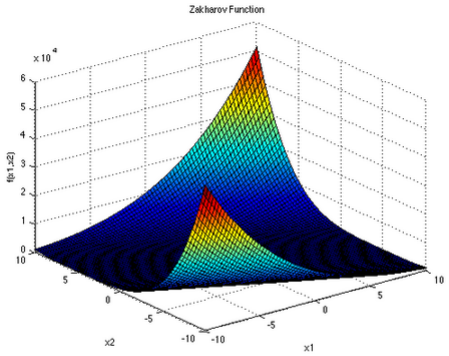
Tabela 5.3: Comparação do DC-GRASP_SA com versões da literatura para funções de baixa dimensionalidade

Função	C-GRASP		DC-GRASP		DC-GRASP_SA[1]		DC-GRASP_SA[2]	
	conv (%)	avals	conv (%)	avals	conv (%)	avals	conv (%)	avals
BR	100	10090	100	4797	100	4797	100	**3142**
GP	100	**53**	100	92	100	86	100	82
EA	100	**5093**	100	107824	100	77553	100	43238
SH	100	18608	100	6494	100	6494	100	**757**
$H_{3,4}$	100	1719	100	2021	100	2021	100	**1507**
$H_{6,4}$	100	29894	100	15659	100	16107	100	**8297**
R_2	100	23544	100	2538	100	1838	100	**107**
R_5	100	182520	100	25622	100	21173	100	**296**
R_{10}	100	725281	100	105222	100	58244	100	**579**
$S_{4,5}$	100	9274	100	1032	100	930	100	**852**
$S_{4,7}$	100	11766	100	1219	100	1074	100	**862**
$S_{4,10}$	100	17612	99	1638	100	1305	100	**215**
Z_5	100	12467	100	940	100	940	100	**588**
Z_{10}	100	2297937	100	43116	100	47186	100	**4230**



$$f(\mathbf{x}) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

Figura 5.1: Representação geométrica da função Rosenbrock utilizada nos experimentos de média e alta dimensionalidade



$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$$

Figura 5.2: Representação geométrica da função Zakharov utilizada nos experimentos de média e alta dimensionalidade

Tabela 5.4: Comparação do DC-GRASP_SA com versões da literatura para funções de média e alta dimensionalidade

Função	DC-GRASP			DC-GRASP_SA		
	conv (%)	avals	tempo (ms)	conv (%)	avals	tempo (ms)
R_{20}	100	150462	91	100	**24278**	16
R_{50}	100	741705	1266	100	**133834**	269
R_{100}	100	1998836	8015	100	**453073**	1807
R_{200}	100	8851145	92884	100	**2543639**	20484
Z_{20}	100	1594039	492	100	**138647**	81
Z_{50}	100	27741708	13102	92	43523008	26092

Capítulo 6

Considerações Finais

Esse trabalho tratou um problema bem conhecido na literatura de otimização global contínua através de uma abordagem baseada em C-GRASP. O método proposto insere mecanismos de adaptação automática de parâmetros, utilizando a meta-heurística PSO para funções de baixa e alta dimensionalidade, no método DC-GRASP.

O mecanismo de contribuição do APS fez com que fosse possível encontrar outros pontos na direção de descida, visto que o método atual é limitado por saber a distância inicial e final da reta de direção. O critério de parada na escolha de um ponto promissor está seguindo a mesma lógica do APS, apenas aumentando a área de direção a ser explorada. A busca linear original também foi modificada para usar a busca inexata baseada em *Armijo Search*.

Avalia-se que as contribuições foram de bastante importância para o método. A maior redução de avaliações média da função ocorreu nas funções EA e R_{10}, além de convergir 100% na função $S_{4,10}$. Utilizando os novos parâmetros propostos, é notável a vantagem obtoda em 12 das 14 funções, onde apenas para as funções GP e EA o algoritmo C-GRASP consegue melhores resultados.

Como trabalhos futuros, visando a ampliação nas estratégias de adaptações e diminuição no número de avaliações da função, têm-se como proposta de continuidade da pesquisa, a exploração de novas configurações da adaptação automática de parâmetros e a condução de novos experimentos envolvendo outras meta-heurísticas para o mesmo problema.

Bibliografia

ALBA, E. *Parallel metaheuristics: a new class of algorithms*. [S.l.]: John Wiley & Sons, 2005.

ANDRADE, L. M. M. d. S. Novas abordagens sequencial e paralela da meta-heurística c-grasp aplicadas à otimização global contínua. *Dissertação de Mestrado, Universidade Federal da Paraíba*, n. 101, 2013.

ANDRADE, L. M. M. d. S. et al. Parallel construction for continuous grasp optimization on gpus. *XLVI Simpósio Brasileiro de Pesquisa Operacional*, 2014.

ARASOMWAN, M. A.; ADEWUMI, A. O. Improved particle swarm optimization with a collective local unimodal search for continuous optimization problems. *The Scientific World Journal*, Hindawi Publishing Corporation, v. 2014, 2014.

ARAÚJO, T. M. U. de et al. Dc-grasp: directing the search on continuous-grasp. *Journal of Heuristics*, Springer, p. 1–18, 2015.

BASHIR, H. A.; NEVILLE, R. S. A hybrid evolutionary computation algorithm for global optimization. In: IEEE. *2012 IEEE Congress on Evolutionary Computation*. [S.l.], 2012. p. 1–8.

CARRIZOSA, E. et al. Gaussian variable neighborhood search for continuous optimization. *Computers & Operations Research*, Elsevier, v. 39, n. 9, p. 2206–2213, 2012.

DOWSLAND, K. A.; THOMPSON, J. M. Simulated annealing. In: *Handbook of Natural Computing*. [S.l.]: Springer, 2012. p. 1623–1655.

EBERHART, R. C.; KENNEDY, J. A new optimizer using particle swarm theory. In: NEW YORK, NY. *Proceedings of the sixth international symposium on micro machine and human science*. [S.l.], 1995. v. 1, p. 39–43.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995.

GLOVER, F.; LAGUNA, M. *Tabu Search∗*. [S.l.]: Springer, 2013.

GLOVER, F. W.; KOCHENBERGER, G. A. *Handbook of metaheuristics*. [S.l.]: Springer Science & Business Media, 2006.

GOLDBERG, D. E.; JOHN, H. Holland. genetic algorithms and machine learning. *Machine learning*, v. 3, n. 2-3, p. 95–99, 1988.

GRONER, R.; GRONER, M.; BISCHOF, W. F. *Methods of heuristics*. [S.l.]: Routledge, 2014.

HANSEN, P.; MLADENOVIĆ, N. *Variable neighborhood search*. [S.l.]: Springer, 2014.

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

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

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

KOVAČEVIĆ, D. et al. De-vns: Self-adaptive differential evolution with crossover neighborhood search for continuous global optimization. *Computers & Operations Research*, Elsevier, v. 52, p. 157–169, 2014.

KRAMER, O. [springerbriefs in applied sciences and technology] a brief introduction to continuous evolutionary optimization ||. In: _____. [s.n.], 2014. v. 10.1007/978-3-319-03422-5. ISBN 978-3-319-03421-8,978-3-319-03422-5. Disponível em: <http://gen.lib.rus.ec/scimag/index.php?s=10.1007/978-3-319-03422-5>.

LUENBERGER, D. G.; YE, Y. *Linear and nonlinear programming*. [S.l.]: Springer, 2015.

LUKE, S. *Essentials of Metaheuristics*. second. [S.l.]: Lulu, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

MARTINS, L. d. C. O uso da heurística adaptive large neighborhood search para resolver o problema de programação de tripulações do transporte público. 2017.

MELANIE, M. An introduction to genetic algorithms. 1996.

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & Operations Research*, Elsevier, v. 24, n. 11, p. 1097–1100, 1997.

MUTHIAH-NAKARAJAN, V.; NOEL, M. M. Galactic swarm optimization: A new global optimization metaheuristic inspired by galactic motion. *Applied Soft Computing*, Elsevier, v. 38, p. 771–787, 2016.

NETO, J. X. V. et al. Solving non-smooth economic dispatch by a new combination of continuous grasp algorithm and differential evolution. *International Journal of Electrical Power & Energy Systems*, Elsevier, v. 84, p. 13–24, 2017.

PATRIKSSON, M.; ANDREASSON, N.; EVGRAFOV, A. *An Introduction to Continuous Optimization*. [S.l.]: Studentlitteratur AB, 2007.

QIN, A. K.; SUGANTHAN, P. N. Self-adaptive differential evolution algorithm for numerical optimization. In: IEEE. *Evolutionary Computation, 2005. The 2005 IEEE Congress on*. [S.l.], 2005. v. 2, p. 1785–1791.

QUEIROGA, E. d. L. Meta-heurísticas de de otimização contínua para cluesterização de dados e segmentação de imagens. *Qualificação de Dissertação de Mestrado, Universidade Federal da Paraíba*, n. 63, 2016.

RESENDE, M. G.; RIBEIRO, C. C. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2010. p. 283–319.

RESENDE, T. A. F. M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, Springer US, v. 6, 03 1995. Disponível em: <http://gen.lib.rus.ec/scimag/index.php?s=10.1007/bf01096763>.

SALEH, A. H. *Constraint reasoning with local search for continuous optimization*. Tese (Doutorado) — Universidade Nova de Lisboa, 2014.

SOUZA, M. J. F. Inteligência computacional para otimização. *Notas de aula, Departamento de Computação, Universidade Federal de Ouro Preto, disponível em http://www. decom. ufop. br/prof/marcone/InteligenciaComputacional/InteligenciaComputacional. pdf*, 2008.

ZENG, Z. et al. Iterated tabu search and variable neighborhood descent for packing unequal circles into a circular container. *European Journal of Operational Research*, Elsevier, v. 250, n. 2, p. 615–627, 2016.

Apêndice A

Definições das Funções Teste

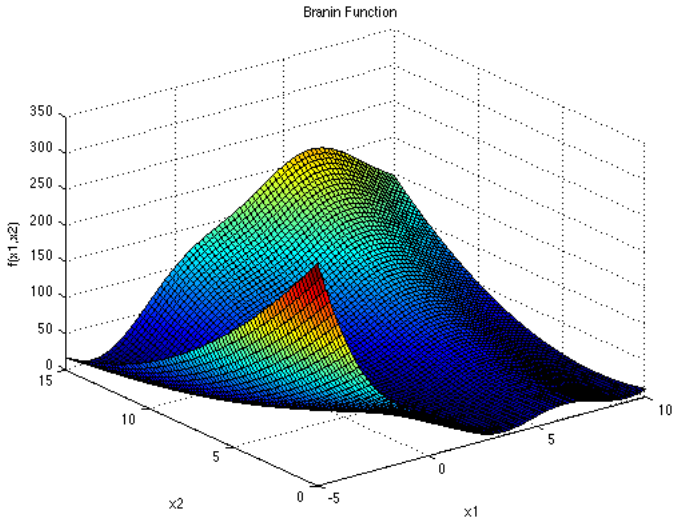


Figura A.1: Representação geométrica da função Branin

Definição:

$$f(x) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)cos(x_1) + s$$

Domínio:

$$x_1 \in [-5, 10], x_2 \in [0, 15]$$

Mínimo Global:

$$f(x^*) = 0.397887, \text{ em } x^* = (-\pi, 12.275), (\pi, 2.275) \text{ e } (9.42478, 2475)$$

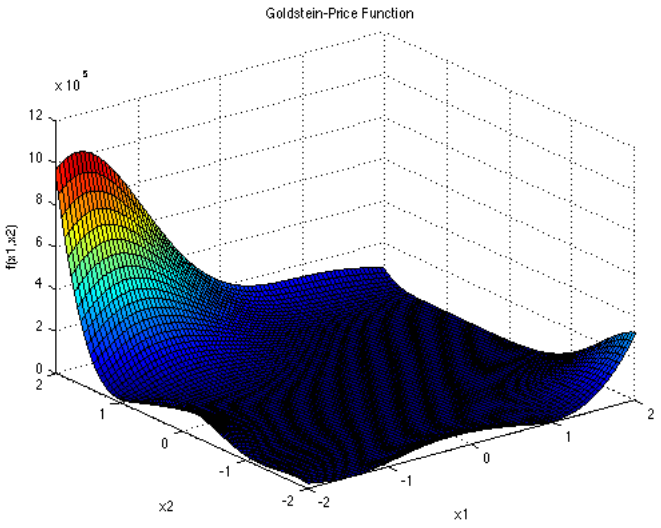


Figura A.2: Representação geométrica da função Goldstein and Price

Definição:

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

Domínio:

$$x_i \in [-2, 2], \text{ para todo } i = 1, 2$$

Mínimo Global:

$$f(x^*) = 3, \text{ em } x^* = (0, -1)$$

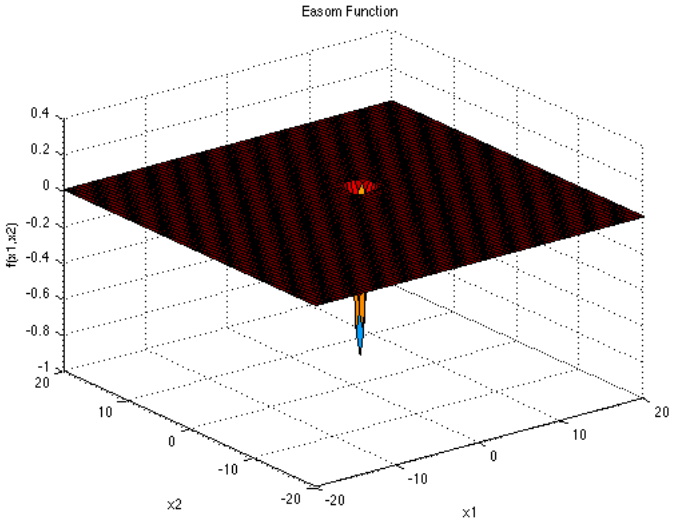


Figura A.3: Representação geométrica da função Easom

Definição:

$$f(x) = -cos(x_1)\ cos(x_2)\ exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$$

Domínio:

$$x_i \in [-100, 100], \text{ para todo } i = 1, 2$$

Mínimo Global:

$$f(x^*) = -1, \text{ em } x^* = (\pi, \pi)$$

$$f(\mathbf{x}) = -\sum_{i=1}^{m}\left(\sum_{j=1}^{4}(x_j - C_{ji})^2 + \beta_i\right)^{-1}, \text{ where}$$

$$m = 10$$

$$\beta = \frac{1}{10}(1, 2, 2, 4, 4, 6, 3, 7, 5, 5)^T$$

$$\mathbf{C} = \begin{pmatrix} 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \\ 4.0 & 1.0 & 8.0 & 6.0 & 3.0 & 2.0 & 5.0 & 8.0 & 6.0 & 7.0 \\ 4.0 & 1.0 & 8.0 & 6.0 & 7.0 & 9.0 & 3.0 & 1.0 & 2.0 & 3.6 \end{pmatrix}$$

Figura A.4: Representação da função Shekel

Domínio:

$x_i \in [0, 10]$, para todo $i = 1, 2, 3, 4$.

Mínimo Global:

Em m = 5: $f(x^*) = -10.1532$ em $x^* = (4, 4, 4, 4)$

Em m = 7: $f(x^*) = -10.4029$ em $x^* = (4, 4, 4, 4)$

Em m = 10: $f(x^*) = -10.5364$ em $x^* = (4, 4, 4, 4)$

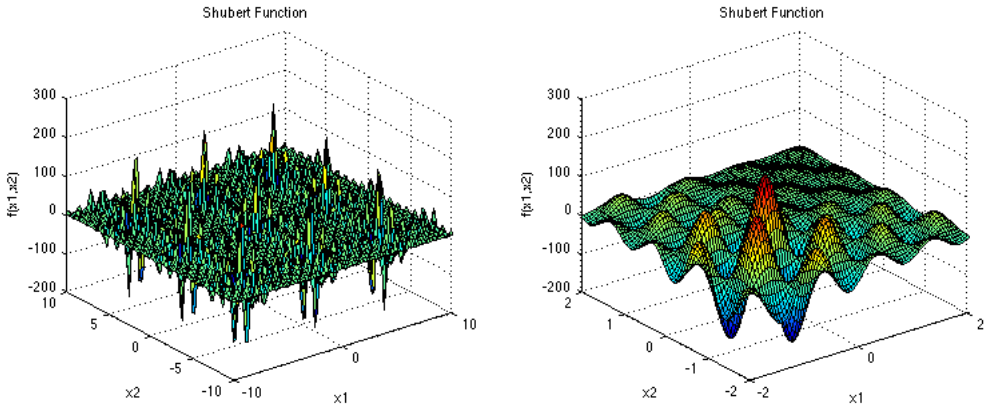


Figura A.5: Representação geométrica da função Shubert

Definição:

$$f(x) = \left(\sum_{i=1}^{5} i \; cos((i+1)x_1 + i) \right) \left(\sum_{i=1}^{5} i \; cos((i+1)x_2 + i) \right)$$

Domínio:

$x_i \in [-10, 10]$, para todo $i = 1, 2$

$x_i \in [-5.12, 5.12]$, para todo $i = 1, 2$

Mínimo Global:

$f(x^*) = -186.7309$

$$f(\mathbf{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right), \text{ where}$$

$$\alpha = (1.0, 1.2, 3.0, 3.2)^T$$

$$\mathbf{A} = \begin{pmatrix} 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3.0 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}$$

$$\mathbf{P} = 10^{-4} \begin{pmatrix} 3689 & 1170 & 2673 \\ 4699 & 4387 & 7470 \\ 1091 & 8732 & 5547 \\ 381 & 5743 & 8828 \end{pmatrix}$$

Figura A.6: Representação da função Hartmann

Domínio:

$x_i \in (0, 1)$, para todo $i = 1, 2, 3$

Mínimo Global:

$f(x^*) = -3.86278$, em $x(^*) = (0.114614, 0.555649, 0.852547)$

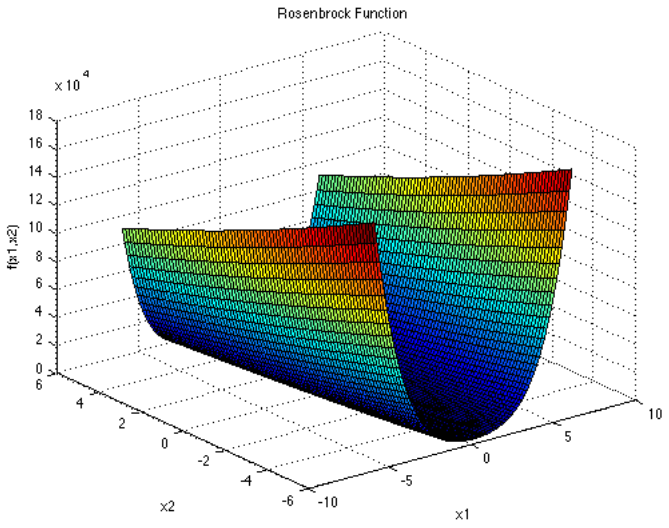


Figura A.7: Representação geométrica da função Rosenbrock

Definição:

$$f(x) = \sum_{i=1}^{d-1} [100(x_i + 1 - x_i^2)^2 + (x_i - 1)^2]$$

Domínio:

$x_i \in [-5, 10]$, para todo $i = 1, \ldots, d$

$x_i \in [-2.048, 2.048]$, para todo $i = 1, \ldots, d$

Mínimo Global:

$f(x^*) = 0$, em $x^* = (1, \ldots, d)$

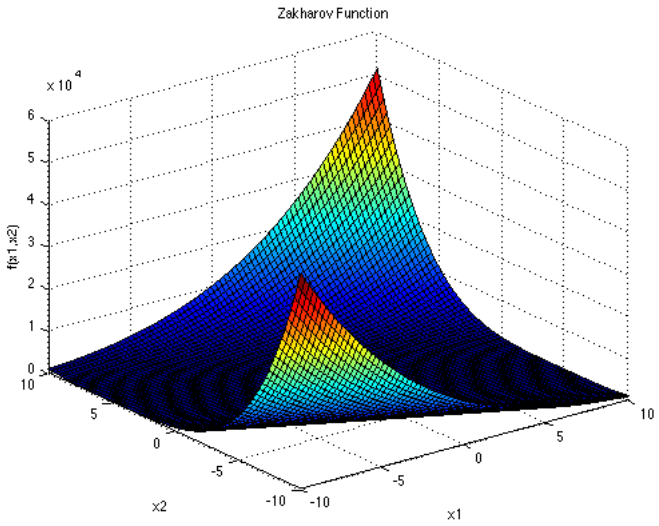


Figura A.8: Representação geométrica da função Zakharov

Definição:

$$f(x) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5 i x_i \right)^2 + \left(\sum_{i=1}^{d} 0.5 i x_i \right)^4$$

Domínio:

$x_i \in [-5, 10]$, para todo $i = 1, \ldots, d$

Mínimo Global:

$f(x^*) = 0$, em $x^* = (0 \ldots, 0)$