UNIVERSIDADE FEDERAL DE PERNAMBUCO – UFPE
CENTRO DE INFORMÁTICA

Graduação em Ciência da Computação

GRASP + C-GRASP para problemas de otimização em problemas de programação não linear inteiro misto

Raul Maia Falcão

Trabalho de Graduação

Recife,
Julho, 2017

UNIVERSIDADE FEDERAL DE PERNAMBUCO – UFPE
CENTRO DE INFORMÁTICA

Raul Maia Falcão

GRASP + C-GRASP para problemas de otimização em problemas de programação não linear inteiro misto

Trabalho de Conclusão de Curso apresentado à Coordenação do Curso de Ciência da Computação da Universidade Federal de Pernambuco como requisito parcial para a conclusão da graduação desse curso.

Orientador: Ricardo Martins de Abreu Silva

Agradecimentos

Gostaria de começar a agradecer aos meus pais que foram importantes para a minha educação e formação como ser humano. Minha mãe Bernadete que sempre me ensinou a importância de se praticar o bem e como agir perante os altos e baixos da vida. Meu pai Falcão que apesar da distância é um espelho que retrata os princípios básicos que um homem deve ter. Meu irmão Victor, tão diferente de mim nas decisões mas tão igual nos pensamentos, com quem as conversas vão de muitas brincadeiras e risadas.

Ao longo dessa jornada acadêmica pude fazer grandes amigos que encontrei dentro desta universidade. Amigos que, assim como eu, amam a ciência e debatem perguntas e respostas, sem nenhum julgamento, que surgem no dia a dia para agregar conhecimento. André, Wilson e Lucas agradeço imensamente a força que me deram e o cohecimento compartilhado. A minha namorada Lyana que me fez encontrar respostas das dúvidas sobre a vida e acima de tudo trabalhar meu conhecimento com a sabedoria: "somos todos um".

Agradeço a todos que fizeram parte deste projeto. Acreditando no meu potencial meu professor e orientador, Ricardo Martins, com muitos puxões de orelha e prazos de entrega, contribuiu para minha formação acadêmica. Pedro, a primeira pessoa a fazer parte do projeto, obrigado no progresso inicial do projeto. Mariana, que chegou depois, obrigado pelas conversas e idéias trocadas nas nossas reuniões.

Obrigado a todos que, de alguma forma, contribuíram para minha formação acadêmica e pessoal .

Resumo

Este trabalho apresenta as meta-heurísticas Greedy Randomized Adaptive Search Procedure (GRASP) e Continuous GRASP (C-GRASP) com o objetivo de minimizar o erro encontrado em relação ao melhor valor conhecido na literatura da função objetiva em problemas Mixed Integer Nonlinear Programming (MINLP). O maior problema nos algoritmos estocásticos de busca local é mover uma solução dentre o conjunto de soluções candidatas para garantir a melhor solução. Funções multimodais para MINLP possuem variáveis continuas e discretas. Nesse estudo, uma implementação foi desenvolvida para integrar eventos intercalados como método eficiente de encontrar boas soluções em uma função multimodal. Eventos do C-GRASP são impregados quando as variáveis continuas permanecem livres e as variáveis discretas fixas. Ao contrátio, os eventos GRASP são empregados quando as variáveis discretas permanecem livres e as variáveis contínuas fixas. A qualidade das soluções encontradas após a análise experimental de quatro problemas selecionados, é apresentada em forma de TTT plot.

Palavras chaves: MINLP, GRASP, C-GRASP, busca local estocástica, função multimodal

Abstract

This work presents the metaheuristics such as Greedy Randomized Adaptive Search Procedure (GRASP) and Continuous GRASP (C-GRASP) aiming to minimize the error found against the best known value at literature of the objective function in Mixed Integer Nonlinear Programming (MINLP). A major problem in stochastic local search algorithms is moving a solution in a set of candidate solutions to guarantees the best solution. A implementation was developed to integrate interleaved events as a efficiently method to find good solutions of a multimodal function. A multimodal function for MINLP have continuous and discrete variables. C-GRASP events was employed when continuous variables were unfixed and discrete variable were fixed. Contrary, GRASP events was employed when discrete variables were unfixed and continuos variables were fixed. The quality of solutions found after the experimental analysis among four problems from GAMS World is shown in the form of TTT plot.

Keywords: MINLP, GRASP, C-GRASP, stochastic local search, multimodal function

Lista de Figuras

1 Arquitetura e fluxo de execução do pacote MINLP 14
2 Distribuição de probabilidade acumulada dos tempos de execução para as instâncias ex1223,gear3,gear2 e alan 22

Lista de Tabelas

1 Variáveis de decisão, número de restrições e valor da melhor solução conhecida dos problemas 9
2 Arquivos da biblioteca $libcgrasp$ e $libgrasp$ 10
3 Quantidade de chamadas a funçao objetiva usando o $C\text{-}GRASP$ no problema de otimização global $ackley$ 11
4 Configurações dos problemas: níveis de valores para os parâmetros h_s, h_e e ϵ 20
5 Resumo estatístico das simulações 21

Sumário

1 Introdução **1**

2 Modelos MINLP **2**

3 História da arte **3**

3.1 *Branch-and-Bound* 3

3.2 *Generalized Bender's Decomposition/OuterApproximation* . 3

4 Fundamentação Teórica **4**

5 Modelos **8**

5.1 Alan 9

5.2 Gear2 e Gear3 9

5.3 Ex1223 9

6 Biblioteca Python/C **9**

6.1 Instalação e uso da biblioteca 11

6.2 Formato de entrada 12

7 Arquitetura **14**

7.1 C-GRASP 15

7.1.1 Busca Local 15

7.2 GRASP 16

7.2.1 Busca Local 18

7.3 C-GRASP + GRASP 18

8 Análise experimental **19**

9 Resultados **20**

10 Conclusão **21**

11 Referências **24**

1 Introdução

Os problemas de Programação Não Linear Inteiro Misto (PNLIM) são aplicados comumente na área científica, na engenharia e no setor público e privado da indústria (BELOTTI et al., 2013; RESENDE; RIBEIRO, 2014). Atualmente a maioria dos métodos utilizados são baseados em $cutting\text{-}plane$ como $Outer\ Approximation$ /$Generalized\ Bender's\ Decomposition$ (BELOTTI et al., 2013) ou na busca em árvores como $Branch\text{-}and\text{-}Bound$ ($B\&B$) . Nesses problemas de otimização global, muitos desafios são encontrados mesmo quando variáveis inteiras são relaxadas resultando em muitos locais mínimos.

Um problema de otimização global é definido matematicamente em encontrar uma solução $x^* \in S \subseteq \mathbb{R}^n$ tal que $f(x^*) \leq f(x), \forall x \in S$, onde S é uma região de $\mathbb{R}^n$ e a função objetiva f é definida por $f : S \to \mathbb{R}$ sendo x^* a solução chamada de mínimo global. A região considerada neste trabalho é, sem perda de generalidade, um hiper-retângulo definido por $S = \{x = (x_1, ..., x_n) \in \mathbb{R}^n : l \leq x \leq u\}$, sendo $l, u \in \mathbb{R}$ e $l \leq u$. O problema de minimização com restrições de caixa consiste em encontrar $x^* = argmin\{f(x)|l \leq x \leq u\}$, onde $f : \mathbb{R}^n \to \mathbb{R}$, onde $l, x, u \in \mathbb{R}$.

As meta-heurísticas já foram mostradas em trabalhos anteriores para resolver problemas de otimização global contínuo: simulated annealing (LAARHOVEN; AARTS, 1987), busca tabu (FEO; RESENDE, 1995), scatter search (LAGUNA; MARTÍ, 2005), algoritmos genéticos (MICHALEWICZ; FOGEL, 2000; MICHALEWICZ; NAZHIYATH, 1995). Dentre as meta-heurísticas também é incluido o $GRASP$ e o $C\text{-}GRASP$. Essas últimas, se baseam na estratégia de construir uma boa solução inicial para posteriormente buscar em torno da vizinhança soluções melhores. As estratégias originais, utilizadas no $GRASP$ e o $C\text{-}GRASP$, sofreram alterações de forma a evitar avaliações desnecessárias da função objetiva, com o propósito de diminuir o tempo de execução para atingir o valor alvo conhecido (HIRSCH; PARDALOS; RESENDE, 2010). As funções de teste utilizadas foram retiradas da coleção de modelos MINLP (BUSSIECK; DRUD; MEERAUS, 2003).

As bibliotecas $libgrasp$ e $libcgrasp$ foram implementadas no estilo GNU (CALCOTE, 2010) de forma similar àquela proposta por (AIEX; RESENDE; RIBEIRO, 2007). Para legibilidade e simplicidade na representação de funções multimodais complexas foi acoplado código Python/C . Para maior precisão aritmética de pontos flutuantes a biblioteca MPFR (FOUSSE et al., 2007) foi utilizada. Um pacote (C-$GRASP$+$GRASP$) como método de solução para problemas MINLP foi implementado para a integração das meta-heurísticas.

O estudo presente será apresentado da seguinte forma: definição de modelos MINLP (seção 2), história da arte com métodos de soluções para MINLP (seção 3), fundamentação teórica das meta-heurísticas descritas (seção 4), modelos usados para a realização dos experimentos (seção 5), detalhes da construção da biblioteca GRASP + C-GRASP (seção 6), arquitetura e fluxo geral do pacote MINLP (seção 7), experimento piloto com quatro problemas modelos selecionados, discussão de parâmetros necessários para realizar uma simulação (seção 8), interpretação dos resultados obtidos com o auxílio de TTT plots (seção 9) e conclusão (seção 10).

2 Modelos MINLP

Os problemas de otimização MINLP possuem variáveis discretas e contínuas modeladas com funções não lineares. Um problema MINLP pode ser formulado algebricamente como:

(1) $min\ f(x, y)$, sujeito a

(2) $g_i(x, y) \leq 0$, i = 1,...,q

(3) $h_j(x, y) = 0$, j = q+1,...,m

(4) $l_i \leq x_i \leq u_i$, $i = 1,...,n$

(5) $y_j \in \mathbb{Z}$, $j = 1,...,s$.

Onde x representa o vetor de n variáveis contínuas em um hiper-retângulo $S = \{x = (x_1, ..., x_n) \in \mathbb{R}^n \mid l \leq x \leq u\}$, $l, u \in \mathbb{R}^n$ tal que $l \leq u$, $\forall$ $i = 1, .., n$, y é o vetor de variáveis inteiras em S, $g(x, y)$ são as restrições de

inequação, $h(x, y)$ são as restrições de igualdade, e $f(x)$ é a função objetiva. Na seção seguinte, discutiremos métodos de solução para os problemas MINLP.

3 História da arte

3.1 $Branch$-and-$Bound$

O algoritmo $Branch$-and-$Bound$ (B&B) mantêm uma lista de problemas não resolvidos além de guardar o registro da melhor solução inteira encontrada. O primeiro passo do algoritmo é resolver subproblemas que foram resultados de um relaxamento de condições discretas nas variáveis binárias. Quando uma solução inteira é encontrada, defini-se um limite superior a solução. Todos os nós da árvore que excedem esse limite, são podados e a busca continua até que todos os nós restante da árvore sejam explorados. A principal desvantagem em usar essa abordagem para problemas MINLP, é que o método requer uma quantidade de grande problemas não lineares tornando-o computacionalmente custoso.

3.2 $Generalized\ Bender's\ Decomposition/Outer Approximation$

$Generalized\ Bender's\ Decomposition$ (GDB) é um algoritmo que depende do relaxamento do problema original de forma a solucionar um problema alternativo de $Mixed\ Integer\ Linear Programming$ (MILP) e dois subproblemas $Non\ Linear\ Programming$ (NLP). A deficiência dessa estratégia é que muitas vezes pode resultar na enumeração total do espaço binário devido a relaxamentos mal feitos.

$Outer\ Approximation$ (OA) é um algoritmo que também consiste em sequências de subproblemas NLP e um problema chave MILP. A vantagem do OA sobre GDB é que o o problema chave do OA já é bem definido. No entanto, tanto o GBD quanto o OA tem suas limitações, o tamanho do problema aumenta a cada iteração tornando a computação inviável.

4 Fundamentação Teórica

São exemplos clássicos problemas de otimização vistos nos setores públicos e privados da economia, como: corte e empacotamento, roteamento de veículos e atribuições de frequências em telefonia de celular. Buscar e encontrar soluções que maximizem ou minimizem as funções objetivas contribui para solucionar esses problemas. A natureza desses problemas são intratáveis do ponto de vista computacional. Apesar das combinações serem finitas, muitos problemas possuem um domínio extremamente grande o que torna custoso explorar e avaliar todas os possíveis casos através de uma função objetiva $f : X \leftarrow \mathbb{R}$.

A medida em que o domímio cresce o espaço de soluções viáveis cresce de forma exponencial. Isso motivou o desenvolvimento de meta-heurísticas, estratégias que envolvem programação inteira, algoritmos probabilísticos e algoritmos de aproximação. Há mais de 60 anos têm se desenvolvido técnicas que evitam explorar todo o domínio para encontrar uma solução de boa qualidade em um tempo de execução menor. Essa área de pesquisa ficou conhecida como otimização combinatorial, hoje amplamente aplicado na área acadêmica e em problemas do mundo real de grande escala.

(FEO; RESENDE, 1995) foram os primeiros a introduzir o uso do $GRASP$ como método de solução para os problemas de conjunto de cobertura, conjunto independente máximo, e do problema da satisfatibilidade máxima, todos problemas NP-completo. A extensão do $GRASP$ para o domínio contínuo, o C-$GRASP$, surgiu como método de otimização global sujeito a restrições de caixa. Tanto $GRASP$ quanto C-$GRASP$ são meta-heurísticas multi-inicio ou iterativa, que tem como objetivo encontrar soluções aproximadas. Cada iteração da meta-heurística possui duas fases: a de construção gulosa e a de busca local.

Na fase de construção gulosa (algoritmo 1), componente comum no $GRASP$ e no C-$GRASP$, uma solução factível é criada. A criação da solução começa permitindo que todas as coordenadas da solução inicial sejam livres. Na linha 10 da busca gulosa o $line\ search$ (algoritmo 2) é executado. Esse procedimento avalia

a função objetiva usando a k-coordenada de x nos valores $\{x_k, l_k, l_k + h, ..., l_k + v * h, u_k\}$, onde $v = max[\mu \mid \mu \in \mathbb{Z}, l_k + \mu * h \leq u_k]$. O valor retornado é a coordenada que produziu o menor valor da função objetiva. O $line\ search$ atua em cada direção de coordenada livre i da solução x com o restante das n - 1 coordenadas mantidas fixas. O valor de z_i para a i-ésima coordenada que minimiza a função objetiva é armazenada bem como o valor g_i da função objetiva. Depois da chamada do procedimento $line\ search$ uma Lista de Candidatos Restritos (LCR), linha 21, é formada contendo os índices das coordenadas livres cujo os elementos incorporados na solução parcial resulta em um menor incremento de custo. O tamanho da lista é controlado pelo parâmetro $\alpha \in \{0, 1\}$ que controla se o grau da construção é puramente guloso ou é uma construção puramente aleatória. (HIRSCH; PARDALOS; RESENDE, 2010) avaliaram o comportamento do parâmetro α, sendo ele mais eficaz quando seu valor varia ao invés de um valor fixo. O componente probabilístico da construção está na escolha aleatória de uma solução candidata, mas não necessariamente a melhor candidata. O procedimento continua até que todas as n coordenadas de x permaneçam fixas. Na linha 8 determina que para cada $i \in UnFixed \setminus \{j\}$, o valor de x é alterado durante o procedimento do $line\ search$, mas o valor de x_j permanece inalterado. Na linha 30 mostra que caso o valor $Reuse$ seja verdade isso implica que na próxima vez as linhas 7 a 17, com $j \notin UnFixed$, a j-ésima coordenada de x irá produzir os mesmos resultados achados na etapa anterior. Assim, se o valor de $Reuse$ é verdadeiro, permite-se que a etapa de construção reuse os valores de z_i e g_i previamente computados.

Na fase de busca local uma vizinhança é gerada em torno da solução criada pela fase de construção com o propósito de melhorar o resultado da função objetiva. Um solução é dita h-mínimo local, se todos os pontos da vizinhança foram examinados e não houver melhora do resultado da função objetiva. Dois componentes são importantes na melhora do mecanismo na busca de boas soluções: a intensificação e a diversificação (FEO; RESENDE, 1995). O primeiro faz uso de uma memória explícita para buscar de vizinhos em um conjunto elite de soluções. Sua implementação foi proposta por (GLOVER, 1996). O uso do $Path$-

Algorithm 1 ConstructGreedyRandomized

$S \leftarrow (1, 2, ..., n)$
$\alpha \leftarrow UnifRand(0, 1)$
$Reuse \leftarrow False$
do:
    $min \leftarrow +\infty$
    $max \leftarrow -\infty$
    for $i = 1 \rightarrow n$ **do:**
        if $i \in S$ **then**
            if $Reuse = False$ **then**
                $z_i \leftarrow LineSearch(x, h, i, n, f(.), l, u)$
                $g_i \leftarrow f(z_i)$
            endif
            if $min > g_i$ **then**
                $min \leftarrow g_i$
            endif
            if $max < g_i$ **then**
                $max \leftarrow g_i$
            endif
        endif
    endfor
    $RCL \leftarrow \emptyset$
    $threshold \leftarrow min + \alpha * (max - min)$
    for $i = 1 \rightarrow n$ **do:**
        if $i \in S$ **and** $g_i \leq threshold$ **then**
            $RCL \leftarrow RCL \cup \{i\}$
        endif
    endfor
    $j \leftarrow RandomlySelectElement(RCL)$
    if $x_j = z_j$ then
        $Reuse \leftarrow True$
    else
        $x_j \leftarrow z_j$
        $Reuse \leftarrow False$
        $Improv_c \leftarrow True$
    endif
    $S \leftarrow S \setminus \{j\}$
while $S \neq \emptyset$**:**
return $(x, Improv_c)$

Algorithm 2 Line Search

```
1: t ← x
2: z_k ← x_k
3: minF ← f(x)
4: t_k ← l_k
5: while t_k ≤ u_k then
6:     if f(t) < minF then
7:         minF ← f(t)
8:         z_k ← t_k
9:     endif
10:    t_k ← t_k + h
11: endwhile
12: t_k ← u_k
13: if f(t) < minF then
14:     minF ← f(t)
15:     z_k ← t_k
16: endif
17: return z_k
```

$Relinking$ mostrou robustez e eficiência no problema de atribuição quadrática generalizada (MATEUS; RESENDE; SILVA, 2011). O segundo componente, por sua vez, força a busca por regiões do espaço de soluções ainda não visitadas, ou gera soluções que são significativamente diferentes de regiões previamente visitadas. A diversificação geralmente é implementada por pertubações que destroem a estrutura da solução atual.

Um outro componente em comum é o critério de parada, para finalizar as iterações do loop mais externos do $GRASP$ e C-$GRASP$, duas condições foram estabelecidas: i) número de máximo de iterações, ou ii) se a solução encontrada convergia a um ponto que:

$$GAP \leq \begin{cases} \mid f(x) \mid \leq \epsilon & \text{se } f(x^*) = 0 \\ \mid f(x) - f(x^*) \mid \leq \epsilon * \mid f(x) \mid & \text{se } f(x^*) \neq 0, \end{cases}$$

onde GAP é definido como tolerância ótima.

As duas estratégias não fazem uso de informação a $priori$ e possuem uma implementação semelhante, diferindo apenas no domínio em que atuam e na estrutura da vizinhança na etapa da busca local. Duas características especiais fazem parte do $GRASP$ e C-$GRASP$: ambos podem ser facilmente implementados e

paralelizados. A implementação possui poucos parâmetros que precisam ser testados e melhorados, enquanto que a paralização é possível devido a independência das iterações.

5 Modelos

Para validação de algoritmos de otimização, um teste de confiabilidade é frequentemente realizado usando um conjunto escolhido de $benchmarks$ padrão ou funções de teste da literatura. Idealmente, as funções de teste utilizadas devem ser diversas e imparciais, no entanto, não existe um conjunto acordado de funções de teste na literatura. Em geral, os problemas são classificados em duas categorias: funções teste e problemas do mundo real. Nesse trabalho foram selecionados quatro modelos de funções teste de pequena e média complexidade.

$MINLPLib$ é uma coleção de modelos MINLP de testes teóricos e práticos. Inicialmente foi criada para combinar problemas existentes na literatura com modelos industriais de grande porte. As áreas de aplicação dos testes selecionados, para esse trabalho, incluem: a análise e escolha de um portfólio, obtenção da relação de transmissão entre um condutor e os eixos acionados de um trem projetado, e design de engenharia. Os problemas de teste algébrico estão disponíveis na linguagem de modelagem GAMS, sendo todos esses disponíveis para serem baixados no site $MINLPLib\ Model$.

Nem todos os modelos disponíveis são bem formulados, e alguns podem ser extremamente difíceis de se resolver ou até mesmo serem inviáveis. Esta variedade de problemas podem ajudar a testar a confiabilidade de um método de solução para um dado modelo.

A tabela abaixo mostra a dimensão de cada problema e a quantidade dos tipos de variáveis, assim como a quantidade de restrições presentes e o valor da melhor solução encontrada na literatura.

Problema	Contínuas	Discretas	Restrições	Melhor solução
alan	4	4	7	2.925
gear2	4	24	4	0.0
gear3	4	4	4	0.0
ex1223	7	4	13	4.5796

Tabela 1: Variáveis de decisão, número de restrições e valor da melhor solução conhecida dos problemas

5.1 Alan

O primeiro modelo de MINLP é um modelo de programação quadrática para análise de portfólio. Esse modelo permite escolher uma carteira de investimentos cujo retorno esperado atende um alvo e minimiza a variância..

5.2 Gear2 e Gear3

Esses dois modelos consiste em projetar um trem de engrenagens para alcançar uma relação de transmissão entre o condutor e os eixos acionados. O objetivo é encontrar o número de dentes das quatro engrenagens para obter uma relação de transmissão necessária de $1/6.931$.

5.3 Ex1223

Modelo proposto por Yuan-et.al (1988) com aplicações na área de design de engenharia, como a síntese de redes de troca de calor, síntese de redes de reciclagem de reatores ou síntese de redes de bombas hidráulica.

6 Biblioteca Python/C

Nessao seção descrevemos a replicação das bibliotecas $libgrasp$ e $libcgrasp$ no estilo GNU dinâmico e partilhada com a biblioteca Python/C das meta-heurísticas

$GRASP$ e C-$GRASP$ respectivamente (AIEX; RESENDE; RIBEIRO, 2007). Como contrubuição, a biblioteca $MPFR$ foi utilizada para garantir múltipla-precisão aritmética de pontos flutuantes; sendo a precisão setada para 12 casas decimais (FOUSSE et al., 2007). Após a compilação das bibliotecas, os programas $cgrasp$ e $grasp$ ficam disponíveis para execução. Os parâmetros das meta-heurísticas são passadas por linha de comando que são analisadas pela biblioteca $boost$. Como alternativa, esses parâmetros também podem ser especificados em um arquivo de configuração passado aos programas. Uma característica importante da biblioteca é que o as funções implementadas em Python são carregadas sem a necessidade de uma nova compilação das bibliotecas. As bibliotecas foram implementadas em C++ e desenvolvidas para rodar em plataformas Linux. Os arquivos e pastas do diretório estão listados na tabela 2.

Arquivo	Descrição
cgrasp.cpp	Código fonte c-grasp
cgrasp.h	Cabeçalho do cgrasp.cpp
grasp.cpp	Código fonte grasp
grasp.h	Cabeçalho do grasp.cpp
src/	Códigos de procedimentos utilizados no cgrasp.cpp e grasp.cpp
libs/	Pasta com funções multi-modais escritas em Python
configure	Script que configura os pacotes e verifica dependências automaticamente
AUTHORS	Nome e endereço eletrônico dos autores
COPYING	GNU General Public License
INSTALL	Instruções para instalação
NEWS	7
README	Arquivo explicando os parâmetros dos programas e um exemplo de uso
THANKS	Agradecimentos aos contribuidores

Tabela 2: Arquivos da biblioteca $libcgrasp$ e $libgrasp$

O algoritmo Mersenne Twister foi incorporado na biblioteca para geração de números pseudo-aleatórios (ANNALS et al., 2013). A aleatoriedade dos números tem importância direta na criação da solução inicial, na escolha da coordenada $j \in RCL$ na construção gulosa, e na escolha de pontos no espaço na $grid$ da

busca local.

Para analisar a perfomance de execução dos programas, $gprof$ foi usado para coletar a quantidade de chamadas a um método ou procedimento (GRAHAM; KESSLER; MCKUSICK, 1982). A tabela 3 ilustra a quantidade de chamadas da função objetiva. Podemos observar que em todas as etapas do C-$GRASP$ a avaliação da função objetiva é realizada. Os vazamentos de memória foram analisados através da ferramenta $Valgrind$ e possibilitaram a correção de $bugs$ no código-fonte (NETHERCOTE; SEWARD, 2007).

Método	Qte.Chamadas
cgrasp()	30/58884
constructGreedyRandomized()	1506/58884
localImprovement()	3030/58884
lineSearch()	54318/58884
PythonInterface::objectiveFunction	58884

Tabela 3: Quantidade de chamadas a funçao objetiva usando o C-$GRASP$ no problema de otimização global $ackley$

6.1 Instalação e uso da biblioteca

(1) descompacte o pacote:

tar -xvf $cgrasp$+$grasp.tar.gz$

(2) execute o arquivo de configuração para verificação de dependências de pacotes:

$./configure$

(3) digite $make$ para compilar os programas e $make\ install$ para instalar o pacote:

$make$

$sudo\ make\ install$

Os binários $cgrasp$ e $grasp$ serão criados na pasta $/usr/bin$ prontos para a execução. Ao serem executados, um arquivo de $trace$ é gerado informando se nas fases de construção gulosa e busca local houve ou não melhora da solução atual.

Se o usuário tiver interesse em escrever seu próprio código e quiser fazer o uso do C-$GRASP$ ou do $GRASP$:

(1) basta fazer o $include$ da biblioteca em seu arquivo $meuprograma.cpp$:

```
# include <cgrasp.h>
# include <grasp.h>
...
double x;
double y; ...
void main(int argc, char **argv){
...
x = cgrasp(argc,argv);
y = cgrasp(argc,argv);
...
}
.
```

(2) adicionar a $flag$ -$lcgrasp$ ou -$lgrasp$ respectivamente no momento da compilação. Exemplo da compilação do $meuprograma.cpp$ usando as bibliotecas C-$GRASP$ e $GRASP$, especificando $pathname$ do cabeçalho Python:

```
gcc -I<pathname> meuprograma.cpp -o meuprograma -lcgrasp -lgrasp
```

6.2 Formato de entrada

A seguir, mostramos os argumentos permitidos aos programas:

- –function <arg>: define o nome da função.
- –dimension $< arg >$: define a dimensão do problema.
- –lowerbound $< arg >$: define os limites inferiores.
- –upperbound $< arg >$: define os limites superiores.

- –startgrid $< arg >$: define o começo da densidade da $grid$ (C-$GRASP$ apenas).
- –endgrid $< arg >$: define o final da densidade da $grid$ (C-$GRASP$ apenas).
- –ro $< arg >$: define a porção da vizinhança a ser analisada (C-$GRASP$ apenas).
- –k $< arg >$: define o número de pontos em torno da vizinhança a serem analisados.
- –it $< arg >$: define o número de iterações.
- –ep $< arg >$: define a tolerância ótima.
- –excpt $< arg\ list >$: define o conjunto de restrição. Exemplo a exceção –excpt 2 2 4 3 -3 -1 seta a segunda dimensão com limite inferior 2 e limite superior 4, e a terceira dimensão com limite inferior -3 e limite superior -1.
- –setexpt $< arg\ list >$: outra forma de definir o conjunto de exceção de restrições. Exemplo a exceção –setexpt 1 3 -1 10 seta a primeira até a terceira dimensão com limite inferior -1 e limite superior 10.
- –seed $< arg >$: define a semente a ser usada.
- –config-file $< file\ name >$: define o arquivo de configuração.

Todas os parâmetros são obrigatórios, a menos que um arquivo de configuração (–config-file) seja passado como parâmetro. Um exemplo do conteúdo do arquivo de configuração é mostrado a seguir:

$cgrasp$ –$function$ ackley –$dimension$ 80 –$lowerbound$ -10 –$upperbound$ 10 –$startgrid$ 0.5 –$endgrid$ 0.25 –r 0.7 –k 100 –it 20 –ep 0.0000000001 –$excpt$ 2 2 4 3 -3 -1 –$setexpt$ 1 5 -1 10 –$seed$ 3145633423

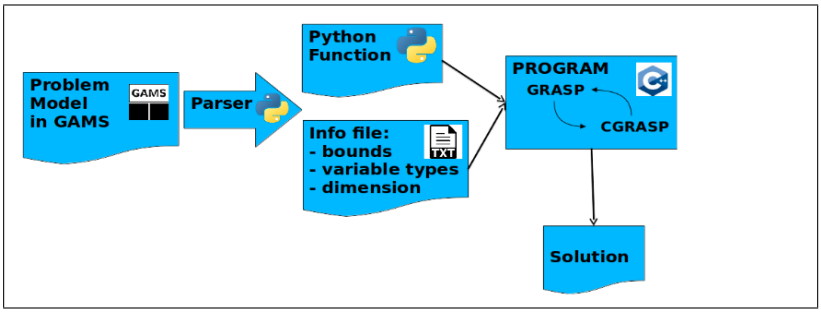


Figura 1: Arquitetura e fluxo de execução do pacote MINLP

Esse arquivo de configuração especifica que o C-$GRASP$ será invocado utilizando a função $ackley$ (oitenta variáveis de decisão), com limites inferiores -10 e superiores 10 (exceto as primeiras cinco dimensões), em uma $grid$ que vai de uma densidade de valor 0.5 até 0.25; é especificado ainda que 70% dos 100 pontos da vizinhança serão analisados. Como critério de parada a tolerância máxima foi da ordem 10^{-10}.

7 Arquitetura

Todos os modelos descritos anteriormente foram originalmente escritos na linguaguem GAMS e traduzidas em uma função Python, possibilitando que o problema seja representado em um problema de otimização global.

O pacote C-$GRASP$+$GRASP$ foi executado para as instância dos problemas selecionados. Onde o domínio do espaço de soluções foi respeitado a cada chamada intercalada, sendo discreto caso se o $GRASP$ tenha sido invocado ou contínuo caso se o C-$GRASP$ tenha sido invocado. Ao final da execução do pacote uma solução é retornada após atingir os critérios de parada citados na subseção 7.3.

7.1 C-GRASP

C-$GRASP$ é uma abordagem bem adaptada para resolver problemas de otimização global, no domínio contínuo e com restrições de caixa. O algoritmo 3 nos mostra o pseudo-código do C-$GRASP$. Dentro do laço de repetição mais interno, o vetor solução criado na fase de construção gulosa serve de entrada para a busca local. Isso possibilita ajustes finos na busca de uma boa solução dentro dos limites da $grid$. As linhas 14 e 15 nos mostra o caso de não melhoria da solução na construção gulosa e na busca local, a densidade da $grid$ é aumentada por um fator de 2. Essa nova $grid$ será utilizada na próxima iteração do C-$GRASP$. A característica multi-início é percebida no laço mais externo, onde uma nova solução inicial é gerada. A seguir discutiremos o componente específico da busca local do C-$GRASP$.

Algorithm 3 C-GRASP

1: $f^* \leftarrow \infty$
2: **while** Stopping criteria not met **do**
3: $x \leftarrow UnifRand(l, u)$
4: $h \leftarrow h_s$
5: **while** $h \geq h_e$ **do**
6: $Improv_c \leftarrow False$
7: $Improv_l \leftarrow False$
8: $[x, Improv_c] \leftarrow ConstructGreedyRandomized(x, f(.), n, h, l, u, Improv_c)$
9: $[x, Improvl] \leftarrow LocalImprovement(x, f(.), n, h, l, u, \rho, Improv_l)$
10: **if** $f(x) < f^*$ **then**
11: $x^* \leftarrow x$
12: $f^* \leftarrow f(x)$
13: **endif**
14: **if** $Improv_c = False$ **and** $Improv_l = False$ **then**
15: $h \leftarrow h/2$
16: **endif**
17: **endwhile**
18: **endwhile**
19: **return** x^*

7.1.1 Busca Local

Os parâmetros de densidade da $grid$ são cruciais na busca de melhores soluções. Se em uma iteração não houver melhorias da solução na construção gulosa e nem na busca local, a busca da solução na próxima iteração será feita em uma

$grid$ mais densa. Seja $S_h(\overline{x}) = \{x \in S \mid l \leq x \leq u, x = \overline{x} + \tau * h, \tau \in \mathbb{Z}_n\}$ o conjunto dos pontos no espaço de solução S que estão espaçadas a passos inteiros de h do ponto $\overline{x}$. Considere a seguinte estrutura de vizinhança $B_h(\overline{x}) = \{x \in S \mid x = \overline{x} + \frac{h}{\|x' - \overline{x}\|} * (x' - \overline{x}), x' \in S_h(\overline{x} \setminus \{\overline{x}\}\}$ como sendo a projeção dos pontos em $S_h(\overline{x}) \setminus \overline{x}$. A busca local é feita em uma estrutura de vizinhança olhando para uma hiperesfera de raio h e centro $\overline{x}$, e a cada investigação de um ponto vizinho se verifica se a função objetiva é minimizada. Caso afirmativo, esse novo ponto se torna o novo centro (pivô) até que o número de pontos vizinhos analisados seja atingido. Na linha 10 do algoritmo 4 mostramos que essa abordagem, conhecida como estratégia $first\text{-}improving$ se mostrou mais eficaz do que a estratégia $best\text{-}improving$ que analisa todos os pontos vizinhos para depois substituir o melhor ponto pelo ponto atual (RESENDE; MAURICIO; RIBEIRO, 2010).

Algorithm 4 Local Improvement

```
1: x* ← x
2: f* ← f(x)
3: maxNeighboor ← ρ * pointsToExamine
4: k ← maxNeighbors
5: S ← {x :|| x* − x ||²= h}
6: while k > 0 do
7:     x ← RandomlySelectElement(S)
8:     if l ≤ x ≤ u then
9:         if f(x) < f* then
10:            x* ← x
11:            f* ← f(x)
12:            S ← {x :|| x* − x ||²= h}
13:            k ← maxNeighboor
14:        else
15:            k ← k − 1
16:        endif
17:    endif
18: endwhile
19: return x*
```

7.2 GRASP

$GRASP$ é uma abordagem bem adaptada para resolver problemas de otimização global no domínio discreto. O algoritmo 5 nos mostra o pseudo-código do $GRASP$. Dentro do laço de repetição mais interno, assim como no $C\text{-}GRASP$,

o vetor solução criado na fase de construção gulosa serve de entrada para a busca local. No subitem 7.2.1 discutiremos o componente específico da busca local do $GRASP$.

Algorithm 5 GRASP

1: $f^* \leftarrow \infty$
2: **while** Stopping criteria not met **do**
3: $x \leftarrow GreedyRandomizedAlgorithm(x)$
4: $x \leftarrow LocalSearch(x)$
5: **if** $f(x) < f^*$ **then**
6: $x^* \leftarrow x$
7: $f^* \leftarrow f(x)$
8: **endif**
9: **endwhile**
10: **return** x^*

7.2.1 Busca Local

O algoritmo 6 nos mostra que na fase de busca local a estrutura de vizinhança é determinada pelos elementos espaçados em passos inteiros de h do centro $Solution^*$. O h-minímo local é definido se na vizinhança do vetor solução não há pontos que melhorem a função objetiva. A estratégia $first\text{-}improving$ foi adotada para a seleção de um novo pivô caso uma nova solução fosse encontrada na vizinhança.

Algorithm 6 LocalSearch

1: $Solution^* \leftarrow Solution$
2: **while** $Solution^*$ não for um local ótimo **do**
3: $Solution \leftarrow SelectRandomElement(Neighborhood(Solution^*))$
4: **if** $Solution < Solution^*$ **then**
5: $Solution^* \leftarrow Solution$
6: **endif**
7: **endwhile**
8: **return** $Solution^*$

7.3 C-GRASP + GRASP

Aqui introduzimos uma integração das meta-heurísticas $C\text{-}GRASP$ e $GRASP$ como método de solução para problemas de programação não linear inteiro misto. Uma mudança sutil foi feita nos critérios de parada. O critério de número de iterações foi setada para 100 de forma a possibilitar a realização de experimentos descritos na seção 8. O outro critério alterado foi obedecer a seguinte restrição:

$$GAP \leq \begin{cases} \mid erro \mid \leq \epsilon & \text{se } f(x^*) = 0 \\ \mid erro - f(x^*) \mid \leq \epsilon * \mid f(x^*) \mid & \text{se } f(x^*) \neq 0, \end{cases}$$

onde $erro = [f(x) - f(x^*)]^2 + R_1^2 +, ..., +R_n^2$, $i = 1, .., n$ sendo R_i o valor da restrição adicionada pelas variáveis de folga i.

O algoritmo 7 nos mostra o pseudo-código do pacote. As linhas 4 a 8 ilustra como a intercalação dos algoritmos $C\text{-}GRASP$ e $GRASP$ é realizada. Até que o critério de parada seja satisfeito, o vetor solução é tratado em um domínio contínuo ou discreto de forma intercalada.

Algorithm 7 solveminlp

1: $x \leftarrow UnifRand(u, l, marks)$
2: $op \leftarrow UnifRand(0, 1)$
3: **while** Stopping criteria not met **do**
4: **if** $op > 0.5$ **then**
5: $x \leftarrow$ C-GRASP(x)
6: **else then**
7: $x \leftarrow$ GRASP(x)
8: **endif**
9: **if** $f(x) < f^*$ **then**
10: $x^* \leftarrow x$
11: $f^* \leftarrow f(x)$
12: **endif**
13: **endwhile**
14: **return** x^*

8 Análise experimental

O experimento piloto foi realizado utilizando quatro $benchmarks\ functions$ ou modelos de problemas MINLP. Para cada função foram montadas 27 configurações com o objetivo de escolher a configuração cuja a qualidade da solução fosse a melhor possível. Três níveis de valores foram criados para os parâmetros h_s, h_e e ϵ : grande (G), médio (M) e pequeno (P). As configurações seguiram a seguinte condição em que $\epsilon > h_e$. A tabela 4 ilustra em ordem descendente os valores estabelecidos nas configurações de cada instância. As configurações escolhidas foram rotuladas com 3 parâmetros, $v = \{[G, M, P]\}$, $x = C\text{-}GRASP$ iterações e

$y = GRASP$ iterações. O valores de x e y foram fixados para cada configuração de uma instância de problema. Com base na qualidade da solução foram eleitas as seguintes configurações: ex1223-P-G-M-1-1, gear2-M-P-P-10-10, gear3-P-P-M-20-20 e alan-G-P-M-5-5. Caso a solução não apresentasse melhorias em 100 iterações, a execução do pacote era interrompida. Ao selecionar a melhor configuração de cada instância problema, uma simulação foi realizada com 200 corridas independentes utilizando sementes diferentes.

Todos os experimentos foram realizados em uma máquina com processador Intel Core i7 2.6Ghz e 8Gb de RAM.

Problema	h_s	h_e	ϵ
alan	[9x10^{-1},7x10^{-1},2x10^{-2}]	[4x10^{-4},2x10^{-4},10^{-4}]	[10^{-1},10^{-2},10^{-3}]
gear2	[10,5,10^{-1}]	[7x10^{-2},5x10^{-2},1x10^{-2}]	[9x10^{-1},5x10^{-1},10^{-1}]
gear3	[5,1,10^{-1}]	[7x10^{-2},5x10^{-2},2x10^{-2}]	[5x10^{-1},2x10^{-1},5x10^{-2}]
ex1223	[1,5x10^{-1},10^{-2}]	[7x10^{-3},5x10^{-3},10^{-3}]	[1,10^{-1},10^{-2}]

Tabela 4: Configurações dos problemas: níveis de valores para os parâmetros h_s, h_e e ϵ

9 Resultados

Os resultados das simulação de cada instância são apresentados na tabela 5. As colunas representam o valor mínimo, máximo, média e desvio padrão dos erros da melhor solução que o pacote C-$GRASP$+$GRASP$ alcançou.

Os TTT plot são comumente utilizados como medição de qualidade de uma heurística encontrar uma solução pelo menos tão boa quanto o valor alvo (AIEX; R. M.; RESENDE; M. G. C.; RIBEIRO, C. C.). Os valores utilizados na medição são baseados no tempo da CPU para encontrar uma solução com um valor da função objetiva pelo menos tão bom quanto um determinado valor alvo. Como a heurística é executada 200 vezes com sementes diferentes, as execuções são assumidas independentes.

Foi observado na instância ex1223 que apenas 20% dos casos, em menos de 10 segundos, a solução ótima foi encontrada respeitando os critérios de parada do C-GRASP + GRASP, sendo o restante quase em sua totalidade resultados de 100 iterações sem melhorias. Já nas instâncias gear2, e gear3 esse valor chegou a 40% dos casos. O melhor resultado obtido foi com a instância alan, onde em 80% dos casos, em torno de 2500 segundos, a solução ótima foi encontrada. A figura 2 mostra os TTT plots das simulações realizadas para cada instância de problema.

Problema	$minimo$	$maximo$	$media$	$desvio\ padrao$
alan	$2.05562e^{-05}$	0.0625916	0.0183511900405	0.0221708644376
gear2	$7.01472e^{-06}$	8.16406	1.90215475247	2.42163876076
gear3	$3.01952e^{-10}$	7.94922	2.26908897345	2.5823852343
ex1223	0.001472	1.19971	0.37689976735	0.329216583186

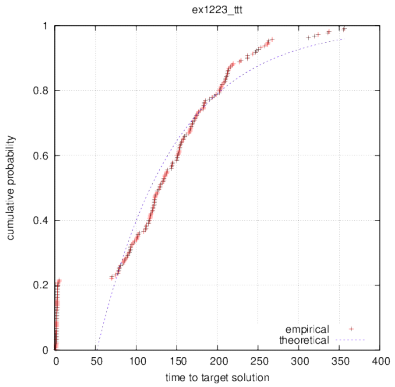
Tabela 5: Resumo estatístico das simulações

10 Conclusão

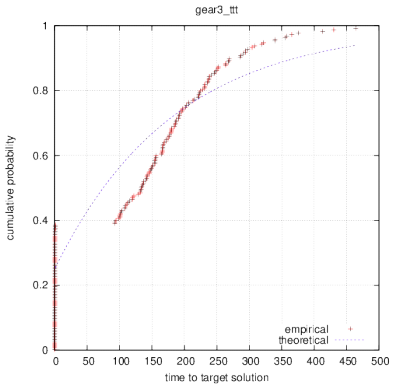
Ao contrário da maioria dos métodos atuais, a abordagem proposta não faz uso de informações $a\ priori$. Boas soluções iniciais são fundamentais para garantir a redução do tempo gasto na busca local. E após a análise experimental pode-se perceber a importância de escolher parâmetros promissores devido a natureza do problema. A partir do erro encontrado foi verificado que os resultados obtidos foram próximos aos melhores valores conhecidos.

O pacote proposto apresentou melhores soluções para o problema alan, onde em 80% dos casos a solução ótima foi encontrada respeitando os critérios de parada sem interrupção no número de iterações sem melhorias. Aos outros problemas, será necessário uma simulação com maior número de iterações sem melhorias afim de encontrar uma solução em uma escala de tempo maior.

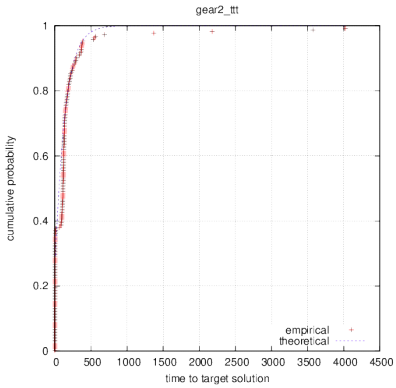
As meta-heurísticas $GRASP$ e $C\text{-}GRASP$ se mostraram eficazes e robustas



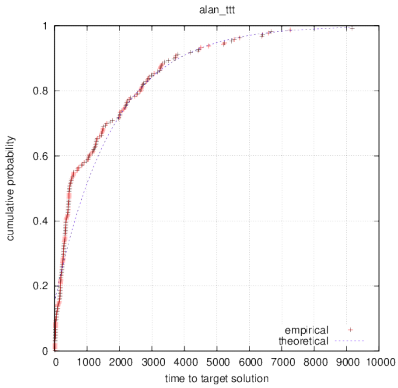
(a) ttt-ex1223



(b) ttt-gear3



(c) ttt-gear2



(d) ttt-alan

Figura 2: Distribuição de probabilidade acumulada dos tempos de execução para as instâncias ex1223,gear3,gear2 e alan

como método de solução para problemas MINLP, sendo uma abordagem bem adotada para resolver problemas de otimização global.

11 Referências

1. BELOTTI, P. et al. Mixed-integer nonlinear optimization. Acta Numerica, v. 22, p. 1–131, 2013.

2. SILVA, R. M. A. et al. A Python/C library for bound-constrained global optimization with continuous GRASP. Optimization Letters, v. 7, n. 5, p. 967–984, 2013.

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

4. ANNALS, T. et al. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. Journal of Computational and Applied Mathematics, v. 23, n. 3, p. 321–343, 2013.

5. J. Calcote. Autotools: A practitioner's guide to GNU Autoconf, Automake, and Libtool. No Starch Press, San Francisco, 2010

6. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, Handbook of Metaheuristics, pages 219-249. Kluwer Academic Publishers, 2003.

7. T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. J. of Global Optimization, 6:109–133, 1995.

8. A. R. Hedar and M. Fukushima. Tabu search directed by direct search methods for nonlinear global optimization. European Journal of Operational Research, 170:329– 349, 2006.

9. LAGUNA, M.; MARTÍ, R. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. Journal of Global Optimization, v. 33, n. 2, p. 235–255, 2005.

10. MICHALEWICZ, Z.; FOGEL, D. B. How to solve it: modern heuristics. Books.Google.Com, p. 467, 2000.

11. Z. Michalewicz and G. Nazhiyath. Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In Proceedings of the 2nd IEEE ICEC, Perth, Australia, 1995.

12. AIEX, R. M.; RESENDE, M. G. C.; RIBEIRO, C. C. TTT plots: A perl program to create time-to-target plots. Optimization Letters, v. 1, n. 4, p. 355–366, 2007.

13. Jay G.Scott. GNU, Rpm, and Solaris Software Packages.

14. MINLPLib Model. Disponível em

<http://www.gamsworld.org/minlp/minlplib/minlpstat.htm>. Acesso em 21 de Julho de 2017.

15. Embedding Python in Another Application. Disponível em

<https://docs.python.org/3/extending/embedding.html>. Acesso em 21 de Julho de 2017.

16. SILVA, R. M. A. et al. Biased random-key genetic algorithm for nonlinearly-constrained global optimization. 2013 IEEE Congress on Evolutionary Computation, CEC 2013. Anais...2013

17. MATEUS, G. R.; RESENDE, M. G. C.; SILVA, R. M. A. GRASP with path-relinking for the generalized quadratic assignment problem. Journal of Heuristics, v. 17, n. 5, p. 527–565, 2011.

18. LAARHOVEN, P. J. M. VAN; AARTS, E. H. L. Simulated Annealing, Theory with Applications. [s.l: s.n.].

19. FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, v. 8, n. 2, p. 67–71, 1989.

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

21. HIRSCH, M. J. et al. Global optimization by continuous grasp. Optimization Letters, v. 1, n. 2, p. 201–212, 2006.

22. RESENDE, M. G. C.; RIBEIRO, C. C. GRASP: Greedy randomized adaptive search procedures. In: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Second Edition. [s.l: s.n.]. p. 287–312.

23. GLOVER, F. Tabu search and adaptive memory programming - advances, applications and challenges. Interfaces in Computer Science and Operations Research, p. 1–75, 1996.