IGAS: uma meta-heurística adaptativa para a solução de problemas de otimização combinatória

Vinícius R. Máximo
Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo
Rua Talim, 330, São José dos Campos – SP – CEP 12231-280 – Brazil
vinymax10@gmail.com

Mariá C. V. Nascimento
Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo
Rua Talim, 330, São José dos Campos – SP – CEP 12231-280 – Brazil
mcv.nascimento@unifesp.br

RESUMO

A solução em tempo viável de diversos problemas de otimização combinatória ainda depende de métodos heurísticos. Em particular, as meta-heurísticas tem um papel importante na solução de vários desses problemas. Amplamente utilizada, a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), que possui iterações independentes, caracteriza-se pela boa qualidade de suas soluções. Entretanto, na maioria de suas versões, ela precisa ser hibridizada a fim de apresentar uma melhor intensificação de seu processo de busca. Tendo isso em mente que, neste artigo, propomos a *Intelligent Guided Adaptive Search* (IGAS). Baseando-se no GRASP, o IGAS se difere dessa meta-heurística devido à existência de uma etapa de aprendizado, por meio de um algoritmo de aprendizado não supervisionado chamado *Growing Neural Gas* (GNG), que auxilia na construção de soluções. Nos experimentos computacionais, para o Problema de Localização de Máxima Cobertura, atestamos que o IGAS obteve um resultado superior em comparação com outras heurísticas da literatura.

PALAVRAS CHAVE. Intelligent Guided Adaptive Search, Growing Neural Gas, Problema de Localização de Máxima Cobertura.

Área Principal: Metaheurísticas

ABSTRACT

A number of combinatorial optimization problems still relies on heuristics for getting good quality solutions in a reasonable time. In particular, metaheuristics play an important role in solving this type of problems. The widely employed metaheuristic *Greedy Randomized Adaptive Search Procedure* (GRASP) is characterized by the independence among the iterations and the good quality of its solutions. Nevertheless, in most of its versions, it is required to hybridize it with some intensification strategy for better exploring the search space. Bearing this in mind that, in this paper, we propose the *Intelligent Guided Adaptive Search* (IGAS). Similar to GRASP, IGAS differs to GRASP due to the existence of a learning stage for guiding the construction of the solutions through a unsupervised learning algorithm, the *Growing Neural Gas* (GNG). In the computational experiments, we attested that IGAS outperformed other heuristics for the Maximum Covering Location Problem.

KEYWORDS. Intelligent Guided Adaptive Search, Growing Neural Gas, Maximum Covering Location Problem.

Main Area: Metaheuristics

1. Introdução

Meta-heurísticas têm sido usadas nas últimas décadas para a solução de diversos problemas de otimização. O principal motivo é que esses métodos de solução geralmente encontram boas soluções em um tempo computacional razoável. Em particular, a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), proposta por [Feo and Resende(1989)], é um algoritmo composto por duas etapas a cada uma de sua iteração: a de construção e a de busca local. As iterações do GRASP são independentes, ou seja, uma solução encontrada em uma iteração não interfere na solução seguinte. Tem sido observado empiricamente que esse comportamento não é interessante, pois as informações obtidas em iterações anteriores de uma heurística podem colaborar positivamente se aquelas informações que caracterizam uma boa qualidade de solução são usadas em iterações futuras.

Pesquisas recentes mostram que o uso de meta-heurísticas em conjunto com técnicas de aprendizado de máquina pode melhorar significativamente o desempenho desses algoritmos [Blum and Roli(2003)]. Tendo essa premissa em vista que, neste trabalho, nós propomos uma meta-heurística chamada *Intelligent Guided Adaptive Search* (IGAS) que é um algoritmo baseado na meta-heurística GRASP em conjunto com a rede neural *Growing Neural Gas* (GNG) [Fritzke(1995)]. O IGAS é uma meta-heurística que possui três fases, construção, busca local e aprendizado. A etapa de aprendizado ocorre por meio de uma memória adaptativa representada pela GNG. Sendo assim, a cada iteração o algoritmo vai aprendendo as características de boas soluções e essas informações são utilizadas na etapa de construção das soluções seguintes.

Para atestar o desempenho das meta-heurísticas, nós abordamos o Problema de Localização de Máxima Cobertura (PLMC). O PLMC consiste em identificar os melhores locais para instalar um determinado número de facilidades com o intuito de atender o maior número de clientes possível. Cada facilidade possui um raio de cobertura que restringirá os clientes que possivelmente serão atendidos. Esse problema pertence à classe NP-difícil [Garey and Johnson(1979)].

2. Trabalhos relacionados

A meta-heurística GRASP é uma das mais utilizadas para resolver problemas de localização de facilidades [Mareth and Pizzolato(2014)]. Ela consiste em construir uma solução e em seguida aplicar uma busca local para aprimorar a qualidade da solução. Se a melhor solução $f(s^*)$ for menor que $f(s)$ então atualiza $f(s^*)$. O Algoritmo 1 apresenta um pseudocódigo do GRASP.

Algoritmo 1: GRASP

```
Entrada: Max, α
f(s*) = 0
para i = 1 até Max faça
    s ← ConstruirSolução(α).
    s ← BuscaLocal(s)
    se f(s*) < f(s) então
        s* ← s
    fim
fim
```

Para a construção da solução, o GRASP emprega uma rotina iterativa e semi-gulosa, controlada pelo parâmetro α do Algoritmo 1. Inicialmente, parte-se de uma solução vazia e, a cada passo da construção, adiciona-se à solução parcial um dos $\alpha\,|L|$ melhores candidatos, sendo L uma lista de candidatos à adição à solução parcial. Esse processo é repetido até a solução ser factível, sendo ela sujeita ao melhoramento por meio de alguma estratégia de busca local.

Pode-se notar o caráter independente que as iterações possuem no GRASP. A fim de melhor explorar os espaços de busca tirando proveito das melhores soluções encontradas

durante as iterações dos algoritmos, algumas estratégias da literatura têm como objetivo a modificação dessa meta-heurística com a inclusão de dependência entre as iterações. O DM-GRASP [Ribeiro et al.(2006)], por exemplo, utiliza mineração de dados para auxiliar o processo de busca. Conforme descrito por [Plastino et al.(2014)], inicialmente, o algoritmo executa n iterações do GRASP puro e as melhores soluções encontradas são armazenadas em um conjunto elite. Em seguida, um processo de mineração de dados é aplicado para extração de padrões desse conjunto. Após esse processo, o algoritmo executa mais n iterações híbridas, ou seja, as informações obtidas na primeira etapa auxiliará o processo de construção das soluções seguintes. Os resultados obtidos com o DM-GRASP mostraram que essa versão híbrida do GRASP obteve soluções melhores com um custo computacional menor em comparação ao GRASP. Nessa mesma linha, temos o MDM-GRASP [Plastino et al.(2014)], que consiste no DM-GRASP com a execução do processo de mineração múltiplas vezes durante o processo de busca. Os resultados obtidos pelo MDM-GRASP permitiu uma melhora tanto na qualidade das soluções quanto no tempo computacional em comparação com o DM-GRASP.

Outra variação importante do GRASP é o GRASP com *Path Relinking* proposta inicialmente por [Laguna and Martí(1999)]. Nesse algoritmo, as melhores soluções encontradas após a busca local do GRASP são armazenadas num conjunto elite. A cada iteração, a solução corrente estará sujeita ao procedimento de *Path Relinking* [Glover(1998)] utilizando uma solução guia do conjunto elite. A ideia básica do *Path Relinking* consiste em construir caminhos entre duas soluções com o intuito de encontrar soluções intermediárias de qualidade superior às soluções encontradas anteriormente.

Neste artigo, nós propomos aprimorar o método de construção utilizado no GRASP por meio da inserção de uma estratégia que permita a dependência entre as iterações. A meta-heurística proposta, a IGAS, apresentada em detalhes nas próximas seções, utiliza um algoritmo de aprendizado não supervisionado para atingir esse objetivo. A *Growing Neural Gas* (GNG) [Fritzke(1995)] foi escolhida para essa tarefa em virtude de suas características de mapeamento de soluções. O objetivo principal é que a rede seja capaz de generalizar o espaço de soluções fornecendo informações acerca das características de boas soluções para que essas possam ser utilizadas posteriormente pelo método de construção de novas soluções. Essa rede neural é explicada em detalhes na próxima seção.

3. *Growing Neural Gas*

A GNG [Fritzke(1995)] é uma rede auto-organizável com aprendizado competitivo não supervisionado. Seu surgimento ocorreu com a junção do Aprendizado Competitivo Hebbiano (CHL - *Competitive Hebbian Learning*) com o *Growing Cell Structures* (GCS). O CHL [Martinetz(1993)] introduziu o conceito de criação de arestas entre dois neurônios vencedores para um determinado dado de entrada. Já o GCS [Fritzke(1993)] determinou um mecanismo para a criação de neurônios dinamicamente.

A GNG é uma rede neural dinâmica que se adapta facilmente ao conjunto de soluções. Essa característica é importante, pois uma solução pode ser apresentada para a rede uma única vez, ou seja, o conjunto de dados que se pretende aprender é dinâmico, por esse motivo, o ideal é que a rede também seja dinâmica. Durante o processo de busca do IGAS, as regiões exploradas pelo algoritmo vão se alterando, por esse motivo, é necessário que a rede neural seja capaz de desconsiderar algumas regiões e migrar para outras onde as soluções são mais promissoras. Outro fator importante, é que o custo computacional de aprendizado não seja grande, pois esse consumo pode prejudicar o desempenho da meta-heurística. Esse é mais um dos motivos de escolha da rede GNG, pois está possui uma topologia que restringe a propagação de informação. O Algoritmo 2 representa o processo de aprendizagem da rede GNG.

A rede GNG será representada pelo conjunto R que contém todos os neurônios da rede. Cada neurônio $i \in R$ possui um vetor de pesos chamado w_i, que representa o espaço de atributos da rede. Cada solução de entrada será representado por s e o conjunto de neurônios vizinhos ao

Algoritmo 2: Apresentando uma solução a GNG.

Entrada: Solução s

$s_1 = argmin\ _{i \in R}(\mathrm{s}-w_i)^2$
$s_2 = argmin\ _{i \in R \setminus \{s_1\}}(\mathrm{s}-w_i)^2$
se *Não existe conexão entre* s_1 *e* s_2 **então**
  Inicia conexão entre s_1 e s_2.
fim
Inicializa a idade da aresta entre s_1 e s_2.
Atualiza o erro do dado de entrada em s_1: $E_{s_1} += (\mathrm{s}-w_{s_1})^2$
Atualiza ε_b conforme Equação (5).
Atualiza os pesos de s_1: $w_{s_1} = \varepsilon_b(\mathrm{s}-w_{s_1})$
se $|R| == 2$ **então**
  Atualiza os pesos da vizinhança de s_1: $w_{viz} = \varepsilon_b(\mathrm{s}-w_{viz}) \quad \forall\ viz \in \delta(s_1)$
fim
senão
  Atualiza os pesos da vizinhança de s_1: $w_{viz} = \varepsilon_n(\mathrm{s}-w_{viz}) \quad \forall\ viz \in \delta(s_1)$
fim
Incrementa em uma unidade a idade das arestas $\{s_1, viz\} \ \forall\ viz \in \delta(s_1)$
Remove as arestas com idade maior que a_{max}.
Remove os neurônios desconexos: $R = R \setminus \{i\} \quad \forall\ i$ tal que $\delta(i) = \varnothing$
se *O número de dados analisados for múltiplo de* λ *e* $|R| < max_R$ **então**
  $q = argmax\ _{i \in R}(E_i)$
  $f = argmax\ _{viz \in \delta(q)}(E_{viz})$
  Adiciona um novo neurônio r entre q e f: $R = R \cup \{r\}$ e $w_r = \frac{(w_q + w_f)}{2}$
  Remove a aresta entre q e f e adiciona r à vizinhança de q e de f.
  Diminui o erro de q e f por meio do fator γ e inicializa o erro de r com a média de q e f: $E_q = \gamma E_q \quad E_f = \gamma E_f \quad E_r = \frac{(E_q + E_f)}{2}$
fim
Decrementa o erro para todos os neurônios: $E_i = \beta E_i \ \forall\ i \in R$

neurônio i será representado por $\delta(i)$. Outro parâmetro importante é a variável λ que indica o número de iterações necessárias para inserção de um novo neurônio. A taxa de atualização do neurônio vencedor e de sua vizinhança é descrita por ε_b e ε_n respectivamente.

Inicialmente, como é possível verificar no Algoritmo 2, a rede GNG é composta por apenas 2 neurônios. Ao longo do aprendizado ocorre periodicamente a inserção de um novo neurônio até o limite de max_R que representa o tamanho máximo permitido para rede. O local ideal para inserção é determinado por meio do erro acumulado em cada neurônio. O novo neurônio será inserido próximo ao neurônio com maior erro acumulado. Esse erro é composto pela distância do dado de entrada com relação a esse neurônio. Esse valor é acumulado ao longo do processo de aprendizagem. Contudo, os erros antigos não devem ser relevantes com relação aos novos erros, por esse motivo, o erro é decrementado de uma determinada porcentagem a cada iteração do algoritmo, mantendo assim a magnitude dos erros atuais mais relevantes do que os erros anteriores.

Ao apresentar um padrão à rede, ocorre o processo de competição que elegerá os dois neurônios mais próximos do dado de entrada. O primeiro mais próximo é chamado de neurônio vencedor ou simplesmente s_1 e o segundo mais próximo é chamado de s_2. Nesse momento uma conexão entre esses neurônios é estabelecida. A partir dessa conexão, ocorre o processo de aprendizagem que visa deslocar o neurônio vencedor e a sua vizinhança em direção ao dado de entrada.

O neurônio vencedor terá uma taxa de atualização maior que a sua vizinhança.

Cada conexão possui uma idade que é inicializada em zero quando é construída, ou quando estiver entre os dois neurônios vencedores para um determinado dado de entrada. As idades das arestas de um neurônio serão incrementadas em uma unidade sempre que esse for vencedor para um determinado dado de entrada. A idade de uma aresta será utilizada para que o algoritmo possa removê-las caso fiquem muito velhas. O processo de remoção de arestas é necessário pois se a idade de uma aresta é maior que um determinado limiar, indica que não existe mais um dado de entrada entre esses neurônios. Nesse caso, essa aresta deve ser removida e pode ocorrer o caso em que alguns neurônios fiquem desconexos da rede. Se isso ocorrer, esses neurônios devem ser removidos.

Para que a GNG tenha um melhor desempenho, uma pequena alteração foi feita no algoritmo original. No início da aprendizagem, quando a rede possui apenas dois neurônios, o parâmetro de ajuste da rede para os neurônios vizinhos de s_1 foi alterado para ε_b em vez de ε_n. Essa alteração foi necessária pois, no início, as soluções são bem dinâmicas e suas características mudam com maior frequência, sendo necessário uma taxa de aprendizagem maior nos dois únicos neurônios da rede.

Neste artigo, o Problema de Localização de Máxima Cobertura (PLMC) será alvo para resolução por meio do IGAS. Esse problema é brevemente explicado na próxima seção.

4. Formulação do PLMC

A seguir, apresentamos uma formulação matemática para o PMLC [Church and ReVelle(1974)].

$$\text{Maximizar} \sum_{i \in N} D_i x_i$$

sujeito a:

$$\sum_{j \in S_i} y_j \geq x_i \qquad \forall i \in N \tag{1}$$

$$\sum_{j \in M} y_j = p \tag{2}$$

$$x_i \in \{0,1\} \qquad \forall i \in N \tag{3}$$

$$y_j \in \{0,1\} \qquad \forall j \in M \tag{4}$$

em que:

- $N = \{1, 2, \ldots, n\}$ é o conjunto de clientes a serem atendidos.
- $M = \{1, 2, \ldots, m\}$ é o conjunto de possíveis locais para instalação das facilidades.
- p representa o número de facilidades a serem instaladas.
- D_i representa a demanda do cliente $i \in N$.
- x_i é uma variável binária que indica se a demanda do cliente $i \in N$ é atendida ou não.
- y_j é uma variável binária que indica se a facilidade está instalada no local $j \in M$ ou não.
- d_{ij} é a distância entre o cliente i e a facilidade j.
- S é o raio de cobertura.
- $S_i = \{j \in M \mid d_{ij} \leq S\}$ é o conjunto de facilidades que podem atender o cliente $i \in N$.

A função objetivo tem o intuito de maximizar a demanda atendida. Quanto maior a demanda de um cliente, maior será o interesse em atendê-lo. As restrições (1) estabelecem que um cliente i só será atendido se existir uma facilidade j tal que $d_{ij} \leq S$. A restrição (2) fixa o número de facilidades instaladas para p.

5. *Intelligent Guided Adaptive Search*

O algoritmo proposto consiste em construir soluções melhores a partir das informações obtidas pela GNG. O Algoritmo 3 apresenta um pseudocódigo do IGAS.

Algoritmo 3: IGAS

Data: Max, α, φ, r, q
$f(s^*) = 0$
para $i = 1$ **até** Max **faça**
  Escolhe aleatoriamente $n \in R$.
  $s \leftarrow$ ConstruirSolução (α,φ, n, r)
  $s \leftarrow$ BuscaLocal(s)
  $limiar = f(\bar{s}) + q \times (f(s^*) - f(\bar{s}))$
  se $f(s) > limiar$ **então**
    Apresenta s para a GNG.
  fim
  se $f(s^*) < f(s)$ **então**
    $s^* \leftarrow s$
  fim
fim

É possível observar no Algoritmo 3 que, para que a GNG possa contribuir no processo de busca, é necessário que as soluções apresentadas à rede neural sejam de boa qualidade. Por esse motivo, foi utilizado um limiar que restringe as soluções que são apresentadas para a rede. O parâmetro q, indica o fator mínimo de qualidade que as soluções precisão ter para serem apresentadas à GNG. Neste trabalho, o espectro das soluções variam entre a média das soluções obtidas após a Busca Local indicado por $f(\bar{s})$ e a melhor solução $f(s^*)$.

A solução para o PLMC será representada pela variável y_i, que indica se uma facilidade está instalada no local i, ou não. Se numa determinada região de boas soluções, uma localidade i sempre possui uma facilidade instalada, então se espera que outras boas soluções que estejam nessa região também tenham uma facilidade em i. Partindo desse pressuposto, o neurônio poderá indicar para uma determinada região quais os melhores locais para se instalar uma localidade.

Com o intuito de aprimorar o aprendizado, é interessante que boas soluções tenham maior influência no aprendizado da rede do que soluções não tão boas. Para fazer essa distinção, o parâmetro ε_b foi adaptado para variar dentro de um determinado intervalo. Para boas soluções utiliza-se ε_b grande, desse modo, o passo do neurônio será maior em direção a essa solução. Para soluções não tão boas, então aplica-se um ε_b pequeno, assim o neurônio não sofrerá influência massiva de soluções não tão qualificadas.

$$\varepsilon_b = \varepsilon_b^{min} + \frac{(f(s) - f(.))}{(f(s^*) - f(.))} \times (\varepsilon_b^{max} - \varepsilon_b^{min}) \tag{5}$$

A Equação (5) representa o processo de atualização de ε_b para uma solução s. Os valores de $f(s^*)$ e $f(.)$ representam a melhor solução e o limiar, respectivamente. Os valores de ε_b^{max} e ε_b^{min} são parâmetros estabelecidos pelo usuário.

Como já foi mencionado, neste artigo, resolveremos heuristicamente o PLMC por meio da meta-heurística proposta, o IGAS. A seguir apresentamos a fase construtiva e de busca local da estratégia proposta para resolver o PLMC.

5.1. Fase de Construção

O GRASP possui uma heurística de construção semi-gulosa, ou seja, o local escolhido não é o melhor candidato, e sim um dos melhores. Para fazê-lo é necessário construir a RCL

(*Restricted Candidate List*) com os melhores locais e escolher aleatoriamente um local $s \in RCL$ para se instalar uma facilidade. Existe um parâmetro α que estabelece um limiar que restringe os locais que serão inseridos na lista RCL. Esse limiar garante que apenas os melhores locais serão escolhidos para instalação de uma facilidade. O valor de α varia no intervalo $[0, 1]$ e quanto menor o valor de α menor será o tamanho da RCL.

Para o PLMC, empregamos a representação da solução, como sendo composta basicamente por locais que representam a posição dos clientes. Para os problemas analisados, as facilidades podem ser instaladas em qualquer cliente. Portanto, o número de facilidades disponíveis para escolha é igual ao número de clientes, ou seja, $M = N$.

O método de construção de uma solução consiste em encontrar a melhor localização para instalação de todas as facilidades. O procedimento de construção segue uma heurística gulosa chamada Algoritmo de Acréscimo Guloso (AAG) [Church and ReVelle(1974)]. Nessa heurística, o primeiro local a ser escolhido para se instalar uma facilidade é aquele que atende a maior demanda em seu raio de cobertura. Em seguida, é escolhido o local que atende a maior demanda dentre os locais não atendidos, e assim sucessivamente até que tenhamos p facilidades instaladas. O benefício de um local c definido como b_c será calculado conforme Equação (6):

$$b_c = \sum_{a \in A_c} D_a \qquad \text{tal que} \qquad A_c = \{l \in C_c \mid t_l = 0\} \tag{6}$$

Na Equação (6), t_l indica o número de facilidades que atendem o local l, enquanto C_c é um conjunto contendo todos os locais que estão dentro do raio de cobertura do local c. O Algoritmo 4 descreve o procedimento de construção de uma solução pelo GRASP.

Algoritmo 4: Construção das soluções pelo GRASP.

Entrada: α, $p' \leq p$
1 **para** $i = 1$ **até** p' **faça**
2 $\quad CL = \{a \in M \mid y_a = 0\}$.
3 $\quad$ Ordene CL em ordem decrescente do benefício conforme Equação (6).
4 $\quad RCL = \{l_j \in CL \mid$ a posição de l_j em $CL \leq \alpha|CL|\}$
5 $\quad$ Selecione aleatoriamente um local $l \in RCL$ e adicione uma facilidade em l.
6 **fim**

O processo de construção de uma solução com o apoio da rede GNG ocorre a partir de um neurônio $n \in R$ que é escolhido aleatoriamente na rede. O neurônio será utilizado para indicar o benefício de cada local do problema. O valor de $w_{n,c}$ representa o peso do local c no neurônio n. Como o atributo que foi passado para a rede foi a variável binária $y \in \{0, 1\}$, então o valor de w pertence ao intervalo de $[0, 1]$. Se $w_{n,c} \approx 1$ significa que grande parte das soluções para o qual o neurônio n foi vencedor possuíam uma facilidade em c. Por esse motivo, o critério de escolha das facilidades será conforme indicado pelo peso $w_{n,c}$.

O Algoritmo 5 mostra o processo de construção de uma solução a partir de um neurônio. O parâmetro φ tem a mesma finalidade do α no Algoritmo 4. A intenção principal é que as informações fornecidas pela rede neural sejam utilizadas com uma certa diversidade. Essa abordagem evita que a rede neural fique viciada. O parâmetro r representa a confiança do método de construção no neurônio. O valor de r indica a porcentagem da solução que será construída conforme informações da rede. Conforme item 8 do algoritmo, o restante da solução será construído conforme o Algoritmo 4.

5.2. Busca Local

A busca local consiste num método que melhora uma solução até que se obtenha um ótimo local. Conforme a estrutura de vizinhança, a busca local numa solução s consiste em encontrar um

Algoritmo 5: Construção das soluções pelo IGAS.

Entrada: α, φ, n, r
para $i = 1$ **até** $(r \times p)$ **faça**
  $CL = \{a \in M \mid y_a = 0\}$
  Ordene CL em ordem decrescente de w_n.
  $RCL = \{l_j \in CL \mid$ a posição de l_j em $CL \leq \varphi\}$
  Selecione aleatoriamente um local $l \in RCL$ e adicione uma facilidade em l.
fim
Adicione o restante das facilidades conforme o Algoritmo 4.

vizinho $s' \in N(s)$ tal que $f(s') > f(s)$ e atualizar o valor de $f(s)$. Esse procedimento é repetido até que não exista mais vizinhos que possam melhorar a solução corrente. O Algoritmo 6 descreve o procedimento de busca local.

Algoritmo 6: Busca Local.

Entrada: Solução s
enquanto $\exists\, s' \in N(s)$ *tal que* $f(s') > f(s)$ **faça**
  $s \leftarrow s'$
fim

A vizinhança adotada neste artigo consiste do conjunto de soluções que se diferenciam de um movimento de troca do local de uma facilidade. O conjunto de todas as soluções s' construídas a partir de s por meio de um movimento de troca da posição de uma facilidade é definido como $N(s)$. Para que ocorra uma troca entre os locais i, j, é necessário que $y_i = 1$ e $y_j = 0$. Nesse caso, o local i deixa de ser facilidade e j passa a ser uma facilidade. O Algoritmo 7 descreve o procedimento de instalação de uma facilidade no local a. O método de adição de facilidade consiste em incrementar o valor da função objetivo com a demanda de todos os locais contidos no raio de cobertura dessa facilidade, caso esse local ainda não tenha sido atendido por nenhuma outra facilidade. O Algoritmo 8 demonstra o procedimento de remoção de uma facilidade. O método de remoção consiste em decrementar do valor da função objetivo, a demanda de todos os locais contidos no raio de cobertura que são atendidos por apenas uma facilidade. A complexidade do algoritmo de troca de uma facilidade é inerente ao tamanho do conjunto C. Desse modo, como $|C| \leq N$, então a complexidade de ambos os métodos é $O(n)$. Essa estrutura de vizinhança mantém atualizado o valor da função objetivo.

Algoritmo 7: Adiciona Facilidade.

Entrada: Local a
$y_a = 1$
para cada $c \in C_a$ **faça**
  se $t_c == 0$ **então**
    $f = f + D_c$
  fim
  $t_c = t_c + 1$
fim

Algoritmo 8: Remove Facilidade.

Entrada: Local a
$y_a = 0$
para cada $c \in C_a$ **faça**
  se $t_c == 1$ **então**
    $f = f - D_c$
  fim
  $t_c = t_c - 1$
fim

6. Resultados

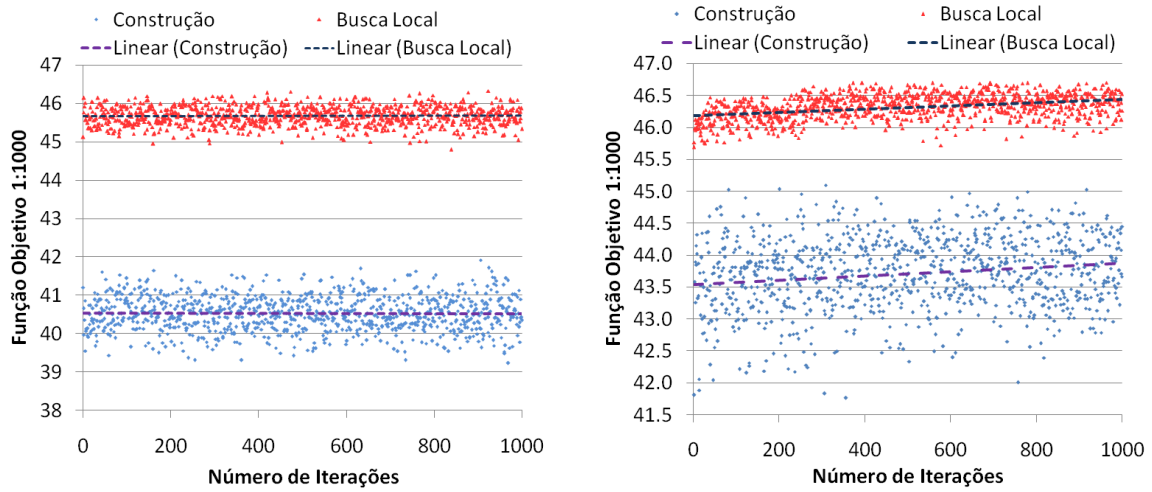
Nós realizamos os testes computacionais utilizando uma metodologia similar a apresentada por [ReVelle et al.(2008)]. Conforme [ReVelle et al.(2008)], para compor as instâncias do pro-

blema, nós escolhemos aleatoriamente, seguindo uma distribuição uniforme, os locais para as facilidades por meio de coordenadas cartesianas com duas dimensões. Cada local é representado por um ponto (x, y) tal que $x, y \in [0, 100]$. A distância utilizada entre os locais foi a distância Euclidiana e a demanda de cada local foi escolhida aleatoriamente no intervalo $[0, 100]$. Nós utilizamos instâncias com 1000, 2000 e 3000 locais seguindo as configurações conforme Tabela 1.

A implementação dos algoritmos foi feita na linguagem JAVA e o computador utilizado durante os testes possui a seguinte configuração: Intel(R) Xeon(R) CPU E5645 @ 2.40GHz com 12MB de cache e 2 GB RAM.

Tabela 1: Configuração das instâncias analisadas.

n	S	p
$1000, 2000, 3000$	7	$60, 68, 76$
$1000, 2000, 3000$	9	$35, 40, 45$
$1000, 2000, 3000$	11	$25, 29, 32$



(a) Qualidade das soluções pelo GRASP.

(b) Qualidade das soluções pelo IGAS.

Figura 1: Qualidade das soluções na etapa de Construção e Busca Local do GRASP e do IGAS.

Inicialmente nós analisamos o comportamento das soluções encontradas após a fase de construção e de busca local do GRASP e do IGAS. Sendo assim, nós ilustramos na Figura 1 a qualidade das soluções no processo de construção e de busca local do GRASP e do IGAS. A instância utilizada nesse teste foi a $n = 1000$, $p = 7$ e $S = 60$, e os algoritmos foram executados até 1000 iterações. A Figura 1(a) demonstra que existe em gap entre as soluções construídas e as soluções obtidas a partir da busca local. Também é possível identificar através do ajuste por meio do Método dos Mínimos Quadrados representado por Linear, que ambos os pontos possuem uma qualidade estável durante a execução do algoritmo. A Figura 1(b) ilustra o resultado para o IGAS. Esse gráfico demonstra um gap menor entre as soluções construídas e as soluções da busca local. Esse comportamento é interessante pois a busca local, não tem muito esforço computacional para chegar em um ótimo local. Podemos perceber também que a qualidade das soluções construídas é superior às soluções produzidas pelo GRASP. O mais interessante no algoritmo proposto é que a qualidade das soluções construídas e das soluções obtidas a partir da busca local vão melhorando ao longo da execução do algoritmo.

Os parâmetros utilizados pela GNG depois de experimentos computacionais estão descritos conforme Tabela 2. Para o IGAS nós utilizamos os seguintes parâmetros: $\alpha = 0.1, \varphi = 25, r =$

ε_b^{max}	ε_b^{min}	ε_n	γ	a_{max}	λ	β	max_R
0.5	0.01	0.0005	0.5	120	50	0.995	15

Tabela 2: Parâmetros utilizados na rede GNG.

0.8 e $q = 0.7$. Já o valor de parâmetro utilizado pelo GRASP foi $\alpha = 0.15$.

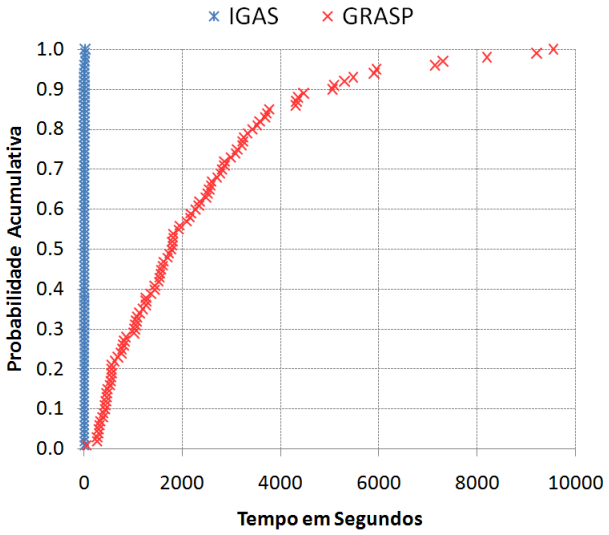


Figura 2: TTTPlot's do IGAS e do GRASP.

Para ilustrar o ganho computacional dos algoritmos, nós utilizamos o bem conhecido *Time-To-Target Plots (TTT-plots)* [Aiex et al.(2007)], ilustrado na Figura 2. Esse gráfico demonstra a probabilidade acumulativa para atingir um valor de função objetivo maior ou igual a 46500 para a instância $n = 1000$, $p = 7$ e $S = 60$. Esse valor foi escolhido tendo em vista o custo computacional do GRASP. Os algoritmos foram executados 100 vezes e o tempo gasto para atingir a solução alvo foi medido em segundos. Em média, o IGAS gastou 4 segundos para atingir a solução alvo enquanto o GRASP gastou 2306 segundos. Esse resultado representa um ganho de mais de 570 vezes mais rápido.

Para analisar a performance da meta-heurística proposta, nós a submetemos ao conjunto de 27 instâncias descritas no início dessa seção juntamente com o método GRASP. Para termos uma noção da dificuldade natural dessas instâncias nós também submetemos essas instâncias ao solver CPLEX [Gay(2011)] com um tempo limite de 2000 segundos.

A Tabela 3 apresenta os resultados para o CPLEX, GRASP e IGAS. Os algoritmos GRASP e IGAS foram executados 30 vezes e cada rodada com um tempo limite de 2000 segundos. O tempo repostado na tabela para o GRASP e IGAS é o tempo em que o algoritmo encontrou a melhor solução. As soluções apresentadas pelo CPLEX com tempo de 2000 segundos não são certificadas como soluções ótimas. Para analisar a qualidade das soluções utilizaremos o gap das soluções obtidas com relação à solução da Relaxação Linear (LP) conforme Equação (7).

$$Gap = 100 \times \frac{(LP - \text{Solução})}{LP} \tag{7}$$

Conforme os resultados apresentados na Tabela 3, podemos verificar que o gap médio obtido pelas piores soluções encontradas pelo IGAS é inferior ao gap médio obtido para as melhores soluções encontradas pelo GRASP. Também podemos identificar que o CPLEX encontra rapidamente a solução ótima para as instâncias com $n = 1000$, no entanto para as instâncias com $n = 2000$ e $n = 3000$ o algoritmo não conseguiu finalizar a busca pela solução ótima com tempo

Tabela 3: Resultado para as instâncias analisadas.

			CPLEX		GRASP				IGAS			
n	S	p	Melhor	Tempo	Melhor	Pior	Mediana	Tempo	Melhor	Pior	Mediana	Tempo
1000	7	60	**0,40**	3,6	0,83	1,31	1,03	1141,8	**0,40**	0,64	**0,40**	747,4
		68	**0,41**	17,1	0,96	1,43	1,22	825,3	**0,41**	0,59	0,42	1057,8
		76	**0,06**	308,0	0,45	0,73	0,56	979,8	**0,06**	0,21	0,09	1205,7
	9	35	**0,66**	8,6	0,68	1,12	0,87	765,4	**0,66**	**0,66**	**0,66**	530,6
		40	**0,34**	7,7	0,72	1,16	1,01	882,2	**0,34**	0,53	0,48	510,2
		45	**0,25**	579,5	0,59	0,88	0,80	986,5	**0,25**	0,43	0,32	869,0
	11	25	**0,62**	3,4	**0,62**	1,16	0,73	934,1	**0,62**	0,68	**0,62**	320,6
		29	**0,72**	130,7	**0,72**	1,13	0,94	1239,8	**0,72**	0,88	**0,72**	140,1
		32	**0,29**	451,9	0,42	0,60	0,53	932,8	**0,29**	0,44	0,32	825,2
2000	7	60	1,07	2000	1,42	1,79	1,61	882,8	**0,98**	1,27	1,06	918,3
		68	1,17	2000	1,74	2,28	2,09	834,3	**1,01**	1,39	1,19	1288,3
		76	0,76	2000	0,94	1,21	1,11	970,1	**0,23**	0,41	0,32	1370,9
	9	35	1,45	2000	1,66	2,23	1,91	962,1	**1,25**	1,64	1,28	830,6
		40	1,48	2000	1,91	2,53	2,23	854,5	**1,47**	1,71	1,59	1002,6
		45	0,84	2000	1,10	1,71	1,49	1111,6	**0,71**	0,96	0,80	891,8
	11	25	1,19	2000	**1,12**	1,73	1,32	812,1	**1,12**	1,55	**1,12**	597,9
		29	**1,03**	2000	1,20	1,48	1,30	1338,2	**1,03**	1,43	1,05	894,3
		32	0,65	2000	0,54	0,88	0,78	955,4	**0,38**	0,73	0,42	669,7
3000	7	60	1,64	2000	1,79	2,58	2,32	976,4	**1,22**	1,77	1,40	1265,4
		68	1,64	2000	2,18	2,97	2,67	1123,1	**1,20**	1,80	1,49	1415,4
		76	1,58	2000	1,25	1,78	1,55	1030,1	**0,52**	0,99	0,69	1522,9
	9	35	1,18	2000	1,36	2,43	1,81	1050,9	**1,09**	1,67	**1,09**	484,8
		40	1,84	2000	2,08	2,55	2,35	882,2	**1,42**	2,02	1,61	1035,0
		45	1,37	2000	1,43	2,02	1,82	1126,2	**0,98**	1,57	1,12	1016,4
	11	25	**0,61**	2000	0,69	0,93	0,84	1042,5	**0,61**	**0,61**	**0,61**	425,2
		29	1,61	2000	1,16	1,67	1,48	1009,7	**1,10**	1,56	1,26	699,7
		32	0,87	2000	0,73	1,07	0,90	802,7	**0,44**	0,76	0,60	665,2
Média			0,953	1389,3	1,122	1,606	1,380	979,7	**0,760**	1,070	0,842	859,3

limite de 2000 segundos. Fazendo uma análise comparativa entre as soluções obtidas, verificamos que o IGAS obteve uma solução mediana melhor ou igual a solução encontrada pelo CPLEX em 19 das 27 instâncias.

7. Conclusão

Neste artigo, nós apresentamos uma nova meta-heurística, a IGAS, empregando o paradigma de redes neurais para explorar regiões promissoras no espaço de soluções. Com isso, inserimos a uma meta-heurística com iterações independentes uma memória adaptativa que aprende com as soluções encontradas em iterações prematuras e que auxilia a construção das soluções posteriores. Para atestar a estratégia proposta, nós estudamos o Problema de Localização de Máxima Cobertura, comparando as soluções obtidas pelo GRASP e pelo IGAS. Identificamos que o IGAS atingiu um resultado consideravelmente superior ao GRASP.

Como trabalhos futuros, pretende-se submeter a meta-heurística proposta a outros problemas de otimização. O algoritmo *Path Relinking* mostrou um bom desempenho em conjunto com GRASP [Laguna and Martí(1999)], como o IGAS também é um algoritmo construtivo, sua junção com o *Path Relinking* pode trazer melhoras.

8. Agradecimentos

Agradecemos à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo apoio financeiro. Gostaríamos de agradecer também ao Prof. Dr. André Carlos Ponce de Leon Ferreira de Carvalho e ao Davi Pereira dos Santos da Universidade de São Paulo (USP) pelo suporte nos experimentos computacionais disponibilizando o acesso à servidores da USP.

Referências

[Aiex et al.(2007)] **Aiex, R. M., Resende, M. G. C., and Ribeiro, C. C.** (2007). Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, 1(4):355–366.

[Blum and Roli(2003)] **Blum, C. and Roli, A.** (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308.

[Church and ReVelle(1974)] **Church, R. and ReVelle, C.** (1974). The maximal covering location problem. *Papers of the Regional Science Association*, 32(1):101–118.

[Feo and Resende(1989)] **Feo, T. and Resende, M.** (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71.

[Fritzke(1993)] **Fritzke, B.** (1993). Growing cell structures - a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460.

[Fritzke(1995)] **Fritzke, B.** (1995). A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press.

[Garey and Johnson(1979)] **Garey, M. R. and Johnson, D. S.** (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[Gay(2011)] **Gay, D. M.** (2011). *IBM ILOG CPLEX Optimization Studio CPLEX User's Manual Version 12 Release 4*. IBM.

[Glover(1998)] **Glover, F.** (1998). A template for scatter search and path relinking. In Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., and Snyers, D., editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 1–51. Springer Berlin Heidelberg.

[Laguna and Martí(1999)] **Laguna, M. and Martí, R.** (1999). Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52.

[Mareth and Pizzolato(2014)] **Mareth, T. and Pizzolato, N. D.** (2014). Mapeamento da utilização dos métodos/algoritmos aplicados na resolução de problemas de localização. *O Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, 6:154–182.

[Martinetz(1993)] **Martinetz, T.** (1993). Competitive hebbian learning rule forms perfectly topology preserving maps. In Gielen, S. and Kappen, B., editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN-93), Amsterdam*, pages 427–434, Heidelberg. Springer.

[Plastino et al.(2014)] **Plastino, A., Barbalho, H., Santos, L., Fuchshuber, R., and Martins, S.** (2014). Adaptive and multi-mining versions of the dm-grasp hybrid metaheuristic. *Journal of Heuristics*, 20(1):39–74.

[ReVelle et al.(2008)] **ReVelle, C., Scholssberg, M., and Williams, J.** (2008). Solving the maximal covering location problem with heuristic concentration. *Computers & Operations Research*, 35(2):427 – 435. Part Special Issue: Location Modeling Dedicated to the memory of Charles S. ReVelle.

[Ribeiro et al.(2006)] **Ribeiro, M., Plastino, A., and Martins, S.** (2006). Hybridization of grasp metaheuristic with data mining techniques. *Journal of Mathematical Modelling and Algorithms*, 5(1):23–41.