Algoritmo Genético Construtivo aplicado ao Problema Generalizado de Atribuição

Marcelo Gonçalves Narciso
Embrapa Informática Agropecuária
Av. Dr. André tosello, s/n, Unicamp
13083-970 – Campinas - SP

Luiz Antônio Nogueira Lorena
LAC/INPE – Instituto Nacional de Pesquisas Espaciais
Av. dos Astronautas, 758 – São José dos Campos – SP

Palavras chave: Algoritmo Genético Construtivo (AGC), Problema Generalizado de Atribuição (PGA), desempenho do AGC.

Resumo

Neste trabalho apresentamos uma aplicação da metaheurística denominada Algoritmo Genético Construtivo (AGC) ao Problema Generalizado de Atribuição (PGA).

O AGC apresenta algumas características inovadoras em relação aos algoritmos genéticos tradicionais (AGT), tais como, população formada apenas de estruturas e/ou esquemas, processo proporcional de avaliação, recombinação entre esquemas, população dinâmica, mutação em estruturas completas, e a possibilidade de uso de heurísticas na representação dos esquemas e/ou estruturas (veja http://www.lac.inpe.br/~lorena/teseJC/CGA-tese.pdf para mais informações sobre o AGC).

O PGA pode ser descrito como o problema de atribuição de n itens a m mochilas, n > m, de forma que cada item seja atribuido a exatamente uma mochila, respeitando suas capacidades.

Em nossa aplicação do AGC ao PGA, este é considerado como um problema de *clustering*. Uma representação binária é usada para esquemas e estruturas, e heurísticas de atribuição particionam os itens nas mochilas. Esquemas não levam em conta todos os dados do problema. São recombinados podendo gerar novos esquemas ou estruturas. Novos esquemas são avaliados de forma proporcional e podem entrar na população se passar por um teste de evolução. Quando estruturas completas resultam da recombinação de esquemas ou quando bons esquemas são complementados, as estruturas sofrem mutação. A melhor estrutura gerada é guardada no processo.

Testes computacionais foram realizados com bons resultados, usando instâncias de larga escala disponíveis na literatura.

Key words: Construtive Genetic Algorithm (CGA) , Generalized Assignment Problem (GAP), performance of CGA.

Abstract

In this paper we present an application of the metaheuristic called Construtive Genetic Algorithm (CGA) to the Generalized Assignment Problem (GAP).

The CGA presents some new features comparing to traditional genetic algorithm (TGA), such as population formed only by structures and/or schemes, proportional process of valuation, recombination among schemes, dynamic population, mutation in complete structures, and the possibility of using heuristics in representation of schemes and/or structures (see http://www.lac.inpe.br/~lorena/teseJC/CGA-tese.pdf for more information about CGA).

The GAP can be described as a problem of assignment of n itens to m knapsacks, n>m, so that each iten can be assigned to exactly one knapsack, according to the constraints of problem.

In our application of CGA to GAP, it is considered as a clustering problem. A binary representation is used for schemes and structures, and assingnment heuristics attribute itens to knapsacks. Schemes do not consider all the problem data. The schemes are recombined, and they can produce new schemes or structures. New schemes are evalueted proportionaly and can be added to the population if they got succeded in a evolution test. When complete structures result from recombination of schemes or when good schemes are completed, then the mutation in structures is observed. The best structure generated is kept in the process.

Computational tests have been done with good results using instances of large scale available in the literature.

1. Introdução

O problema de se maximizar rendimentos (ou lucros, produção, etc.) ou minimizar custos (ou tempo para realizar tarefas, perdas, gastos com ingredientes, etc.) ao se atribuir n tarefas a m agentes, n > m, tal que cada tarefa seja atribuída a apenas um único agente, levando-se em conta as restrições de capacidade de cada agente, é conhecido na literatura como problema generalizado de atribuição *(PGA)*. Muitos problemas da vida real podem ser modelados como um *(PGA)*. Podemos citar como exemplos, problemas de investimento de capitais [5], alocação de espaço de memória [4], projeto de redes de comunicação com restrições de capacidades para cada nó da rede [5], atribuição de tarefas de desenvolvimento de software a programadores [1], atribuição de tarefas a computadores em uma rede [1], subproblemas de roteamento de veículos [6] , e outros. Como uma conseqüência lógica de sua aplicação, um grande esforço tem sido realizado no desenvolvimento de algoritmos para este problema [10].

Este problema é NP-hard [4, 7, 17] e poucas heurísticas existem na literatura para a resolução deste problema de forma a encontrar boas soluções viáveis. Dentre as que mais se destacam podemos citar as de Klastorin [11], Martello e Toth [15] e Lorena e Narciso [14].

O *(PGA)* É formulado matematicamente (versão de minimização) como:

$$v(PGA) = min \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}.x_{ij}$$

(PGA)

$$\text{sujeito a} \quad \sum_{j=1}^{n} w_{ij}.x_{ij} \leq b_i, \ i \in M = \{1,...,m\} \quad (1.1)$$

$$\sum_{i=1}^{m} x_{ij} = 1, \ j \in N = \{1,...,n\} \quad (1.2)$$

$$x_{ij} \in \{0,1\}, \ i \in M, \ j \in N \quad (1.3)$$

As restrições (1.1) impõem que os pesos dos itens escolhidos não devem exceder à capacidade de cada mochila e as restrições (1.2) impõem que cada item deve ser atribuído a somente uma mochila.

Neste trabalho, o enfoque será a resolução do PGA usando-se o AGC. O AGC teve bons resultados nos trabalhos de Furtado [8], Lorena e Furtado [12], Lorena e Lopes [13], e Ribeiro e Lorena [16]. A importância deste trabalho se reside no fato de ainda não haver uma proposta para fornecer soluções ao PGA usando-se o AGC. Além disso, visto

que bons resultados foram fornecidos ao PGA pelo AGC, em um número limitado de iterações, esta heurística vem se somar às poucas heurísticas que realmente fornecem soluções boas ao PGA.

2 – Algoritmo Genético Construtivo - resumo

Nos trabalhos de Furtado [8] e Lorena e Furtado[12] tem-se uma descrição detalhada do AGC. Neste item, vamos resumir a proposta do AGC.

O AGC trabalha com uma população inicial formada apenas por estruturas ou esquemas. Neste trabalho, será denominada de *estrutura* qualquer cadeia contendo ou não o símbolo #, o qual pode representar 0 ou 1, e quando se referir a *esquema*, estará se fazendo referência explícita a uma estrutura que contém o símbolo #. Uma representação de uma estrutura, para um problema contendo 3 mochilas e 10 itens, seria (1,0,0,1,0,0,1,0,0,0). Um esquema seria a representação (#,0,1,0,1,0,#,#,1,0). Como um esquema não representa uma solução completa do problema, o AGC usa uma medida de adaptação proporcional do esquema à população. A cada nova iteração do processo, a população aumenta com a inclusão de novos indivíduos (estruturas) gerados com a combinação e mutação de estruturas da população atual. Os bons esquemas (ou estruturas) são preservados dentro dos novos esquemas (ou estruturas) gerados, que geralmente se aproximam de representações completas da solução do problema. Também a cada iteração do processo, um parâmetro evolutivo é usado na eliminação de indivíduos que não satisfazem um critério de permanência na população. Uma cópia da melhor solução encontrada até o momento é mantida salva, e o processo termina quando um limite de iterações previamente estipulado é atingido ou eventualmente a população fique vazia [16].

2.1 - Representação

Um esquema pode ser representado pela cadeia $s_k = (s_{k1},s_{k2},...,s_{kn})$, onde n é, por analogia, o número de itens a serem colocados na mochila. No exemplo do PGA, um esquema pode ser representado por uma cadeia que apresenta três tipos de informações, os quais, denotamos:

- 1 : representa um item atribuído a uma mochila;
- 0: representa um item não atribuído a mochila
- # : representa item que poderá futuramente ser um item atribuído a uma mochila

No início do algoritmo do AGC para o PGA, são gerados esquemas que comporão a população inicial. Estes esquemas serão do tipo (#, 0, ..,1, 0, 1, #,...). Cada esquema tem n posições, que representam os itens. Inicialmente, m itens são atribuídos. É importante mencionar que o tamanho da população inicial deve ser suficientemente grande, de forma a capturar todas informações do problema. Para exemplificar, se tivermos 3 mochilas com 10 itens a serem atribuídos, e se tivermos um esquema com representação inicial (#,1,#,0,1,0,0,#,1,0), isto significa que o item 2 é atribuído a mochila 1, o item 5 é atribuído a mochila 2, e o item 9, para a mochila 3. Os itens cobertos com 0 são itens não atribuídos e os itens com # são itens que podem ser atribuídos ou não.

2.2 – População inicial

Para a resolução do PGA usando-se o AGC, inicialmente é necessário estabelecer uma população inicial. Esta população inicial foi gerada aleatoriamente com 20% das colunas compondo os esquemas com símbolo 0, m itens compondo o esquema com valor 1, e o restante sendo cobertos por #.

2.3 – Seleção e recombinação

A população (tanto a inicial como nas demais iterações do AGC) é mantida ordenada de acordo com um critério que provilegia os esquemas mais próximos da solução completa e com melhor qualidade, tal como sugerido em Furtado [8].

O método de seleção para recombinação utiliza dois esquemas: um esquema chamado de base e o outro de guia. O esquema base é selecionado dentro dos 20% melhores esquemas e o esquema guia, dentros dos outros 80%, conforme a ordenação feita previamente. Se o esquema base for completo (é uma estrutura e todos os itens foram atribuídos), então é aplicado a este esquema (além da seleção e recombinação) um critério denominado de mutação. Neste critério, 1 ou mais itens podem ser trocados (alguma atribuição de item pode ser mudada de mochila).

Se base não for uma estrutura completa (se tiver algum # em alguma posição), a recombinação entre ambos (base e guia) é usada. É feita uma mescla da base com a guia, conforme um critério previamente determinado. Mais detalhes sobre este critério pode ser visto em Furtado e Lorena [12]. Desta mescla, é gerado um novo indivíduo na população. O número de 1 tem que ser mantido sempre constante e igual ao número de mochilas.

A Recombinação entre o esquema base e o esquema guia é descrita a seguir.

Sejam Sbase(j) e Sguia(j) a representação dos esquemas base e guia respectivamente para um item j. Tem-se a seguinte regra para um item j, quando da recombinação (geração do esquema Snovo(j)):

1 - *Sbase*(j) = # e *Sguia*(j)=# então *Snovo*(j) = #
2 - *Sbase*(j) = 1 e *Sguia*(j)=1 então *Snovo*(j) = 1
3 - *Sbase*(j) = 0 e *Sguia*(j)=0 então *Snovo*(j) = 0
4 - *Sbase*(j) = 1 e *Sguia*(j)=# então *Snovo*(j) = 1
5 - *Sbase*(j) = 0 e *Sguia*(j)=# então *Snovo*(j) = 0
6 - *Sbase*(j) = # e *Sguia*(j)=0 então *Snovo*(j) = 0
7 - *Sbase*(j) = # ou 0 e *Sguia*(j)=1 então
Snovo(j) =1 e *Snovo*(i) =0 para algum *Snovo*(i)=1, i ∈ {1,2,...,n} i diferente de j
8 - *Sbase*(j) = 1 e *Sguia*(j)=0 então
Snovo(j) = 0 e *Snovo*(i) = 1 para algum *Snovo*(i)=0, i ∈ {1,2,...,n} i diferente de j

Através destas regras um novo esquema é gerado (recombinação).

2.4 – Funções de avaliação f e g

Definida uma população de esquemas iniciais, é realizada a sua avaliação mapeando o espaço das estruturas em R_+. Considera-se P_a como a população de estruturas no instante de evolução a (que será descrito posteriormente) e considerando duas funções f e g, definidas como: $f{:}P_a \to R_+$ e $g{:}P_a \to R_+$ e calculadas tais que $f(s_k) \leq g(s_k)$, para toda $s_k \in P_a$. Também definimos um limite superior comum, $g_{\max} > \underset{s_k \in P_a}{Max}\, g(s_k)$. A esta dupla forma de avaliar s_k denominaremos *avaliação-fg*. Supondo uma população P_a de m estruturas, a figura 1 mostra uma possível avaliação-*fg* de cada estrutura em P_a e o limite superior g_{max}.

Para cada problema de otimização é necessário definir as funções f, g e o limite g_{max}. Para o Problema Generalizado de Atribuição (PGA), cuja função objetivo é a de minimização, a função f é um limitante inferior e g, um limitante superior. Além disso, g fornece uma solução viável ao PGA (par as posições no esquema que tenham 1 e 0) e f funciona como se fosse uma relaxação do PGA (para as posições que tenham 1 e 0 no esquema). Estas funções serão explicadas a seguir.

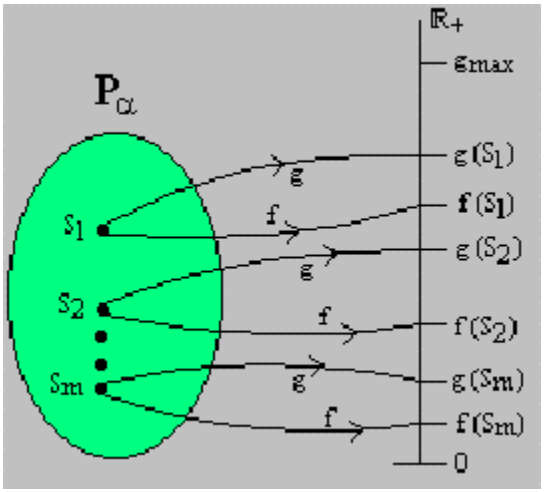

Figura 1. Possível avaliação de uma população.

Para assegurar que g_{max} seja sempre um limite superior, após a recombinação, cada nova estrutura gerada, s_{nova}, é rejeitada se $g_{max} \pounds g_{nova}$.

Para o PGA, a função g é uma heurística que visa a dar uma solução viável associada a estrutura. Por exemplo, suponha que se tenha a estrutura (#,0,0,1,0,#,1,0,#,1) para o caso de 3 mochilas e 10 itens. Considera-se os itens 4, 7 e 10 atribuídos as mochilas 1,2 e 3 respectivamente. Os demais itens iguais a 0 e # são preenchidos pela heurística, a qual dá um valor a função objetivo do problema. Seja g' o valor resultante. O valor de g é dado por g' menos os custos (pij) dos itens que são preenchidos por # no esquema. Desta forma, o esquema (#,0,0,1,0,#,1,0,#,1) tem um valor associado. A função f é uma outra heurística que visa a dar um limite inferior ao problema. Tal como a função g, Considera-se os itens 4, 7 e 10 atribuídos as mochilas 1,2 e 3 respectivamente. Os demais itens iguais a # e 0 são preenchidos pela heurística, a qual dá um valor a função objetivo do problema de forma diferente da função g, isto é, tenta atribuir valores de pij menores do que os atribuídos quando da função g. É permitida a não verificação de uma das restrições (e somente uma) e mesmo assim a violação desta restrição deve ser a mínima possível. No caso, a restrição violada é uma das restrições de capacidade. Seja f'' o valor obtido. O valor de f será igual ao de f'' menos o valor dos custos (pij) nos quais tem-se a posição j igual a # no esquema. Desta forma, garante-se que f será menor do que g e com um valor próximo ao de g.

O valor de gmax é calculado da seguinte forma: seja a população inicial. Suponha que se tenham k estruturas. Usa-se a função para cada uma destas k estruturas. Entretanto, considera-se também o #, isto é, os itens iguais a 0 e # também são atribuídos. Desta forma, ter-se-á k valores de g. o Valor de gmax é o maior destes k valores.

Vale a pena mencionar que os resultados são influenciados pela qualidade das heurísticas para g e f, isto é, quanto melhor forem as heurísticas, melhor serão os

resultados. Entretanto, as heurísticas não podem ser de execução demorada (lenta), pois desta forma, fica inviável usar o algoritmo genético, pelo tempo que demoraria para executá-lo.

Mais informações sobre a funções de avaliação f e g podem ser vistas em http://www.lac.inpe.br/~lorena/teseJC/cga-tese.pdf.

2.5 - Mutação

Através das sucessivas recombinações, obtém-se estruturas completas que representam soluções para o problema de otimização. Estas estruturas, normalmente representam soluções de boa qualidade. No entanto, é possível melhorar esta qualidade realizando uma mutação na estrutura, como uma forma de busca local.

Assim, analisando criteriosamente o problema de otimização, podemos implementar uma forma de mutação adequada para estruturas completas.

A idéia da mutação é tomar uma estrutura completa (sem item igual a #) e escolher um item igual a 1 e torná-lo igual a zero. Em seguida, tornar um item zero (desde que não seja o item que passou de 1 para zero) e torná-lo igual a 1. Esta nova estrutura também é avaliada pelas funções f e g e demais critérios. Se a nova estrutura não for aceita, então procede-se uma outra mutação. Para o caso do PGA, o limite máximo era 10 mutações, pois caso contrário, o tempo de execução já começava a ser comprometido.

Mais informações sobre a mutação podem ser vistas em http://www.lac.inpe.br/~lorena/teseJC/cga-tese.pdf.

2.6 – Avaliação da População

A avaliação-*fg* fornece limites para cada estrutura que serão, agora, comparados ao limite superior, g_{max}. Para cada estrutura s_k, existe um desvio associado (desvio percentual de $g(s_k)$) dado por:

$$d_k = (g(s_k) - f(s_k)])/g(s_k) \quad k = 1,...,m$$

Seja d um desvio máximo pré-estabelecido. Em http://www.lac.inpe.br/~lorena/teseJC/cga-tese.pdf. há um critério para verificar se uma dada estrutura deve ser ou não rejeitada. Este critério é dado pela expressão:

$d_k\ g(s_k) + \alpha.d.[g_{max} - g(s_k)] \geq d\ g_{max} \quad \alpha \geq 0$

Através desta fórmula, podemos verificar se uma estrutura deve ou não ser rejeitada (mais detalhes, ver http://www.lac.inpe.br/~lorena/teseJC/cga-tese.pdf). Se, ao substituirmos os valores, a desigualdade for satifeita, então a estrutura s_k é rejeitada.

Esta expressão é equivalente a $d_k \geq d[(1- \alpha)(g_{max} / g(s_k)) + \alpha]$

Quando a =1, existe a comparação direta d_k ³ d, e para a =0, a expressão torna-se: $d_k \geq d(g_{max} / g(s_k))$

Considerando que $g_{max}/g(s_k)$ ³ 1, no intervalo $0 \leq a < 1$ as estruturas são em geral conservadas, a menos daquelas que apresentem $d_k >> d$. Quando $a > 1$, uma estrutura s_k mesmo tendo d_k £ d, poderá ser rejeitada. É importante observar que $g_{max}/g(s_k) >> 1$ quando $g(s_k)$ é pequeno comparado a g_{max}, o que ocorre, em geral, nos esquemas. Considerando que os (bons) esquemas precisam ser preservados para (re)combinação, o parâmetro de evolução a inicia com 0 e é gradativamente incrementado, em pequenos intervalos, de geração a geração. P_0 é a população inicial para a=0. Nós denotamos a população no instante de evolução a como P_a.

O parâmetro a pode ser isolado, produzindo a expressão:

$\alpha \geq (dg_{max} - d_k\ g(s_k))/(d.[g_{max} - g(s_k)]) = \delta(s_k)$

No instante da criação, cada estrutura recebe um valor correspondente $d(s_k)$ que será comparado com o parâmetro de evolução corrente a. Desta forma, no momento da criação da estrutura, pode-se prever o tempo de sua sobrevivência, sendo que quanto maior $d(s_k)$, mais tempo para a estrutura s_k sobreviver e recombinar-se. Ainda, analisando a razão $d(s_k)$, torna-se claro que esquemas *mais completos* ($g(s_k)$ próximo de g_{max}) e/ou apresentando d_k pequeno, terão mais tempo para se recombinar.

2.7 – O algoritmo

Os passos da forma básica do algoritmo genético construtivo pode ser resumidos, conforme algoritmo a seguir.

AGC() {Algoritmo Genético Construtivo}

α:=0;

ε = 0.05; {intervalo de tempo}

Inicializar P_a; {população inicial}

Avaliar P_a;

Para todo $s_k \in P_\alpha$ **calcule** $\delta(s_k)$ {cálculo do rank}

Fim_Para

Enquanto não (condição de parada) **fazer**

Para todo $s_k \in P_\alpha$ satisfazendo $\alpha < \delta(s_k)$ **fazer** {teste de evolução}

$a := a + e$;

Selecionar P_{a+e} de P_a; {operador reprodução}

Recombinar P_a;

Avaliar P_a; {calcula adaptação proporcional}

Fim_Para

Para todo novo $s_k \in P_\alpha$ **calcule** $\delta(s_k)$ {cálculo do rank}

Fim_Para

Fim_enquanto

Alguns passos são notavelmente diferentes do AGT. O AGC trabalha com uma população dinâmica, que aumenta após o uso dos operadores recombinação e pode diminuir com a aumento do parâmetro de evolução a. O parâmetro do programa ε, incrementa o valor de α. Após a sua criação, cada estrutura recebe uma avaliação proporcional e um valor $d(s_k)$ (qualidade relativa da estrutura s_k em relação as outras estruturas) usado no teste de evolução. Outra diferença fundamental está na recombinação e avaliação direta de esquemas (ver http://www.lac.inpe.br/~lorena/teseJC/cga-tese.pdf).

O algoritmo pára o processo evolutivo e consequentemente a busca, quando algum dos seguintes critérios é satisfeito:

1. A solução ótima do problema é obtida (quando esta é conhecida);
2. A população torna-se vazia, pois todas as estruturas são rejeitadas (obtido através de um parâmetro α suficientemente grande);
3. Um número pré-estabelecido de iterações é atingido.

Nos dois últimos casos a melhor solução obtida no processo deve ser preservada.

3 – Algoritmo genético construtivo aplicado ao PGA - resultados

O desempenho do algoritmo genético será mostrado para um conjunto de instâncias para o *(PGA)*, todas obtidas da OR-Library [8]. Vale a pena mencionar as instâncias são considerados de larga escala. Foram utilizados 24 tipos de problemas com dimensões diferentes. Estes problemas são conhecidos como problemas de classes A, B, C, e D. As dimensões de cada uma das instâncias destes problemas são: (5 x 100), (5 x 200), (10 x 100), (10 x 200), (20 x 100) e (20 x 200). Estes testes foram executados em uma máquina SUN ULTRA SERVER 2, 200 MHz com compilador C. Os critérios de parada foram:

1 – número máximo de iterações igual a 150

2 - A solução ótima do problema é obtida (quando esta é conhecida);

3 - A população torna-se vazia, pois todas as estruturas são rejeitadas (obtido através de um parâmetro α suficientemente grande);

Instância	Melhor solução conhecida	Solução obtida pelo ACG	Número de iterações do algoritmo	Tempo gasto para a execução (s)
A 5x100	1698	1698	51	253
A 5x200	3235	3235	1	502
A 10x100	1360	1360	87	308
A 10x200	2623	2623	72	930
A 20x100	1158	1158	1	350
A 20x200	2339	2339	19	860
B 5x100	1843	1843	89	168
B 5x200	3553	3601	150	432
B 10x100	1407	1410	150	165
B 10x200	2831	2831	95	659
B 20x100	1166	1166	102	314
B 20x200	2340	2347	150	683
C 5x100	1931	1941	150	195
C 5x200	3458	3460	150	405
C 10x100	1403	1423	150	203
C 10x200	2814	2815	150	498
C 20x100	1244	1244	105	309
C 20x200	2397	2397	132	859
D 5x100	6373	6479	150	259
D 5x200	12823	12823	109	853
D 10x100	6379	6390	150	497
D 10x200	12601	12634	150	1321
D 20x100	6269	6280	150	974
D 20x200	12452	12471	150	2158

Para os resultados acima, foram usados os seguintes valores para as variáveis d e α:
d = 0.15.
α inicial é igual a 0.
α = é incrementado de 0.1 em cada iteração até que atinja o valor 1;
α = é incrementado de 0.01 em cada iteração a partir de quando α for maior do que 1.

Na tabela acima, a melhor solução conhecida foi obtida do trabalho de Beasley [3] e o número de iterações é o número de iterações da condição "**Enquanto não** (condição de parada) **fazer"** do algoritmo descrito no item 2.7 deste trabalho

Tal como foi feito por Beasley[3], para cada instância foram tomadas 10 amostras de resultados para uma mesma instância. O melhor resultado (dentre as 10 amostras) para cada instância é o que está na tabela.

Sobre a tabela acima, pode-se afirmar que os resultados foram muito próximos aos conseguidos por Beasley[3]. Um outro ponto importante a se ressaltar é que não foram colocados os tempos obtidos por Beasley [3] devido ao fato dele colocar como unidade de tempo "segundos de CPU" e também pelo fato da máquina que ele usou ter arquitetura diferente (Silicon Graphics Indigo, RS4000, 100 MHz).

Beasley tinha o seguinte critério de parada: cada teste com uma dada instância terminava quando o número de esquemas (ou estruturas) fosse igual a 500.000 sem que a melhor solução obtida não melhorasse e os esquemas fossem todos distintos (não duplicados). Para os testes feitos com o PGA usando o AGC, o limite máximo era 25.000. Desta forma, o esforço computacional foi bem menor (versão AGC/PGA) e os resultados se aproximaram muito dos que Beasley obteve (melhor solução obtida).

4 – conclusões e comentários

Quando o algoritmo genético foi proposto por Holland [9] em 1975, ele já acreditava na idéia de que boas soluções para um problema, são resultado da combinação de *bons* blocos construtivos, o que deu origem a noção de esquema. No AGC, esta idéia foi adotada e foi criada uma forma diferente de avaliar diretamente esquemas, o que ainda não tinha sido realizado com sucesso. Para o PGA, esta nova proposta mostrou ser interessante, tal como mostram os resultados obtidos.

A agregação sucessiva de informações ao problema, ou seja, a recombinação, onde diferentes esquemas são unidos para gerar novos esquemas ou estruturas completas, conseguiu gerar estruturas de alta qualidade, o que permitiu que através de um pequeno esforço computacional adicional, representado pela mutação, as soluções ótimas ou aproximadas fossem obtidas.

Os resultados obtidos com o AGC para o PGA mostraram que o novo método é eficiente na obtenção de boas soluções. Na maioria das instâncias analisadas, as soluções ótimas ou soluções com valores estando a menos de 2% das soluções ótimas foram obtidas.

As funções f e g influenciaram nos resultados obtidos pelo AGC para o PGA. Pesquisas sobre heurísticas para f e g devem ser feitas para que o tempo de execução seja menor e a qualidade das soluções obtidas sejam tão boas ou melhores do que as obtidas por este trabalho.

5 – Referências bibliográficas

[1] Balachandran, V. An integer generalized transportation model for optimal job assignment in computers networks. **Operations Research**, v. 24, n.4, p. 742-749, 1976.

[2] Beasley, J. E. OR-Library: Distributing test problems by eletronic mail. **Journal of Operational Research Society**, v. 41,n. 11, p. 1069-1072, 1990.

[3] Beasley, J.E. e Chu, P. C. A genetic algorithm for the generalised assignment problem. The Management School. Imperial College, London, 1995.

[4] Catrysse, D. Van Wassenhove, L. N. A survey of algorithms for the Generalized Assignment Problem. **European Journal of Operational Research**. v. 60, p. 260-272, 1992.

[5] De Maio, A., Roveda, C. An all zero-one algorithm for a certain class of transportation problems. **Operations Research**. v. 19, p. 1406-1418, 1971.

[6] Fisher, M. L., Jaikumar, R. A generalized assignment heuristic for vehicle routing. **Networks**. v. 11, p. 109-124, 1981.

[7] Fisher, M. L., Jaikumar, R., Wassenhove, L. N. V. A multiplier adjustment method for the generalized assignmet problem. **Management Science**. v. 32, p. 1095-1103, 1986.

[8] Furtado, J. C. "Algoritmo Genético Construtivo na Otimização de Problemas Combinatoriais de Agrupamentos". Tese de Doutorado em Computação Aplicada no INPE, 1998

[9] Holland, J.H. Adaptation in natural and artificial systems. **MIT Press**, p. 11-147, 1975.

[10] Jornsten, K.; Varbrand, P. Relaxation techniques and valid inequalities applied to the generalized assignment problem. **Asia Pacific Journal of Operational Research**. v. 7, p. 172-189, 1990.

[11] Klastorin, T. D. An effective subgradiente algorithm for the Generlized Assignment Problem. **Computers and Operations Research**. v. 6, p. 155-164, 1979.

[12] Lorena, L. A. N. and Furtado, J. C. Constructive genetic algorithm for clustering problems. Apresentado no Optimization 98- Coimbra, Portugal - 20-22 julho de 1998.

[13] Lorena, L.A.N. and Lopes, L.S., Genetic Algorithms Applied to Computationally Difficult Set Covering Problems. Journal of the Operational Research Society 48, 440-445, 1997.

[14] Lorena, L. A. N., Narciso, M. G. Relaxation heuristics for a generalized assignment problem. **European Journal of Operational Research**. v. 91, n. 1, p. 600-610, 1996.

[15] Martello, S Toth, P. Knapsack Problems - Algorithms and Computer Implementations. New York: John Wiley & Sons, USA, 1990.

[16] Ribeiro Filho, G.; Lorena, L.A.N. A constructive algorithm for cellular manufacturing desing. EURO XVI - 16th European Conference on Operational Research, 12 a 15/07/98, Bruxelas, Bélgica.

[17] Ross, G. T., Soland, M. S. A branch and bound algorithm for the generalized assignment problem. **Mathematical Programming**, v. 8, p. 91-103, 1975.