MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

ALGORITMO GENÉTICO CONSTRUTIVO NA OTIMIZAÇÃO DE PROBLEMAS COMBINATORIAIS DE AGRUPAMENTOS

João Carlos Furtado

Tese de Doutorado em Computação Aplicada, orientada pelo Dr. Luiz Antônio Nogueira Lorena.

São José dos Campos
Março de 1998

AGRADECIMENTOS

Agradeço ao meu orientador Dr. Luiz Antonio Nogueira Lorena, pelo incentivo e apoio durante todas as fases de desenvolvimento desta tese.

Agradeço aos meus pais, Benno e Lilly, e aos meus irmãos Nilo, Marli, Miria e Rosane.

Agradecimento especial à Lúcia Beatriz, pelo amor e compreensão.

Agradeço também a Universidade Federal de Santa Maria, pela oportunidade de realizar este curso.

Agradeço ainda os colegas da Computação Aplicada e todos os meus amigos.

RESUMO

Nesta tese introduzimos um novo método heurístico denominado Algoritmo Genético Construtivo – AGC. Esta heurística apresenta uma população de esquemas (blocos construtivos) que carregam informações sobre as propriedades estruturais do problema e são avaliados através de funções que determinam o quanto promissor é cada um destes esquemas. Os esquemas são avaliados diretamente através de funções apropriadas, sendo que os melhores são incentivados a se recombinar com outros para gerar novos esquemas ou estruturas completas. O objetivo é produzir, através de sucessivas gerações, novos esquemas ou estruturas completas que além de agregar mais informações sobre o problema, apresentem bom desempenho nas funções de avaliação. Os esquemas ou estruturas que não obtém boa avaliação são eliminados da população através de um crítério de poda. No final do processo, estruturas de qualidade superior são obtidas. Este trabalho também mostra a aplicação do AGC em três diferentes problemas combinatoriais: problema das *p*-medianas, problema de agrupamento capacitado e problema capacitado de particionamento de grafos. Os resultados para estes problemas são mostrados e comparados com outras heurísticas. Por fim, conclusões e sugestões são apresentadas.

ABSTRACT

We present in this thesis a new heuristic called Constructive Genetic Algorithm – CGA. This heuristic presents a population formed of schemas (building blocks) that carries information about structural proprieties of the problem. The CGA use functions that evaluate schemas. The schemas with best fitness are stimulated to recombine with other for generate new schemas or complete structures. The goal is to get schemas or structures with high quality. The population is dynamic, increasing and decreasing of convenient evolution times. The schema evalution is updated and used as a criterion for schema permanency. At the time of the schema creation, it receives a rank that indicates how long it will survive. The better schema found so far is kept. The process finishes with an emptied population or when an iteration limit is reached. This work, also, present the CGA applied to three diferents optimizations problems: The *p*-median problem; the capacitated clustering problem and min-cut problem. Results are shown for instances from the literature using a microcomputer implementation and significant conclusions are described.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE SÍMBOLOS

- s estrutura;
- S conjunto de todas estruturas;
- b comprimento de uma estrutura;
- A alfabeto;
- $|A|$ número de símbolos em A;
- H esquema;
- $M(H)$ número de representantes de H na população;
- $o(H)$ ordem de H;
- $\psi(H)$ comprimento de H;
- η função adaptação de uma estrutura;
- P população;
- P_o população inicial;
- $|P_o|$ tamanho da população inicial;
- P_α população de estruturas no instante de evolução α;
- $T(s,t)$ taxa de amostragem da estrutura *s* na geração t;
- p_c taxa de *crossover*;
- p_m taxa de mutação;
- $\hat{u}(H,t)$ adaptação média observada de H no instante t;
- $E(M(H,t))$ número esperado de representantes de H no instante t;
- $\rho_c(H)$ probabilidade de H sobreviver a um *crossover*;
- c função de avaliação do algoritmo A*;
- h função de avaliação do algoritmo A*;
- p número de medianas;
- f função de avaliação do AGC;
- g função de avaliação do AGC;
- g_{max} função de avaliação do AGC;
- d_k desvio percentual da estrutura k;
- d desvio percentual global admitido;
- α parâmetro de evolução;
- $\delta(s_k)$ "ranking" de s_k;
- ε parâmetro do AGC;
- V conjunto de vértices de um grafo;
- μ custo (distância);
- Δ chave de ordenação de estruturas;
- C_j agrupamento com mediana em j;
- λ_j custo (distância) mínimo de atribuição no agrupamento j;

- Q_i capacidade do agrupamento i;
- ζ_i centro do agrupamento C_i;
- Γ conjunto dos agrupamentos;
- a_k demanda do cliente (vértice) k;

1 – INTRODUÇÃO

Na otimização combinatória estudamos problemas que se caracterizam pelo número finito de soluções possíveis; e embora, em princípio, a solução ótima possa ser obtida através de uma simples enumeração, na prática, freqüentemente isto torna-se inviável, devido ao número extremamente alto de soluções viáveis. Assim, estudando as propriedades estruturais do problema, métodos heurísticos têm sido apresentados pela comunidade científica para obter soluções exatas ou aproximadas.

Embora ainda existam algumas críticas aos métodos heurísticos, a característica de obterem boas soluções (e em muitos casos soluções ótimas) em intervalos de tempos reduzidos e compatíveis com as exigências de situações reais, têm contribuído para o grande interesse pelo assunto, sendo que para diferentes problemas combinatoriais, os melhores resultados da literatura foram obtidos através destes métodos.

Os intervalos de tempo e memória exigidos pelas heurísticas podem ser medidas de diferentes maneiras. Uma grande extensão da literatura está dedicada a estudar a complexidade do pior caso, isto é, obter um limite superior para o tempo e espaço (memória) necessários para obter uma solução para um problema. Mesmo sendo uma medida importante, freqüentemente, existe uma grande diferença entre o tempo e espaço do pior caso e a sua média. Outra medida da eficiência de uma heurística baseia-se na qualidade de suas soluções, que pode ser obtida através da comparação com algum

limite (inferior ou superior) ou outro método (heurístico ou exato). Uma significativa discussão sobre o tema é dada por Roberts (1990).

Algumas heurísticas, denominadas meta-heurísticas, podem ser usadas na resolução de diversos problemas de otimização combinatória, através de uma representação adequada e a adaptação de alguns parâmetros para cada problema específico. Podemos citar, entre outras, meta-heurísticas de sucesso, como o *algoritmo genético* (Holland,1975 e Goldberg, 1989), a *busca tabu* (Glover,1989, 1990) e "*simulated annealing*" (Kirkpatrick *et al.*,1983 e Cerny, 1985). O algoritmo genético é um dos representantes dos algoritmos evolutivos, que têm inspiração baseada na evolução natural dos seres vivos.

Os algoritmos evolutivos são baseados num processo coletivo de aprendizagem dentro de uma população de indivíduos (estruturas), cada um dos quais, representando um ponto no espaço de busca de soluções, para um dado problema. A população é aleatoriamente inicializada e evolui no espaço de busca através dos operadores seleção, recombinação e mutação. Durante o procedimento, informações da qualidade (valor da adaptação) dos pontos de busca são obtidos, e são usados para direcionar a busca, que favorece a escolha (no processo de seleção) de indivíduos mais adaptados, para que estes, gerem novos indivíduos. O mecanismo de recombinação permite misturar informações de uma geração e passá-las aos seus descendentes, e a mutação introduz inovação na população.

A teoria tradicional do algoritmo genético assume que ele funciona descobrindo, enfatizando e recombinando bons *blocos construtivos* nas soluções. Acredita-se que boas soluções são obtidas através da agregação de bons blocos construtivos, idéia que foi formalizada através da introdução da definição de *esquema* (Holland, 1975).

Nesta tese introduzimos um novo método heurístico, denominado Algoritmo Genético Construtivo-AGC. A idéia inicial surgiu com o artigo "A Dynamic List Heuristic for 2D-Cutting", desenvolvido por Lorena e Lopes (1996) para resolver um problema de cortes de estoques. A heurística também foi inspirada no algoritmo A*,

(Pearl,1985) conhecido em Inteligência Artificial, onde é usado para direcionar procedimentos de busca. No decorrer do desenvolvimento da tese, verificamos algumas semelhanças entre o novo método e o Algoritmo Genético Tradicional-AGT, ocasião em que resolvemos adotar a atual denominação, definições e nomenclatura.

O AGC inicia com uma população de esquemas (blocos construtivos). Os esquemas carregam informações sobre propriedades estruturais do problema e são avaliados através de funções que determinam o quanto promissor é cada um destes esquemas. Os melhores esquemas são incentivados a recombinarem-se com outros, de tal forma que através de sucessivas gerações, novos esquemas ou estruturas completas são produzidas que, além de agregar mais informações sobre o problema, apresentem bom desempenho nas funções de avaliação. Os esquemas ou estruturas que não obtiverem boa avaliação serão eliminados da população através de um critério de poda. No final do processo, esperamos que estruturas de alta qualidade sejam obtidas, pois acreditamos que agregando sucessivamente informações sobre o problema, possa-se obter soluções (estruturas) de melhor qualidade para o problema de otimização.

No Algoritmo Genético Construtivo os esquemas são avaliados diretamente através das funções de avaliação, o que permite a junção de diferentes *bons* esquemas, o que por sua vez, pode dar origem a novos esquemas ou estruturas de alta qualidade. A avaliação direta dos esquemas no AGC representa uma das diferenças em relação ao Algoritmo Genético Tradicional, onde os esquemas são avaliados através das instâncias que produzem.

A tese esta organizada da seguinte forma. No segundo capítulo, iremos apresentar uma revisão sobre os Algoritmos Evolutivos, onde inicialmente a evolução natural será descrita. As Estratégias de Evolução e a Programação Evolutiva serão brevemente comentados, e será dada especial atenção ao Algoritmo Genético, bem como os avanços em relação a sua forma básica serão mostrados. Veremos também como o Teorema do Esquema descreve o crescimento de uma população.

No terceiro capítulo apresentamos o algoritmo genético construtivo. No início, o algoritmo A* é mostrado, pois algumas de suas idéias são aproveitadas no

AGC. Em seguida um breve histórico é traçado. Também, mostramos como podemos representar e avaliar esquemas (ou estruturas completas) e como pode ser a seleção e recombinação no AGC. O critério de eliminação de estruturas (esquemas) é explicado. Por fim, um pseudo-algoritmo é apresentado.

No capítulo seguinte, o método é usado para obter soluções no problema das *p*-medianas, que é reconhecidamente um problema difícil, isto é, *NP-hard* (Garey e Johnson,1979). A busca de *p*-medianas num grafo é um problema clássico de localização. O objetivo é localizar *p* facilidades (medianas), de forma a minimizar a soma das distâncias de cada vértice a sua facilidade mais próxima.

Usamos também o AGC para resolver o problema de agrupamento capacitado ("capacitated clustering problem"–CCP). O CCP é o problema no qual, dado um conjunto de objetos com diferentes pesos, deseja-se particionar este conjunto em diferentes agrupamentos, de tal forma que o peso total dos objetos em cada agrupamento seja menor ou igual a um dado valor. O objetivo é minimizar a dispersão total dos objetos em relação ao centro do agrupamento ao qual foram atribuídos.

Além disso, usamos o AGC em um problema capacitado de particionamento de grafos. O problema consiste em particionar os vértices de um grafo em *k* agrupamentos, tal que a soma dos pesos dos vértices de cada agrupamento tenha um limite inferior e superior, enquanto maximiza (minimiza) a soma dos custos das arestas dentro (fora) de cada agrupamento.

No final, os resultados são analisados e comparados aos obtidos por outros métodos de otimização, conclusões e sugestões apresentados.

2 – ALGORITMOS EVOLUTIVOS

Neste capítulo vamos mostrar como as idéias da evolução natural foram aproveitadas para obter soluções de alta qualidade em problemas matemáticos difíceis, especialmente em problemas de otimização. Será dada uma descrição da evolução natural, processo no qual os *Algoritmos Evolutivos* estão baseados. As *Estratégias de Evolução* e a *Programação Evolutiva* serão mostradas. O *Algoritmo Genético* será apresentado na sua forma básica e algumas das suas principais variações e avanços serão descritos. Além disso, os principais grupos de pesquisa atuando na área são destacados.

2.1 – EVOLUÇÃO NATURAL

Em 1859, Charles Darwin (Mitchell, 1996) apresentou a *seleção natural*, principio segundo o qual *"os indivíduos mais adaptados ao meio, apresentam maior possibilidade de sobrevivência"*. Este princípio é resultado da observação de que as mais diferentes formas de vida são suscetíveis a adaptação, que ocorre através de lentas transformações genéticas, de geração em geração. A adaptação de um indivíduo está relacionada ao ambiente em que vive, sendo portanto uma medida relativa e não absoluta. Assim, podemos dizer que um pingüim está adaptado ao frio da Antártica, mas não estaria ao calor da região tropical.

O processo de evolução ocorre através de ciclos de gerações fixas. Cada indivíduo nasce, cresce, normalmente gera um ou mais filhos e morre. Os filhos recebem atributos genéticos dos pais que definem a aparência de um indivíduo, o que é denominado *fenótipo.* As informações genéticas estão codificadas num grande conjunto de *genes*, que são as unidades de transferência da hereditariedade. Cada gene pode assumir um valor particular, o *alelo*. Os genes formam os *cromossomos*, que por sua vez formam as cadeias de DNA (ácido desoxirribonucléico). Um conjunto de cromossomos em um organismo que define sua informação genética completa é denominada *genótipo.*

2.2 – COMPUTAÇÃO EVOLUTIVA

Nas décadas de 1950 e 1960, diversos grupos de pesquisadores estudaram, independentemente, sistemas evolutivos, acreditando que o princípio da evolução deveria ser aproveitado como ferramenta de otimização em problemas de engenharia. A idéia nestes sistemas, era criar uma população de soluções candidatas para um dado problema, usando operadores inspirados na evolução e seleção natural.

Quatro décadas de pesquisas e aplicações dos, atualmente denominados *Algoritmos Evolutivos – AE*, mostraram que o procedimento de imitar a evolução natural dos organismos vivos, embora muitas vezes com grandes simplificações, produzem algoritmos capazes de obter boas soluções em problemas de otimização e contribuem com outras áreas do conhecimento, tais como: economia, imunologia, sistemas sociais, inteligência artificial e vida artificial, entre outros.

Conforme visto na introdução, a população de indivíduos (também denominadas estruturas) sofre um processo coletivo de aprendizagem, com cada um dos indivíduos, representando um ponto no espaço de busca de soluções, para um dado problema. A população é inicializada aleatoriamente ou baseada num critério que capture informações sobre a estrutura do problema, e evolui no espaço de busca através do procedimento de seleção, mutação e recombinação (omitida na Programação Evolutiva). Durante a busca, informações da qualidade (valor da adaptação) dos pontos de busca são obtidos, e que são usados para direcionar a busca, que favorece a escolha (no processo de seleção) de estruturas mais adaptadas, para que estas, gerem novas

estruturas. O mecanismo de recombinação permite misturar informações de uma geração e passá-las aos seus descendentes, e a mutação introduz inovação na população.

2.3 – ESTRATÉGIA DE EVOLUÇÃO E PROGRAMAÇÃO EVOLUTIVA

Na Alemanha, Rechengerg (1965) e Schwefel (1965) introduziram a *Estratégia de Evolução*–EE, que foi inicialmente usada para otimização de parâmetros contínuos. As primeiras tentativas de imitar princípios da evolução natural num computador ainda assemelhavam-se com métodos iterativos de otimização conhecidos na época. No chamado (1+1)-ES ("Evolutions-Strategie"), um mecanismo simples de mutação-seleção operava sobre um indivíduo (estrutura), dando origem a um indivíduo filho, por geração, através de uma mutação gaussiana. No procedimento denominado (μ+1)-ES, onde μ representa o tamanho da população de indivíduos pais, $\mu \geq 1$ indivíduos recombinam-se para formar um novo indivíduo, que após mutado, eventualmente substitui o pior indivíduo pai, de forma similar ao método simples. Embora este método nunca tenha sido muito usado, foi importante para a transição aos chamados métodos (μ+λ)-ES e (μ,λ)-ES, onde λ representa o tamanho da população de indivíduos filhos, e "+" ou "," definem o processo de seleção. Em (μ,λ)-ES a população da próxima geração é composta de λ filhos, o que significa que os indivíduos não sobrevivem mais de uma geração, enquanto que em (μ+λ)-ES, também indivíduos pais podem ser selecionados para a próxima geração. Atualmente as ES incluem recombinação de esquemas de múltiplos pais e auto-adaptação através de parâmetros estratégicos no código dos indivíduos. Maiores detalhes podem ser obtidos em (Bäck e Schwefel, 1993).

O livro "Artificial Intelligence Through Simulated Evolution" (Fogel *et al.*,1966) marca o início da *Programação Evolutiva* - PE. A PE é uma estratégia estocástica de otimização semelhante a EE e aos Algoritmos Genéticos, mas ao invés de enfatizar o relacionamento dos indivíduos pais com os indivíduos filhos, busca simular operadores genéticos específicos, de acordo com os observados na natureza. Mais detalhes podem ser vistos em (Fogel, 1991).

2.4 – ALGORÍTMO GENÉTICO BÁSICO (FORMA CANÔNICA)

O algoritmo genético foi concebido por John Holland em 1960 e aperfeiçoado por Holland, seus estudantes e colegas da Universidade de Michigan nas décadas de 1960 e 1970. Ao contrário da Programação Evolutiva e Estratégias de Evolução, o objetivo original de Holland não foi projetar algoritmos para problemas específicos, mas estudar como o fenômeno da adaptação ocorre na natureza e como este mecânismo poderia ser introduzido nos sistemas computacionais. No livro de 1975, "Adaptation in Natural and Artificial Systems", Holland apresenta o algortimo genético como uma abstração da evolução biológica (Holland, 1975). Refinamentos do método surgiram com De Jong (1975), Grefenstette (1986, 1987) e Goldberg (1989).

No Algoritmo Genético Tradicional-AGT, uma estrutura *s* é codificada como uma cadeia ("string") de comprimento *b* (*b*>0) sobre um alfabeto *A*. O espaço das estruturas é definido como o conjunto de estruturas pertencentes a A^b. Então, o tamanho do espaço das estruturas é $|A|^b$, onde |*A*| é o número de símbolos em *A*. Uma escolha popular para *A* é o conjunto binário {0,1}. Um elemento crucial no estudo do AGT é o conceito de esquema introduzido por Holland, em 1975. Um esquema é uma cadeia de comprimento *b* definida sobre o conjunto alfabeto *A*U{#}. O símbolo # pode ser substituído por qualquer símbolo de *A*. Uma vez que um esquema corresponda a um plano, no hiperplano definido pelo produto cartesiano de *A*, o esquema também é chamado de hiperplano na literatura sobre AGT. Por exemplo, dada uma estrutura *s* = (0 1 1 0 1 1) e um esquema *H* = (0 # 1 0 1 1), nós dizemos que *s* é um representante de *H* ou *instância* de *H*, porque *s* pode ser derivada de *H* substituindo o símbolo # em *H* por 1. O número de representantes de *H* é denotado por *M(H)*. No exemplo anterior, *M*(0#1011) é dois. As outras duas importantes propriedades de um esquema *H* são:

1. Ordem de *H*, denotada por *o(H):* o número de símbolos fixos em *H*. Por exemplo, *o*(0#1011)=5;

2. Comprimento de *H*, denotado por $\psi(H)$*:* a diferença entre o primeiro e último símbolo fixo em *H*. Por exemplo, ψ(#1#01##)=5-2=3.

A adaptação de uma estrutura é medida pela função η definida como $\eta:S\rightarrow R^+$, onde S é o conjunto de todas as estruturas e R^+ é o conjunto de números reais não negativos. Se a função objetivo do problema de otimização for sempre positiva, então poderá ser usada diretamente como η. Caso contrário, η deverá ser sua transformação. A forma atual de transformação depende de alguns fatos:

1. Se a função objetivo é de minimização ou maximização;
2. Do mecanismo de seleção usado; e
3. Da função de escala usada.

Detalhes podem ser obtidos em (Goldberg, 1989).

Quando aplicado para resolver problemas de otimização de uma função, o AGT opera da seguinte maneira: Uma população de soluções representadas por cadeias de tamanho fixo são mantidas durante a busca. Em cada iteração, uma nova população P_{t+1} é criada, retendo as soluções antigas e gerando novas soluções a partir da população anterior P_t. Isto é realizado aplicando operadores genéticos nas soluções de P_t. Três operadores básicos formam a maioria das implementações dos AGT:

1. Operador reprodução (seleção);
2. Operador *crossover* (recombinação); e
3. Operador mutação.

Outros operadores também têm sido apresentados, mas são derivados dos três anteriores ou criados para problemas específicos e serão vistos na seção 2.6. A idéia é melhorar a qualidade total das soluções a cada iteração. O processo iterativo pára quando um certo nível de qualidade das soluções é atingido ou observada a convergência.

2.4.1 - Operador reprodução

Com o operador reprodução, a chance de uma estrutura ser selecionada para permanecer na nova população, P_{t+1}, é proporcional a sua adaptação Este operador

atribui a cada estrutura uma taxa de amostragem *T(s,t)*, definida como o número esperado de filhos a serem gerados por esta estrutura na geração *t*. Assim, dadas duas estruturas *s'* e *s''*, se $\eta(s') > \eta(s'')$, então $T(s',t) > T(s'',t)$. A taxa de amostragem mais freqüente é

$$T(s,t) = \frac{\eta(s)}{\bar{\eta}(t)} \tag{2.1}$$

Onde o numerador é a adaptação de *s* e o denominador é a adaptação média da população P_t. Observe que estruturas com adaptação superior à média terão maior probabilidade de sobreviver do que estruturas com adaptação inferior à média.

2.4.2 - Operador *crossover*

Usando somente o operador reprodução, a população se tornará mais e mais homogênea após cada geração. O operador *crossover* é incluído no algoritmo genético com dois propósitos. Primeiro, introduz novas estruturas através da recombinação das estruturas existentes. Segundo, ajuda a eliminar esquemas com baixa adaptação. O funcionamento é o seguinte: dadas duas estruturas *s'* e *s''*, elas permutam subcadeias de acordo com o ponto de *crossover* para formar duas novas estruturas. Para evitar um desenvolvimento caótico, nem todas estruturas na nova população são geradas através do operador crossover. A probabilidade de aplicar este operador, ou simplesmente a taxa de *crossover*, é denotada por p_c. Existem diversos tipos de operadores *crossover* e, os operadores de 1-ponto e 2-pontos, ilustrados nas figuras 2.1 e 2.2, são os mais usados.

INDIVÍDUO PAI 1	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**
INDIVÍDUO PAI 2	0	1	0	0	0	0	1	1	1
RUPTURA			*						
FILHO 1	**1**	**1**	0	0	0	0	1	1	1
FILHO 2	0	1	**0**	**1**	**1**	**0**	**0**	**0**	**1**

Figura 2.1: *Crossover* de um ponto.

INDIVÍDUO PAI 1	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**
INDIVÍDUO PAI 2	0	1	0	0	0	0	1	1	1
RUPTURA			*			*			
FILHO 1	**1**	**1**	0	0	0	0	**0**	**0**	**1**
FILHO 2	0	1	**0**	**1**	**1**	**0**	1	1	1

Figura 2.2: *Crossover* de dois pontos.

2.4.3 - Operador mutação

Este operador introduz mudanças aleatórias nas estruturas da população, através da alteração, com probabilidade (ou taxa de mutação) p_m, de um símbolo da estrutura. Veja figura 2.3.

ESTRUTURA	1	1	0	0	1	1	0	0	1
MUTAÇÃO				*					
NOVA ESTRUTURA	1	1	0	1	1	1	0	0	1

Figura 2.3: Mutação no quarto elemento.

Além disso, uma implementação completa do AGT, envolve especificar um número de parâmetros, incluindo tamanho da população, número de gerações e escala da função de adaptação. O algoritmo genético na sua forma básica, também denominada canônica, poder ser vista no algoritmo 2.1.

```
AGT()
{ Um algoritmo genético para minimizar uma função}
Iniciar
t=0;
inicializar P_t;
Avaliar P_t;
Enquanto não (condição de parada) fazer
        Iniciar
                t = t +1;
                Selecionar P_t de P_t-1; {operador reprodução}
                Recombinar P_t; {operador "crossover" e mutação}
                Avaliar P_t;
        Fim
Fim
```

Algoritmo 2.1: Algoritmo Genético Tradicional.

2.5 – FUNCIONAMENTO DA ALGORITMO GENÉTICO

Embora os AGT sejam simples de descrever e programar, seu funcionamento pode ser complicado, e existem muitas questões abertas a respeito de como e porque os algoritmos genéticos funcionam.

Na teoria tradicional, AGT age descobrindo, enfatizando e recombinando bons *blocos de construção* nas soluções. A idéia é que boas soluções são produzidas por segmentos de *bons blocos*. Estes blocos são representados pelos esquemas. Assim, para um alfabeto binário, qualquer cadeia de comprimento b é uma instância de 2^b diferentes esquemas. Então, qualquer população de n cadeias contêm instâncias entre 2^b e $n \times 2^b$ de diferentes esquemas. Se todas cadeias são idênticas, então existem instâncias de exatamente 2^b diferentes esquemas; caso contrário, o número é menor ou igual a $n \times 2^b$. Isto significa que, a uma dada geração, enquanto o AGT está calculando explicitamente a adaptação das n cadeias da população, está estimando implicitamente a adaptação média de um número muito maior de esquemas, onde a adaptação média de um esquema é definida como a média da adaptação de todas possíveis instâncias deste

esquema. Por exemplo, a instância produzida na figura 2.4 pode ser tanto resultado do esquema 1 como do esquema 2 e, quando esta instância é avaliada, estamos avaliando indiretamente os dois esquemas.

Esquema 1	(# 1 # 0 0)
Esquema 2	(# 1 # 0 #)
Instância	(1 1 1 0 0)

Figura 2.4: Instância obtida a partir dos esquemas 1 e 2.

Como os esquemas não são explicitamente representados e avaliados pelo AGT, as estimativas das médias de adaptação dos esquemas não são calculados ou armazenados explicitamente pelo AGT. No entanto, como será visto a seguir, o procedimento do AGT, em termos do aumento e diminuição do número de instâncias de dados esquemas na população pode ser descrito e assim, as médias de adaptação podem ser calculadas e armazenadas.

Podemos calcular a dinâmica do crescimento e diminuição de instâncias de esquemas conforme a seguir. Considere *H* ser um esquema que possua pelo menos uma instância na população no instante *t*. Considere *M(H,t)* o número de instâncias de *H* no instante *t*, e considere *û(H,t)* a adaptação média observada de *H* no instante *t* (isto é, a adaptação média das instâncias de *H* à população no instante *t*). Queremos calcular *E(M(H,t+1))*, o número esperado de instâncias de *H* no instante *t*+1. O número esperado de filhos da cadeia *s*, conforme a equação 2.1, é igual a $\eta(s)/\bar{\eta}(t)$, onde *η(s)* é a adaptação de *s* e $\bar{\eta}(t)$ é a adaptação média da população no instante *t*. Então, assumindo que *s* esteja na população no instante *t*, e $s \in H$, denota que *s* é uma instância de *H*, e por enquanto ignorando os efeitos de recombinação e mutação, nós temos por definição:

$$E(M(H,t+1)) = \sum_{s \in H} \frac{\eta(s)}{\bar{\eta}(t)} = \left(\frac{\hat{u}(H,t)}{\bar{\eta}(t)}\right) M(H,t) \qquad (2.2)$$

desde que:

$$\hat{u}(H,t) = \frac{(\sum_{s \in H} \eta(s))}{M(H,t)} \tag{2.3}$$

para *s* na população no instante *t*. Então, mesmo o AGT não calculando $\hat{u}(H,t)$ explicitamente, o aumento ou diminuição de instâncias do esquema na população dependem desta quantidade.

Recombinação e mutação podem destruir ou criar instâncias de *H*. Por enquanto vamos considerar apenas o efeito destrutivo, isto é, que diminuem o número de instâncias de *H*. Incluindo estes efeitos, nós modificamos o lado direito da equação 2.2. Considere p_c ser a probabilidade de um *crossover* (recombinação onde os pais foram divididos em apenas duas partes) sobre uma cadeia, e suponhamos que uma instância do esquema *H* seja particionada para ser um pai. O esquema *H* é dito *sobrevivente* a uma recombinação (de um ponto) se um dos seus filhos também é uma instância do esquema *H*. Podemos dar um limite inferior na probabilidade $\rho_c(H)$ que *H* sobreviva a um *crossover* de um ponto:

$$\rho_c(H) \geq 1 - p_c(\frac{\psi(H)}{b-1}) \tag{2.4}$$

Onde $\psi(H)$ é definido como o comprimento de *H* e *b* é o comprimento da cadeia de bits no espaço de busca. Isto é, *crossover* (recombinação) ocorrendo dentro de um comprimento definido de *H* pode destruir *H* (i.e., podem produzir filhos que não são instâncias de *H*). Assim, nos multiplicamos a fração da cadeia que *H* ocupa, pela probabilidade de recombinação para obter um limite superior da probabilidade que ela será destruída. Subtraindo este valor de 1, obtemos um limite inferior da probabilidade de sobrevivência $\rho_c(H)$. Em resumo, a probabilidade de sobrevivência na recombinação é maior para esquemas menores.

Os efeitos destrutivos da mutação podem ser quantificados conforme segue: Sendo p_m a probabilidade de qualquer bit ser mutado, então $\rho_m(H)$, a probabilidade que o esquema *H* sobreviva a mutação de uma instância *H* é igual a $(1-p_m)^{o(H)}$, onde $o(H)$ é a ordem de *H*. Isto é, para cada bit, a probabilidade que o bit não seja mutado é $1-p_m$.

Assim, a probabilidade que bits do esquema *H* não serem mutados, é esta quantidade multiplicada por ela mesma, *o(H)* vezes. Em resumo, a probabilidade de sobrevivência numa mutação é maior para esquemas de baixa ordem. Os efeitos destrutivos podem ser usados para completar a equação 2.2. Então:

$$E(M(H,t+1)) \geq \frac{\hat{u}(H,t)}{\bar{\mu}(t)} M(H,t)(1-p_c \frac{\psi(H)}{b-1})[(1-p_m)^{o(H)}] \quad (2.5)$$

A equação 2.5 é conhecida como o Teorema do Esquema (Holland,1975 e Goldberg,1989) e descreve o crescimento de uma população à próxima. Este teorema implica em afirmar que esquemas curtos e de baixa ordem, cujas adaptação permanecem acima da média, recebem aumento exponencial do número de instâncias com o avançar das iterações, desde que as instâncias destes esquemas não sejam particionadas e permaneçam com adaptação acima da média, aumentando por um fator $\hat{u}(H,t)/\bar{\eta}(t)$ a cada geração.

O teorema do esquema da equação 2.5 é um limite inferior, uma vez que somente os efeitos destrutivos do *crossover* e mutação foram considerados. Entretanto, na teoria clássica, o operador crossover é a maior fonte de poder do algoritmo genético, com a capacidade de recombinar instâncias de bons esquemas e produzir novas instâncias igualmente boas ou melhores ainda. A suposição de que é desta maneira como o algoritmo genético opera é denominada Hipótese do Bloco Construtivo (“Building Block Hypothesis”) (Goldberg, 1989).

Quando o algoritmo genético esta avaliando uma população de *n* estruturas, ele está implicitamente estimando a adaptação média de todos esquemas que estão presentes na população, e aumentando ou diminuindo sua representação, de acordo com o teorema do esquema. A avaliação simultânea implicita de um grande número de esquemas numa população de *n* estruturas é conhecida como *Paralelismo Implícito* (Holland, 1975). Na seleção são obtidas, gradativamente, mais e mais instâncias de alguns esquemas, cujas avaliações estão acima da média.

Para Holland (1975) a proposta da mutação era previnir a perda de diversidade de uma determinada posição de bit. Por exemplo, sem a mutação, todas

estruturas de uma população poderiam, eventualmente, apresentar o número 1 para a primeira posição da estrutura numa determinada geração e não seria mais possível obter uma cadeia iniciando com bit zero.

O teorema do esquema e suas implicações ainda é tema de muitas críticas e discussões.

2.6 – EXTENSÕES AO ALGORITMO GENÉTICO BÁSICO

John Holland lançou as bases para o algoritmo genético, mas para resolver e modelar problemas reais, verificou-se que o algoritmo genético simples tinha poder limitado em vários aspectos. Consequentemente surgiram diferentes propostas para a representação dos problemas, para formas de seleção e os operadores recombinação e mutação. Veremos algumas destas idéias nesta seção, no entanto, sem a pretensão de esgotar o assunto, uma vez que os algoritmos genéticos são fonte de intensa pesquisa e novas propostas surgem constantemente. Inicialmente, porém, mostraremos uma formulação para o algoritmo genético, denominada "Messy Genetic Algorithm - MGA", proposta por Goldberg e seus colegas (Goldberg *et al*.,1989; Goldberg *et al*.,1993, kargupta,1995), pois este algoritmo apresenta algumas das características do algoritmo genético construtivo, proposto nesta tese.

2.6.1– *"Messy genetic algorithms"*

A idéia geral, que têm motivação biológica, é melhorar o desempenho das funções de otimização do algoritmo genético, através de um processo progressivo de junção de pequenas cadeias bem adaptadas e, através de sucessivas gerações, obter boas estruturas.

Considere um problema particular de otimização com soluções candidatas representadas por cadeias binárias. No MGA, cada bit possui a identificação da sua posição na cadeia, mas para uma dada estrutura, nem todas posições são especificadas ou algumas posições são usadas por diferentes bits. Por exemplo, em um problema de quatro bits, as seguintes estruturas poderiam ser encontradas na população: {(1,0)(2,0)(4,1)(4,0)} e {(3,1)(3,0)(3,1)(4,0)(4,1)(3,1)}. A primeira estrutura não

especifica nenhum valor para a posição 3 e dois valores para a posição 4. A segunda não especifica valores para as posições 1 e 2, dois valores para a posição 4 e quatro valores para a posição 3.

Para avaliação, o método verifica as estruturas e o seguinte procedimento é adotado:

1. Quando existem dois ou mais bits atribuídos a mesma posição: varre-se a cadeia da esquerda para direita e considera-se válida a primeira atribuição obtida;
2. Quando não existe atribuição para uma determinada posição: preenche-se a posição com #, formando, desta forma, esquemas.

Por exemplo, a primeira cadeia acima, dá origem ao esquema 00*1.

Para avaliar os esquemas, inicialmente Goldberg e seus colegas propuseram o seguinte procedimento: para um dado esquema são gerados aleatoriamente valores para as posições que ainda não têm atribuição (estão com #) e assim através da função objetivo é realizado sua avaliação. O método é repetido diversas vezes e a média das avaliações é obtida. A idéia era estimar a qualidade média dos esquemas. Mas logo, perceberam que a variância desta média era muito alta, para que a média refletisse algum resultado significativo.

Goldberg e seus colegas usaram, então, um método denominado *"templates" competitivos*. A idéia não era estimar a avaliação média dos esquemas, mas ver se um esquema poderia render um ótimo local. Um ótimo local é, por definição, uma cadeia que não obtém melhor avaliação através da alteração de um único bit. O método inicia obtendo um ótimo local através da técnica conhecida como subida de encosta ("hill climbing") e só então inicia o MGA, e quando calcula a avaliação de um esquema, preenche as posições com # com valores obtidos do ótimo local (obtido previamente). Se, através deste procedimento, um esquema consegue melhorar o ótimo local atual, a busca segue com esta nova estrutura.

O MGA funciona em duas fases: fase inicial e fase de justaposição. O objetivo da fase inicial é criar uma população de esquemas pequenos e promissores e o

objetivo da fase de justaposição é unir estes esquemas de uma maneira útil. No método, supõe-se uma determinada ordem, *o(H)*, para os esquemas e forma-se a população inicial, enumerando completamente todos os esquemas. Por exemplo, se a dimensão da população é *n*=8 e *o(H)*=3, a população inicial pode ser:

{(1,0)(2,0)(3,0)},{(1,0)(2,0)(3,1)},...,{(1,1)(2,1)(3,1)},{(1,0)(2,0)(4,0)},{(1,0)(2,0)(4,1)},...,

{(6,1)(7,1)(8,1)}

Após a população inicial ter sido formada e avaliada usando os “templates” competitivos, a fase inicial continua através da seleção (fazendo cópias de cadeias, de acordo com sua avaliação, sem *crossover* e mutação) e retirando metade da população em intervalos regulares. Em determinada geração (um parâmetro do algoritmo) a fase inicial termina e inicia a fase de justaposição.

O tamanho da população permanece fixa, a seleção continua e dois operadores de justaposição são introduzidos. O primeiro operador, divide uma cadeia em uma posição aleatória. Por exemplo: {(2,0)(3,0)(1,1)(4,1)(6,0)} poderia ser cortado após a segunda posição e produzir: {(2,0)(3,0)} e{(1,1)(4,1)(6,0)}. O segundo operador une duas cadeias. Por exemplo: {(1,1)(2,1)(3,1)} e {(1,0)(4,1)(3,0)} poderia ser unido produzindo {(1,1)(2,1)(3,1), (1,0)(4,1)(3,0)}.

Na representação do MGA os dois operados sempre produzem cadeias válidas. O objetivo é que a fase inicial produza todos os “blocos construtivos” necessários para criar uma cadeia ótima e um número suficiente para que isto ocorra rapidamente na fase de justaposição.

Goldberg *et al*. (1989) realizaram uma análise matemática robusta para este algoritmo, justificando porque o novo método deveria funcionar melhor do que o algoritmo genético simples e mostraram sua eficiência empiricamente.

Os dois principais problemas do método são:

1. Determinar um valor adequado para a ordem do esquema;
2. Para problemas reais, a combinação de todos esquemas passa a ser intratável.

Assim, Goldberg *et al.* (1993) propuseram, como solução para resolver o problema da explosão combinatorial dos esquemas iniciais, usar uma inicialização completamente probabilística.

Infelizmente, mesmo com uma inicialização completamente probabilística, o tamanho inicial necessário da população aumenta exponencialmente com a ordem para os esquemas, e desta forma, o método somente pode ser usado em problemas em que é possível uma ordem pequena nos esquemas. Goldberg e seus colegas assumem que na maioria dos problemas de interesse isto ocorre, mas ainda não realizaram a devida demonstração.

2.6.2– Representação

A maneira como as soluções candidatas são representadas é de fundamental importância no sucesso dos métodos de busca. A maioria dos algoritmos genéticos apresenta representação binária, com cadeias de tamanho fixo, devido a diversas razões. Uma dessas razões é histórica: Holland e seus colegas concentraram-se nesta forma de representação, e produziram a maioria da teoria existente sobre AGT considerando cadeias binárias de tamanho fixo. Holland também argumentou que a representação binária aumenta o paralelismo implícito, pois as estruturas formadas contém mais esquemas que os de outros tipos de representação; idéia questionada por alguns pesquisadores (Antonisse, 1989).

Mas, a codificação binária não é uma representação natural para a maioria dos problemas e novas formas foram propostas, sendo que em diversos casos a representação usando múltiplos caracteres ou valores reais apresentaram melhor desempenho (Wright, 1991;Janikow e Michalewicz,1991).

Koza (1992,1994) propôs uma representação usando codificação em árvore, que apresenta algumas vantagens, como não limitar o tamanho das estruturas, mas em contrapartida podem produzir árvores muito grandes e difíceis de manipular.

Além disso, outras formas de representação têm recebido considerável atenção, principalmente as que consideram características do problema e usam estas

informações no processo de representação. Davis (1991) advoga o uso da mais natural possível representação para o problema, e a adaptação do algoritmo genético a esta representação.

2.6.3- Seleção

O mecanismo de seleção original proposto foi projetado com base na evolução natural, i.e., indivíduos bem adaptados possuem maior probabilidade de serem selecionados para reproduzir. Na técnica denominada *seleção por roleta* (Holland, 1975) a probabilidade de um indivíduo ser selecionado é proporcional ao valor da sua adaptação dividido pela média da adaptação de todos os indivíduos da população (seção 2.4).

O procedimento *elitista*, introduzido por De Jong (1975), força o algoritmo genético a preservar os melhores indivíduos a cada geração, enquanto os filhos substituem os indivíduos menos adaptados. Os melhores indivíduos poderiam ser perdidos se eles não fossem selecionados para reprodução ou se eles fossem destruídos pelos operadores *crossover* ou mutação.

Freqüentemente, é conveniente que a forma de seleção varie ao longo da busca. Uma abordagem deste tipo é denominada *seleção de Boltzmann*, que pode ser encontrada em Goldberg (1990) e Prügel-Bennett e Shapiro (1994). Existe um controle de “temperatura” que varia durante a busca. A temperatura inicia alta, o que significa que a pressão na seleção é baixa, isto é, todos indivíduos possuem alguma probabilidade de reprodução. A medida que a temperatura diminui, a pressão na seleção aumenta, fazendo com que os indivíduos mais adaptados tenham maior probabilidade de reprodução.

Na *seleção por torneio* (Goldberg e Deb, 1991), duas estruturas são escolhidas aleatoriamente da população. Então, um número aleatória r é escolhido entre 0 e 1. Se $r<l$ (onde l é um parâmetro, por exemplo, 0.60), o indivíduo mais adaptado será selecionado para reproduzir; caso contrário, o indivíduo menos adaptado será selecionado. Os dois indivíduos retornam para a população e podem ser selecionados

novamente. A seleção por torneio apresenta a vantagem de exigir que apenas os indivíduos selecionados sejam avaliados quanto a sua adaptação.

Na *seleção por truncamento* (Mühlenbein e Schlierkamp-Voosen,1994), somente um percentual dos melhores indivíduos são considerados para gerar os filhos. E na *seleção com ordenação* (baker, 1985) os indivíduos da população são ordenados de acordo com a adaptação e o valor esperado de cada indivíduo (isto é, o número esperado de vezes que um indivíduo será selecionado para reproduzir) depende da ordem em que se encontra ao invés de sua adaptação absoluta. Não existe necessidade de escala para o valor da adaptação e evita problemas de convergência, que ocorrem em algumas formas de seleção.

Na seção 2.4 vimos a AGT na sua forma básica, onde todos os indivíduos da população são substituídos pelos seus filhos no final de cada geração. Entretanto, alguns dos filhos podem ser idênticos aos pais, uma vez que a mutação e recombinação é realizada a uma determinada taxa. No denominado, algoritmo genético *de estado fixo*, somente alguns poucos indivíduos são substituídos. A seleção de estado fixo foi analisada por Syswerda (1991) e De Jong e Sarma (1993).

2.6.4- Operadores genéticos

Os procedimentos recombinação usados também evoluíram do simples *crossover* para mecanismos mais sofisticados. Bersini e Seront (1992) apresentaram o operador *crossover*-mutação que considera três cadeias (cromossomos) durante a recombinação. Eiben *et al.* (1994) usaram um procedimento de recombinação com múltiplos pais para otimização de funções, descrevendo procedimentos com mais de dez indivíduos pais.

Uma visão comum da comunidade científica atribuía ao operador *crossover* a função de variar e inovar no algoritmo genético e a mutação uma simples função secundária. Esta visão difere da programação evolutiva e estratégias de evolução (vista no início deste capítulo), nos quais a mutação é o principal operador. Spears (1993) realizou alguns estudos comparativos entre o poder do *crossover* e a mutação.

2.7 – AUTO-ADAPTAÇÃO

Assim como em outras meta-heurísticas, o sucesso do algoritmo genético depende de um conjunto de parâmetros de controle, tais como o tamanho da população, taxas de mutação e recombinação (*crossover*) e o tipo de seleção usado. Embora existam algumas tentativas para determinar parâmetros robustos para uma grande variedade de aplicações, recentemente a auto-adaptação, onde os parâmetros adaptam-se dinamicamente durante o processo de otimização, tem recebido mais atenção.

Fogarty (1989) foi um dos primeiros a trabalhar nesta área e descreveu diferentes maneiras para taxas de variação de mutação durante otimização de aplicações em engenharia. Nos seus testes, ele mostra que os benefícios da mutação dinâmica dependem da composição da população inicial.

Bäck (1992) e Smith e Fogarty (1996) incluem a auto-adaptação da taxa de mutação na codificação de cada indivíduo. Eles investigaram diversas estratégias de seleção.

Julstrom (1995) descreveu uma abordagem, em que existe uma adaptação probabilística dos operadores *crossover* e mutação. A cada operador é atribuído um crédito, dependendo da contribuição que ele realizou na criação de filhos em relação a média da população. Para isso, é necessário uma árvore que armazena cada indivíduo, descrevendo os operadores que criaram seus ancestrais. Aplicando este método ao problema do caixeiro viajante, ele mostrou que o operador *crossover* foi mais importante, ao contrário da maioria das opiniões.

2.8 – PRINCIPAIS GRUPOS DE PESQUISA

Existem atualmente conferências internacionais dedicadas a tratar de algoritmos genéticos. O número de artigos sobre AGT crescem rapidamente. Alander (1994) catalogou uma extensa bibliografia de mais de 2500 livros, artigos, teses dedicados AGT. Alander mostrou que existe um rápido crescimento, com uma taxa anual de aproximadamente 40%. Existem alguns laboratórios de pesquisa famosos

trabalhando na computação evolutiva. Nos EUA, temos como exemplo o grupo de Holland na Universidade de Michigan, o de Golberg em Ilinois (http://gal4.ge.uiuc.edu), o de Goodman e Punch no Estado de Michigan (http://isl.msu.edu//GA/), o de De Jong na Universidade George Mason (http://www.cs.gmu.edu/research/gag/). Também existem trabalhos no Centro Naval de Pesquisa Aplicada em Inteligência Artificial (http://www.aic.nrl.navy.mil) e no Instituto Santa Fé (http://www.santafe.edu). No Reino Unido existem o grupo de Flemings em Sheffield (http://www.shef.ac.uk/uni/projects/gaipp/index.html), o grupo de computação evolutiva da Universidade do Oeste da Inglaterra em Bristol (http://www.btc.uwe.ac.uk/evol/index.html) e o da Universidade de Edinburgh (http://www.dai.ed.ac.uk). Na Alemanha, existe o grupo de pesquisa em Sistemas Adaptativos de Mühlenbein no GMD (http://borneo.gmd.de/AS/pages/as.html).e o grupo de Rechengerg na Universidade Técnica de Berlim (http://lautaro.fb10.tu-berlin.de). No Brasil, existem diversos pesquisadores e instituições com trabalhos publicados na área (Lorena e Lopes, 1996b; 1997). Além destes grupos, existem muitos outros pesquisadores trabalhando na área.

3 – ALGORITMO GENÉTICO CONSTRUTIVO

Baseados na idéia de que boas soluções para problemas de otimização são obtidas através de bons blocos construtivos e nas idéias de um artigo de Lorena e Lopes (1996a) na resolução do problema de cortes de estoques, propusemos o Algoritmo Genético Construtivo – AGC.

O AGC inicia com uma população de esquemas, que carregam informações sobre propriedades estruturais do problema. Estes esquemas são avaliados e os melhores são incentivados a se recombinarem, de tal forma que através de sucessivas gerações, novos esquemas ou estruturas completas sejam produzidas. Os esquemas ou estruturas que não obtiverem boa avaliação são eliminados da população através de um critério de poda. A primeira questão que surge é: Como avaliar esquemas? Vimos no segundo capítulo que o método denominado "Messy Genetic Algorithms" proposto por (Goldberg *et al.*, 1989) apresentaram algumas formas de avaliação, que no entanto, não obtiveram o sucesso esperado. Por outro lado, no algoritmo genético tradicional, os esquemas são avaliados indiretamente através das instâncias que produzem.

Assim, propusemos uma forma de avaliar diretamente esquemas, usando algumas idéias do algoritmo A*, e do já mencionado artigo de Lorena e Lopes (1996a). Portanto, vamos inicialmente mostrar como funciona o algoritmo A*, para em seguida, o AGC ser apresentado.

*3.1 – ALGORITMO A**

O algoritmo de busca denominado A*, apresentado pela primeira vez por Hart *et al.* (1968; 1972), é conhecido por ser usado para obter o menor caminho num grafo. Muitas formas de resolver um problema, podem ser vistas como uma busca em um grafo, onde os vértices representam soluções (possivelmente parciais) para o problema a ser resolvido e as arestas representam passos entre as soluções. Por exemplo, os vértices poderiam representar cidades e as arestas ruas; ou os vértices poderiam ser posições num jogo e as arestas movimentos legais.

Iniciando a partir de um ou mais vértices iniciais, o algoritmo A* busca por um *estado (vértice) meta*, que satisfaça a condição objetivo. O procedimento de busca funciona através da aplicação de um ou mais operadores no(s) vértice(s) inicial(is) para produzir um ou mais vértices sucessores que são adicionados a uma lista de candidatos a futura expansão. A cada ciclo o vértice de menor custo que ainda não foi expandido é escolhido para futura expansão. O algoritmo A* usa uma função custo da seguinte forma: $c(n) + h(n)$, onde $c(n)$ é o custo de ir do vértice inicial até o vértice n, e $h(n)$ (a função heurística) é a estimativa do custo de encontrar o vértice meta a partir do vértice n. A busca continua até que o vértice meta seja encontrado ou todos os vértices no grafo sejam explorados. Para previnir ciclos e evitar esforços desnecessários, o algoritmo mantém uma lista de vértices candidatos não expandidos (a lista denominada ABERTA) e uma lista de vértices que já foram visitados (a lista denominada FECHADA). O algoritmo A* pode ser resumido nos seguintes passos:

1. Inicializar a lista ABERTA com o vértice inicial e o avaliar;
2. Repetir até que a lista ABERTA esteja vazia:
   - Remover o primeiro vértice da lista ABERTA e adicionar a lista FECHADA;
   - Verificar se o critério de parada é satisfeito (isto é, o vértice representa um vértice objetivo), se sim, parar;
   - Expandir o vértice aplicando os operadores de busca;
   - Rejeitar qualquer sucessor inválido (isto é, os vértices já pertencentes a lista ABERTA ou a lista FECHADA);
   - Avaliar os vértices sucessores remanescentes e inserí-los, ordenados conforme seu custo, na lista ABERTA;

Algoritmo 3.1: Algoritmo A*.

O algoritmo A* é completo (isto é, dados recursos suficientes, ele garante encontrar a solução ótima se ela existe) e ótimo (isto é, ele garante encontrar a melhor solução). Entre os algoritmos ótimos deste tipo – algoritmos que buscam a partir de um vértice inicial – o algoritmo A* é eficiente para qualquer função heurística dada *h(n)* que sub-estimar o custo real de ir do vértice *n* até o vértice objetivo, no sentido de que não existe outro algoritmo ótimo que garante expandir menos vértices que o algoritmo A*. Para problemas reais de grande dimensão, a memória necessária no algoritmo A* é impraticável e em muitos casos é necessário usar variações no A*, tais como o algoritmo denominado A_ε* (Pearl, 1982), o qual garante encontrar soluções que podem ser piores que o ótimo, em no máximo ε.

Após esta introdução ao algoritmo A*, vamos mostrar como as idéias foram evoluindo até o desenvolvimento do algoritmo genético construtivo.

3.2 – BREVE HISTÓRICO

O Algoritmo Genético Construtivo surgiu a partir do artigo “A dynamic list heuristic for 2D-cutting” de Lorena e Lopes (1996a). Neste artigo, o problema de cortes em 2D é atacado, sendo que o objetivo da heurística é eliminar a explosão combinatorial da seqüência de combinações horizontais e verticais de retângulos, observada no artigo de Wang (1983). A heurística usa uma lista dinâmica, onde soluções parciais (constituídas de combinações de retângulos) são avaliadas e mantidas. Analogamente ao

algoritmo A*, as soluções candidatas com melhores avaliações são exploradas primeiro. As avaliações são baseados em duas funções que medem a *perda* de área local e total na combinação de dois retângulos. O objetivo é minimizar esta perda. Além disso, uma lista tabu auxiliar é usada para evitar excessivas repetições na busca.

Com os bons resultados obtidos no problema de cortes e através de uma análise mais profunda do método, verificamos que a mesma heurística (com as devidas adaptações de parâmetros, típica das meta-heurísticas) poderia ser usada para diferentes problemas combinatoriais. A partir deste momento, buscamos formalizar diferentes problemas combinatoriais que poderiam ser atacados pela nova meta-heurística. Inicialmente, a maior dificuldade consistia em identificar as funções de avaliação. Uma abordagem bem sucedida foi a coloração de grafos apresentada na dissertação de mestrado de (Ribeiro Filho, 1997). O problema de coloração de grafos, consiste em obter o menor número possível de cores para colorir um grafo, de tal forma que dois vértices adjacentes (unidos por uma aresta) não tenham a mesma cor.

Durante o desenvolvimento desta tese, verificamos que a heurística apresentava um conjunto de características típicas dos algoritmos genéticos, tais como uma população de soluções candidatas, seleção, recombinação e mutação. Verificamos que a definição de esquema apresentava semelhança com as *soluções parciais* do método que vinhamos desenvolvendo e compartilhava com a idéia de que boas soluções são geradas a partir de bons blocos construtivos. Também observamos que a heurística continha algumas características do “Messy Genetic Algorithm”. No entanto, a forma direta de avaliar esquemas garantia avanço real do novo método. Desta forma, resolvemos adotar a nomenclatura do algoritmo genético e procuramos destacar as suas principais semelhanças e diferenças.

3.3 – INICIANDO COM UM EXEMPLO

Com o objetivo de facilitar a compreensão do Algoritmo Genético Construtivo, acreditamos que a introdução de um exemplo poderá ser útil; para isto, usamos o problema das *p*-medianas, que será formalmente definido no capítulo 4.

Para o problema das *p*-medianas um grafo completo com *m* vértices é fornecido, onde as arestas do grafo representam as distâncias entre os vértices. O objetivo consiste em localizar *p* (*p*<*m*) medianas (vértices) no grafo e atribuir os demais vértices (que não são medianas) a estas medianas, de forma a minimizar a soma total das distâncias dos vértices as medianas correspondentes. Suponhamos que desejamos localizar *p*=3 medianas no grafo com *m*=10; uma solução possível seria a mostrada na figura 3.1.

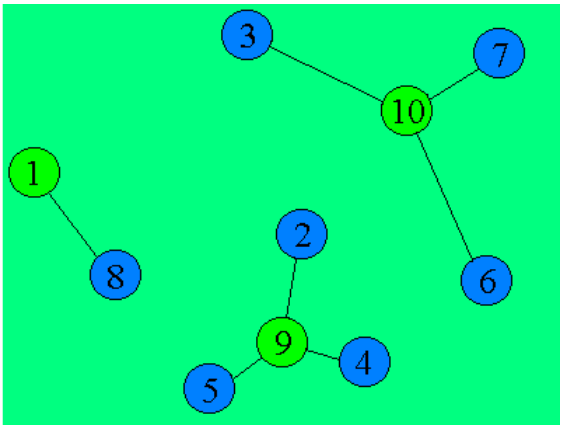


Figura 3.1: Possível solução para o problema com *m*=10 e *p*=3.

3.4 - REPRESENTAÇÃO

O primeiro passo na implementação de um AGC é a definição de uma codificação que permite o mapeamento entre soluções do problema de otimização e as estruturas. Uma estrutura pode ser representada pela cadeia $s_k = (s_{k1},s_{k2},...,s_{km})$, onde *m* é o número de variáveis no problema (o número de vértices no caso das *p*-medianas). No exemplo das *p*-medianas, um esquema pode ser representado por uma cadeia que apresenta três tipos de informações, os quais, denotamos:

- 1 : vértice representando uma mediana;
- 2 : vértice que é atribuído a mediana mais próxima;

- # : vértice curinga, ou seja, poderá futuramente ser um vértice mediana ou um vértice que será atribuído a uma mediana.

Desta forma a solução da figura 3.2, poderia ser representada pela seguinte cadeia: (1 # 2 2 2 # 2 2 1 1). Percebemos que os vértices 2 e 6 (posições 2 e 6 na cadeia), embora sejam dados do problema analisado, não fazem parte das informações agregadas até este instante ao processo de busca.

Observamos que existe uma importante diferença entre a estrutura mostrada na figura 3.1, s_1 = (1 2 2 2 2 2 2 2 1 1) e a estrutura (esquema) da figura 3.2, s_2 = (1 # 2 2 2 # 2 2 1 1). Enquanto a estrutura s_1 não apresenta nenhum curinga e portanto a qualidade de sua solução pode ser diretamente analisada através da função objetivo, a estrutura s_2, que também denominaremos esquema, apresenta dois vértices curingas e para que possamos avaliar a sua qualidade será necessário a criação de novas funções de avaliação, que serão apresentados na seção 3.6.

Esclarecemos que nesta tese, denominaremos *estrutura*, qualquer cadeia contendo ou não curingas entre os seus vértices, e quando nos referirmos a *esquema*, estaremos fazendo referência explícita a uma estrutura com curingas.

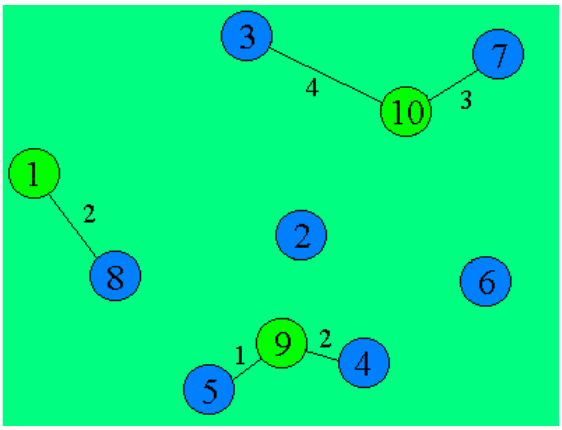


Figura 3.2: Representação de s=(1#222#2211).

3.5 – POPULAÇÃO INICIAL

A população inicial, denominada P_0, é formada por $|P_0|$ esquemas gerados aleatoriamente. É importante que, mesmo sendo aleatória, os esquemas iniciais capturem informações (mesmo que locais ou fracionados) do problema de otimização e que através da população inteira, todas as informações estruturais do problema sejam abrangidas. Freqüentemente é necessário alguma interferência determinística para que este objetivo seja alcançado. Desta forma, buscamos a população de esquemas, em que cada esquema represente um segmento dos dados do problema, para que após os processos de recombinação, possamos agregar estes diferentes segmentos e gerar estruturas completas.

O tamanho da população inicial deve ser suficientemente grande, de forma a capturar todas informações do problema. No entanto, as dimensões podem ser muito menores do que as apresentadas no “Messy Genetic Algorithms”.

3.6 – FUNÇÕES DE AVALIAÇÃO

Definida uma população de estruturas iniciais, é realizada a sua avaliação mapeando o espaço das estruturas em R_+. Considera-se P_α como a população de estruturas (esquemas ou estruturas completas) no instante de evolução α (que será descrito posteriormente) e considerando duas funções f e g, definidas como: $f{:}P_\alpha \rightarrow R_+$ e $g{:}P_\alpha \rightarrow R_+$ e calculadas tais que $f(s_k) \leq g(s_k)$, para toda $s_k \in P_\alpha$. Também definimos um

limite superior comum, $g_{\max} > \underset{s_k \in P_\alpha}{Max}\, g(s_k)$. A esta dupla forma de avaliar s_k denominaremos *avaliação-fg*. Supondo uma população P_α de *m* estruturas, a figura 3.3 mostra uma possível avaliação-*fg* de cada estrutura em P_α e o limite superior g_{max}.

Para cada problema de otimização é necessário definir as funções *f*, *g* e o limite g_{max}. Veja como estas funções foram formalmente definidas para o problema das *p*-medianas no capítulo 4. No exemplo, podemos mostrar como cada uma destas funções pode ser definida. Considere a estrutura s = (1 # 2 2 2 # 2 2 1 1) da figura 3.2, *g* pode ser a soma dos valores das arestas das medianas aos vértices que estão atribuídos a

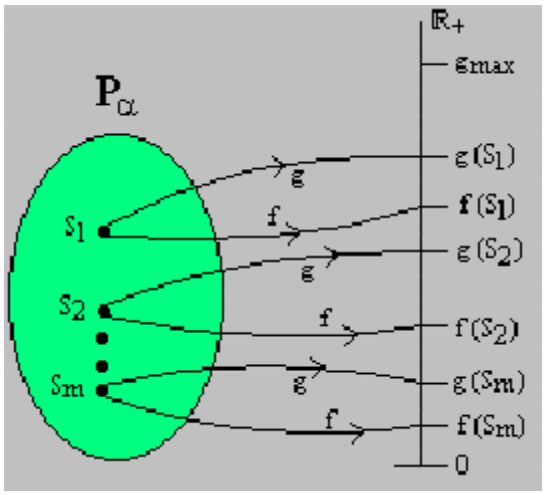


Figura 3.3:Possível avaliação de uma população.

estas medianas, isto é, g = 4 + 2 + 1 + 2 + 3 = 12. Podemos observar na figura 3.2 que após a atribuição dos vértices às medianas mais próximas, existe a formação de *p* agrupamentos ("clusters"). Se para cada agrupamento, observarmos o menor custo de atribuição, e multiplicar este valor pelo número de vértices que estão atribuídos neste agrupamento, e posteriormente somar os valores de todos os agrupamentos, poderemos obter uma boa medida para a função *f*. Para o exemplo, *f* = 1*2 + 3*2 + 2*1 = 10. Observe que definindo *f* e *g* desta maneira garantimos que $g(s) \geq f(s)$, conforme fora definido.

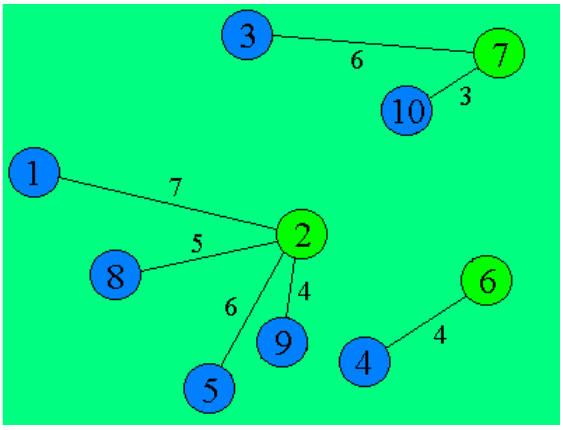


Figura 3.4: Estrutura s=(2 1 2 2 2 1 1 2 2 2).

O limite superior g_{max} é obtido no início do algoritmo, através da geração de uma estrutura completamente aleatória e usando a mesma avaliação de g. Como a estrutura é gerada aleatoriamente, em geral não consegue uma boa avaliação. Por exemplo, a figura 3.4 mostra uma solução obtida aleatoriamente. Podemos observar que g_{max} = 6+3+7+5+6+4+4=35. Para assegurar que g_{max} seja sempre um limite superior, após a recombinação (detalhado na seção 3.8), cada nova estrutura gerada, s_{nova}, é rejeitada se $g_{max} \leq g_{nova}$.

3.7 – SELEÇÃO

Após obter uma representação adequada para o problema, gerar e avaliar a população inicial, é necessário optar por algum método de seleção. Vimos na seção 2.6.3 diversos métodos de seleção para o AGT. Em princípio, qualquer um destes métodos é possível de ser aplicado ao AGC. Nesta tese, usamos um procedimento de seleção, onde a população permanece ordenada de acordo com a qualidade de cada estrutura e os seguintes passos são dados:

1. Escolhemos aleatoriamente uma estrutra entre um percentual pré-definido das melhores estruturas, que denominaremos de s_{base};

2. Escolhemos aleatoriamente outra estrutura, que denominaremos s_{guia}, da população inteira;

3. Agora, as estruturas (esquemas) s_{base} e s_{guia} são unidas de uma forma útil, ou seja, usando algum critério que possa melhorar a qualidade da estrutura s_{nova} gerada.

3.8 – RECOMBINAÇÃO

Após a seleção das estruturas s_{base} e s_{guia}, essas devem ser agregadas para gerar uma nova estrutura s_{nova}. O objetivo é gerar uma estrutura s_{nova} de qualidade superior ao seus ancestrais. Para que isto aconteça, é necessário que no processo de união sejam preservados os *bons blocos construtivos* e agregadas mais informações a estes blocos.

A estrutura s_{base}, por se tratar geralmente da estrutura de mais alta qualidade, serve como fonte principal para geração da nova estrutura, sendo que a estrutura s_{guia} fornece informações adicionais do problema, que devem se juntar aos *bons blocos construtivos* que já constituem a estrutura s_{base}.

Imediatamente, percebemos que o processo de recombinação deve ser cuidadosamente analisado para cada problema de otimização, de forma a atingir os objetivos.

3.9 - 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.

3.10 – 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:

$$d_k = \frac{g(s_k) - f(s_k)}{g(s_k)}, k = 1,..., m. \quad (3.1)$$

O desvio absoluto de $g(s_k)$ é obtido através do produto $d_k\ g(s_k)$. Suponha que seja admitido um desvio percentual global de g_{max}. Denominando d este desvio, então $d\ g_{max}$ é o desvio absoluto admissível de g_{max}. Estamos, então, interessados em comparar os desvios absolutos $d_k\ g(s_k)$ e $d\ g_{max}$.

Usando uma analogia ao algoritmo A* (onde tinhamos $g(n)+h(n)$), nosso *estado-meta* é o desvio absoluto admissível de g_{max} ($d\ g_{max}$), sendo que o desvio absoluto corrente de $g(s_k)$ é $d_k\ g(s_k)$ (análogo a $c(n)$ no algoritmo A*). Para obter o *estado-meta*, uma estimativa do quanto nós teremos adicionalmente no futuro como desvio absoluto pode ser dado por $d[g_{max} - g(s_k)]$ (análogo a $h(n)$ no algoritmo A*). Assim, se

$$d_k\, g(s_k) + d[g_{\max} - g(s_k)] \geq d g_{\max} \quad (3.2)$$

a estrutura s_k não têm futuro e será rejeitada.

Suponhamos que temos as estruturas s_1 = (1 # 2 2 2 # 2 2 1 1) da figura 3.2 e a estrutura s_2 = (1 # 2 2 2 # # 2 1 1) da figura 3.5, teremos respectivamente g_1 = 1 + 2 +2 + 3 + 4 = 12; f_1 = 1*2 + 3*2 + 2*1 = 10; g_2 = 1 + 2 + 2 + 4 = 9 e f_2 = 1*2 + 4*1 + 2*1 = 8 e para os desvios d_{k1} = 0,11 e d_{k2} = 0,16. Suponhamos ainda, que admitamos um desvio d = 0.15 (15%). Neste caso, a estrutura s_2 seria rejeitada, enquanto a estrutura s_1 permaneceria na população.

Observando a equação 3.2, vemos que ela é equivalente a $d_k\ g(s_k) \geq d\ g(s_k)$, ou simplesmente, $d_k \geq d$. Isto implica em afirmar que se o percentual corrente de desvio da estrutura s_k é maior que um percentual de desvio admitido, a estrutura será rejeitada.

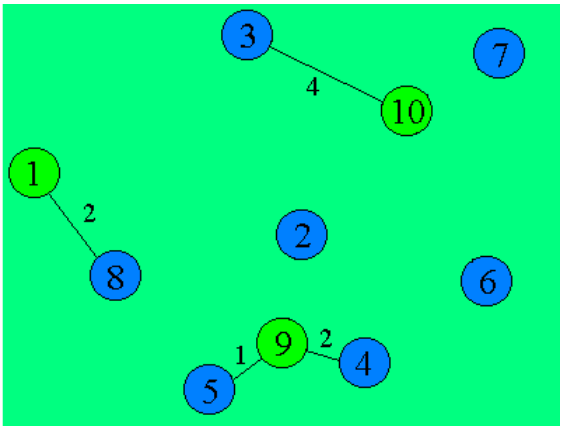


Figura 3.5: Representação de s_2=(1#222##211).

Existem situações em que a simples comparação de d_k com d não é representativa, pois se numa fase do processo evolutivo, uma estrutura apresenta $d_k \geq d$, futuramente, no processo de recombinação, esta estrutura (esquema) poderá gerar uma estrutura com melhor avaliação. Veja no exemplo, onde a estrutura s_1 poderia ter sido derivada da estrutura s_2 e $d_{k1} < d$ enquanto $d_{k2} > d$. Desta forma, introduzimos um *parâmetro de evolução* $\alpha \geq 0$ que modifica a expressão 3.2:

$$d_k g(s_k) + \alpha d[g_{\max} - g(s_k)] \geq d g_{\max} \tag{3.3}$$

Esta expressão é equivalente a

$$d_k \geq d[(1-\alpha)(\frac{g_{\max}}{g(s_k)}) + \alpha] \tag{3.4}$$

Quando $\alpha = 1$, existe a comparação direta $d_k \geq d$, e para $\alpha = 0$, a expressão torna-se:

$$d_k \geq d \frac{g_{\max}}{g(s_k)} \tag{3.5}$$

Considerando que $g_{max}/g(s_k) \geq 1$, no intervalo $0 \leq \alpha < 1$ as estruturas são em geral conservadas, a menos daquelas que apresentem $d_k >> d$. Quando $\alpha > 1$, uma estrutura s_k mesmo tendo $d_k \leq d$, 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 α inicia com 0 e é gradativamente

incrementado, em pequenos intervalos, de geração a geração. P_0 é a população inicial para α=0. Nós denotamos a população no instante de evolução α como P_α.

O parâmetro α pode ser isolado, produzindo a expressão:

$$\alpha \geq \frac{dg_{\max} - d_k g(s_k)}{d[g_{\max} - g(s_k)]} = \delta(s_k) \tag{3.6}$$

No instante da criação, cada estrutura recebe um valor correspondente $\delta(s_k)$ que será comparado com o parâmetro de evolução corrente α. Desta forma, no momento da criação da estrutura, pode-se prever o tempo de sua sobrevivência, sendo que quanto maior $\delta(s_k)$, mais tempo para a estrutura s_k sobreviver e recombinar-se. Ainda, analisando a razão $\delta(s_k)$, 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.

Reescrevendo a equação 3.6, como:

$$\delta(s_k) = \frac{1 - \frac{d_k g(s_k)}{dg_{\max}}}{1 - \frac{g(s_k)}{g_{\max}}} \tag{3.7}$$

podemos isolar os efeitos de d e g_{max} no tamanho da população. Torna-se claro que para pequenos desvios d, a população presente cresce lentamente. Para desvios d maiores, surge o problema de manter na memória uma população de grande dimensão, mas que eventualmente apresenta estruturas de melhor qualidade.

Um efeito indesejável também pode ser observado para um grande número de estruturas $s_k \in P_\alpha$, quando $g_{max} >> g(s_k)$, pois todas estruturas obterão $\delta(s_k) \cong 1$, e podem ser eliminadas em conjunto, de uma geração à próxima, quando α =1. Este efeito pode reduzir drasticamente o tamanho da população de uma geração à próxima. Assim, a definição dos parâmetros d e g_{max} precisam ser cuidadosamente estudados.

Um comportamento típico do tamanho da população, apresenta um incremento nas iterações iniciais, atingindo um limite superior, para em seguida voltar a diminuir, devido o incremento nos valores do parâmetro de evolução.

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.

3.11 – O ALGORITMO

Os passos da forma básica do algoritmo genético construtivo pode ser resumidos, conforme algoritmo 3.2.

```
AGC()  {Algoritmo Genético Construtivo}
α:=0;
Inicializar P_α; {população inicial}
Avaliar P_α;
Enquanto não (condição de parada) fazer
        Para toda s_k ∈ P_α satisfazendo α<δ(s_k) fazer {teste de evolução}
                α: = α +ε;
                Selecionar P_α+ε de P_α; {operador reprodução}
                Recombinar P_α;
                Avaliar P_α;   {calcula adaptação proporcional}
        Fim
Fim
```

Algoritmo 3.2: Algoritmo Genético Construtivo.

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 α. O parâmetro do programa ε, incrementa o valor de α. Após a sua criação, cada estrutura recebe uma avaliação proporcional e um valor $\delta(s_k)$ (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.

O AGC foi testado em três problemas combinatoriais de agrupamentos. O primeiro problema é um problema de localização de facilidades, onde *p* medianas em um grafo devem ser identificadas e os outros vértices atribuídos à mediana mais próxima. Uma medida de distância foi considerada entre os vértices.

O segundo problema considera capacidades na formação dos agrupamentos de medianas e vértices atribuídos. A adaptação do AGC para este problema foi praticamente direta, com a introdução de uma heurística de transição que considera as capacidades e que permite usar a mesma representação de estruturas usada no problema das *p*-medianas.

O último problema considera a partição dos vértices de um grafo em agrupamentos capacitados, onde o objetivo é minimizar a soma dos pesos das arestas entre agrupamentos. Com a introdução de vértices sementes e usando uma heurística de transição, foi possível usar a mesma representação de estruturas dos problemas anteriores.

Os capítulos 4, 5 e 6 descrevem estas aplicações.

4 – PROBLEMA DAS P-*MEDIANAS*

A busca de *p*-medianas num grafo é um problema clássico de localização. O objetivo é localizar *p* facilidades (medianas), de forma a minimizar a soma das distâncias de cada vértice a sua facilidade mais próxima.

Hakimi (Hakimi, 1964; 1965) foi o primeiro pesquisador a formular o problema, para a localização de uma única mediana, em seguida, generalizando para múltiplas medianas. Ele propôs um procedimento simples de enumeração para o problema. O problema é reconhecidamente NP-hard (Garey e Johnson, 1979). Diversas heurísticas têm sido desenvolvidas para o problema das *p*-medianas. Algumas são usadas para obter boas soluções iniciais ou para calcular soluções intermediárias em vértices numa árvore de busca. Teitz e Bart (Teitz e Bart, 1968) propuseram uma heurística simples de permutação. Heurísticas mais complexas exploram uma árvore de busca. Estas estão em Efroymson e Ray,1966; Jarniven e Rajala,1972; Neebe,1978; Christofides e Beasley,1982; Beasley,1985 e Galvão e Raggi, 1989. Algumas abordagens bem sucedidas usam informações primal/dual do problema (Senne e Lorena, 1997; Beasley, 1993).

O problema das *p*-medianas pode ser formulado como um problema de programação inteira binária. Consideremos um grafo completo para uma dada instância, obtido através da aplicação do algoritmo de Floyd (Beasley, 1993) e o conjunto de vértices indexados resultantes $V=\{1,...,n\}$. Para cada $i, j \in V$, sendo $\mu_{ij}(\geq 0)$ o custo

(distância) de atribuir o vértice *j* ao vértice *i*, (μ_{ii}=0), e as variáveis x_{ij}=1, se o vértice *j* é atribuído ao vértice *i* e x_{ij}=0 caso contrário. O problema então é

$$\min z = \sum_{i \in V} \sum_{j \in V} \mu_{ij} x_{ij}, \tag{4.1}$$

$$\text{sujeito a} \quad \sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \tag{4.2}$$

$$\sum_{i \in V} x_{ii} = p, \tag{4.3}$$

$$x_{ij} \leq x_{ii}; i, j \in V \tag{4.4}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in V, j \in V \tag{4.5}$$

As restrições (4.2) e (4.4) asseguram que cada vértice *j* seja atribuída somente a um vértice *i*, que necessariamente deve ser uma mediana. Restrição (4.3) determina o número exato de medianas que devem ser localizadas e a restrição (4.5) nos indica a condição binária de atribuições (ou não) de vértices a medianas.

4.1 – REPRESENTAÇÃO

Como já visto no exemplo da seção 3.4, uma estrutura para o problema das *p*-medianas pode ser uma cadeia com a seguinte representação: s_k=(1 2 2 2 2 2 2 2 1 1), onde cada posição na cadeia representa um vértice. O conjunto *V* de vértices pode ser dividido em dois; o conjunto V_1 das medianas e o conjunto V_2 dos vértices que não são medianas. No exemplo, V_1 = {1,9,10} e V_2 = {2,3,4,5,6,7,8}. Os vértices 1, 9 e 10 são medianas e os demais vértices são atribuídos a mediana mais próxima (de menor custo).

Em muitos casos, teremos estruturas incompletas (ou esquemas) como por exemplo: s_k ==(1 2 # 2 2 # # 2 1 1) e um novo conjunto será definido: $V_3=V-(V_1(s_k)UV_2(s_k))$. No caso do exemplo, teremos: V_1={1,9,10}, V_2={2,4,5,8} e V_3={3,6,7}.

4.2 – POPULAÇÃO INICIAL

A população inicial é formada somente por esquemas, que denominamos P_0. Para cada esquema, uma porcentagem das posições recebe aleatoriamente as etiquetas (“labels”) 2 e exatamente *p* posições aleatórias recebem as etiquetas 1. As posições

restantes recebem a etiqueta #. Para os testes computacionais da seção 4.6, 20% das posições receberam a etiqueta 2, e n foi o tamanho considerado para a população P_0. Poderíamos ter como população inicial, para o exemplo de 3 medianas, as seguintes estruturas (n=10):

s_1 = (1,2,#,1,1,#,2,#,#,#), s_2 = (#,1,2,2,#,1,#,#,#,1), s_3 = (#,#,1,2,#,1,1,#,#,2),

s_4 = (2,#,#,1,2,#,#,1,#,1), s_5 = (1,#,2,#,#,1,2,1,#,#), s_6 = (#,2,#,#,1,1,1,2,#,#),

s_7 = (#,#,#,2,1,2,1,#,#,1), s_8 = (#,1,2,2,#,#,1,1,#,#), s_9 = (2,2,#,#,#,1,#,1,1,#),

s_{10} = (2,1,2,1,#,#,#,1,#,#).

Os esquemas da população P_0 irão se recombinar para produzir estruturas filhas, sendo que algumas etiquetas #, serão substituídas por etiquetas 1 ou 2, na tentativa de buscar estruturas completas. No caso de uma etiqueta # ser substituída por 1, alguma posição original com etiqueta 1, precisa ser modificada para que o número pré-fixado de medianas permaneça. Esta condição poderia ser relaxada, mas não foi considerada nesta aplicação.

4.3 – RECOMBINAÇÃO

As estruturas (completas ou esquemas) da população P_α são ordenadas de forma não crescente, usando a seguinte chave:

$$\Delta(s_k) = \frac{1 + d_k}{|V_1(s_k)| + |V_2(s_k)|} \tag{4.6}$$

As estruturas completas ($V_3(s_k)=\varnothing$), esquemas com $|V_3(s_k)|$ pequenos, e estruturas apresentando pequenos d_k são melhores e aparecem nas primeiras posições.

Duas estruturas são selecionadas para recombinação. A primeira estrutura é denominada *base* (s_{base}) e é selecionada aleatoriamente das n primeiras posições de P_α. Geralmente a base é uma *boa* estrutura completa ou um bom esquema. Se for um esquema, a recombinação procurará preservar as etiquetas (“labels”) 1 ou 2 já atribuídas a s_{base}. A segunda estrutura é denominada *guia* (s_{guia}) é obtida através de uma seleção na população inteira. O objetivo da estrutura s_{guia} é orientar as modificações na estrutura

s_{base}. As duas estruturas, s_{base} e s_{guia}, são comparadas em suas posições correspondentes e uma (ou mais) estrutura(s) s_{nova} (filha) é (são) obtida(s) após a recombinação. Os seguintes operadores são possíveis:

1. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = \#$ e $s_{guia\,j} = \#$, temos $s_{nova\,j} = \#$;
2. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = 1$ e $s_{guia\,j} = 1$ temos $s_{nova\,j} = 1$;
3. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = 2$ e $s_{guia\,j} = 2$ temos $s_{nova\,j} = 2$;
4. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = 1$ e $s_{guia\,j} = \#$ temos $s_{nova\,j} = 1$;
5. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = 2$ e $s_{guia\,j} = \#$ temos $s_{nova\,j} = 2$;
6. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = \#$ e $s_{guia\,j} = 2$ temos $s_{nova\,j} = 2$;
7. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = \#$ ou 2 e $s_{guia\,j} = 1$ então duas situações são possíveis:

- temos $s_{nova\,j} = 1$ e $s_{nova\,l} = 2$ para um $l \in \{1,...,n\}$ apresentando $s_{base\,l} = 1$ (selecionado aleatoriamente),
- temos $s_{nova\,j} = 1$ e $s_{nova\,l} = 2$ para cada $l \in \{1,...,n\}$ apresentando $s_{base\,l} = 1$, gerando p novas estruturas;

8. Para cada $j \in \{1,...,n\}$ apresentando $s_{base\,j} = 1$ e $s_{guia\,j} = 2$ quando dois casos são possíveis:

- temos $s_{nova\,j} = 2$ e $s_{nova\,l} = 1$ para um $l \in \{1,...,n\}$ apresentando $s_{base\,l} = 2$ (selecionado aleatoriamente),
- temos $s_{nova\,j} = 2$ e $s_{nova\,l} = 1$ para cada $l \in \{1,...,n\}$ apresentando $s_{base\,l} = 2$, gerando $|V_2(s_{base})|$ novas estruturas.

4.4 – MUTAÇÃO

Quando s_{base} é uma estrutura completa ($V_3(s_k)=\varnothing$), a seguinte heurística de permutação é realizada como uma forma de mutação:

HP
{Heurística de permutação}

Para cada $j \in V_1(s_{base})$ **fazer**
    Para cada $i \in V_2(s_{base})$ **fazer**
        Permutar j e i gerando uma estrutura filha s_{nova}; {geração de filhos}
        Permutar i e j;
    Fim
Fim

Algoritmo 4.1: Algoritmo de mutação para o problema das p-medianas.

4.5 – FUNÇÕES DE AVALIAÇÃO

Considere uma estrutura (completa ou esquema) $s_k \in P_\alpha$, então a função g é definida como:

$$g(s_k) = \sum_{i \in V_2(s_k)} \min_{j \in V_1(s_k)} \{\mu_{ij}\} \tag{4.7}$$

Os vértices presentes em s_k, que não são medianas, são atribuídos a mediana mais próxima. Após esta atribuição, existe a formação de p agrupamentos ("clusters"). O conjunto de agrupamentos são formados pelas medianas e os vértices que a elas são atribuídas. Considere $C_j(s_k)$ o agrupamento formado pela mediana j e pelo conjunto de vértices correspondentes (não medianas), sendo $|C_j(s_k)|$ é o número de vértices no conjunto $C_j(s_k)$. O custo (distância) mínimo de atribuição no agrupamento j é:

$$\lambda_j = \min_{i \in V_2(s_k)} \{\mu_{ij}\},\ j \in V_1(s_k) \tag{4.8}$$

A função f é definida como:

$$f(s_k) = \sum_{j=1}^{p} \lambda_j . [\,|C_j(s_k)| - 1] \tag{4.9}$$

Claramente $f(s_k) \leq g(s_k)$, para todas $s_k \in P_\alpha$. Idealmente a diferença $g(s_k)$-$f(s_k)$ precisa ser a menor possível.

O limite superior g_{max} é obtido no início do algoritmo, através da geração de uma estrutura completamente aleatória e usando a mesma avaliação de g. Como a estrutura é gerada aleatoriamente, em geral não consegue uma boa avaliação. Para assegurar que g_{max} seja sempre um limite superior, após a recombinação, cada nova estrutura gerada, s_{nova}, é rejeitada se $g_{max} \leq g_{nova}$.

4.6 – TESTES COMPUTACIONAIS

O AGC foi inicialmente testado para o problema das *p*-medianas, usando instâncias da biblioteca OR ("OR-library") (Beasley,1990). O tamanho das instâncias são de 100, 200, 300, 400 e 500 vértices, sendo que o número de medianas varia de 5 a 67.

O algoritmo foi codificado na linguagem de programação C e executado num computador com processador Pentium 166 Hz.

Os resultados resumidos podem ser vistos na tabela 4.1. A solução AGC é a melhor avaliação de $g(s_k)$ durante a busca. O *diferença* é dada por:

$$diferença = \frac{(solução\ AGC - solução\ ótima) * 100}{solução\ AGC} \tag{4.10}$$

O controle de evolução usado foi: ε=0.05 para $0 \leq \alpha \leq 1$ e ε=0.025 para $\alpha > 1$. E o desvio absoluto admitido é d = 0.1. Estes valores para os parâmetros se mostraram adequados aos testes, mas podem ser modificados para, por exemplo, acelerar o método, ou ao contrário, procurar soluções ainda melhores, com maior tempo computacional.

Podemos observar na tabela que toda as *diferenças* são inferior a 0.73% e para nove instâncias a solução ótima foi obtida. Vemos também, que o tempo computacional aumenta para um número maior de medianas. Isto pode ser facilmente explicado, pela busca local (mutação) que é mais demorada nestes casos. Este fato não nos causa surpresa, pois também ocorre com outros métodos heurísticos relatados na literatura.

A média da *diferença* em relação a solução ótima foi de 0,10, e o pequeno intervalo de tempo computacional necessário, mostra que o AGC foi eficiente na otimização dos problemas testados.

Tabela 4.1: Resultados para as instâncias da biblioteca OR.

Problema	Vértices	Medianas	Solução ótima	CGA	Diferença (%)	Tempo (s)
pmed1	100	5	5819	5819	0	28
pmed2	100	10	4093	4093	0	37
pmed3	100	10	4250	4250	0	34
pmed4	100	20	3034	3034	0	230
pmed5	100	33	1355	1360	0.36	375
pmed6	200	5	7824	7824	0	172
pmed7	200	10	5631	5631	0	238
pmed8	200	20	4445	4454	0.20	1055
pmed9	200	40	2734	2754	0.73	3331
pmed10	200	67	1255	1257	0.15	4325
pmed11	300	5	7696	7696	0	369
pmed12	300	10	6634	6637	0.04	677
pmed16	400	5	8162	8162	0	555
pmed21	500	5	9138	9138	0	1875

Nos gráficos seguintes mostramos algumas das características do AGC, usando o problema *pmed1*. A figura 4.1 mostra a evolução do tamanho da população por geração. O número máximo no tamanho da população é obtida na geração 25 ($P_{1.25}$) com aproximadamente 2000 estruturas. A população diminui após α=1.25, mostrando que o número de novas estruturas por geração é menor que o número de estruturas que não sobrevivem ao teste de evolução. O procedimento pára quando a população torna-se vazia, em α=4.5, ou seja, após 90 gerações.

A figura 4.2 mostra o número máximo de vértices participantes para cada estrutura e que foram obtidas através da recombinação e busca local. Somente após 13

gerações, uma estrutura completa foi obtida. Mesmo após obter uma população com estruturas completas, uma parte representativa da população é formada por esquemas, o que nos faz relembrar que um esquema pode receber uma avaliação-*fg* melhor do que uma estrutura completa, em alguns casos.

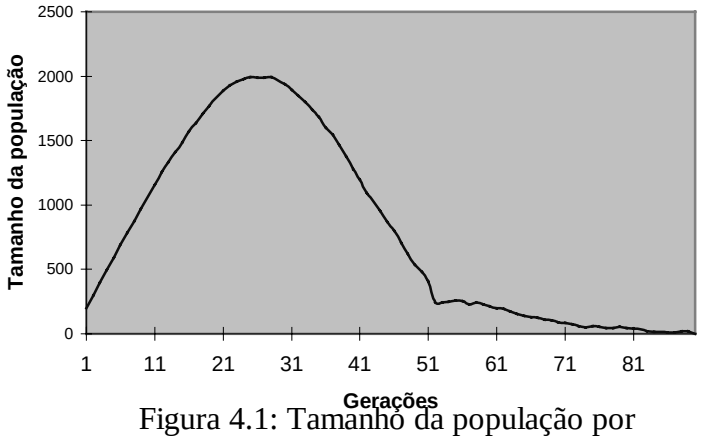


Figura 4.1: Tamanho da população por geração.

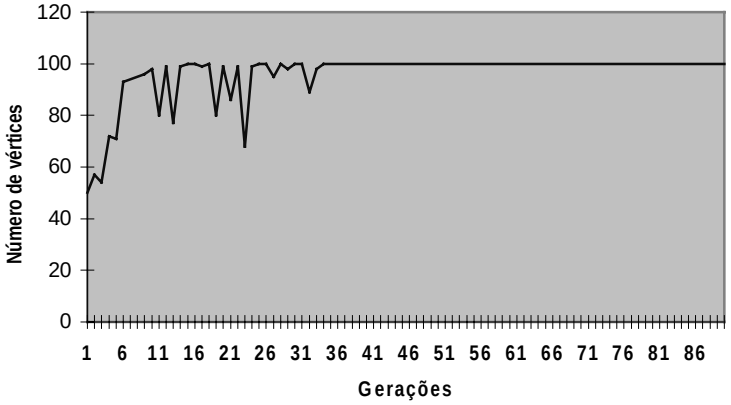


Figura 4.2: Número máximo de vértices por geração.

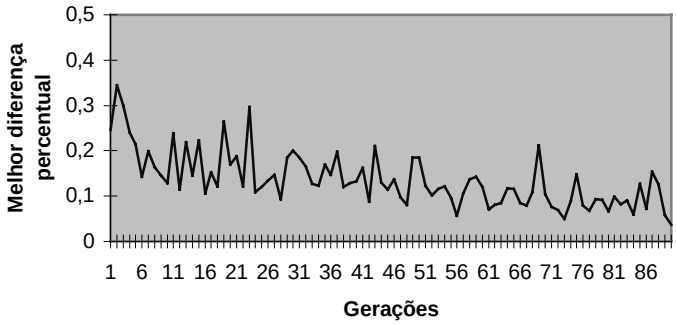


Figura 4.3: Melhor d_k por geração.

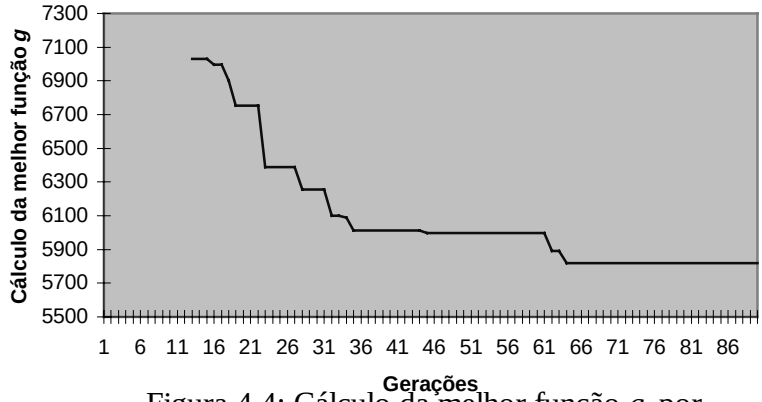


Figura 4.4: Cálculo da melhor função g por geração.

A figura 4.3 mostra como d_k é melhorado através das gerações e finalmente a figura 4.4 mostra a melhor avaliação g (a solução para o problema das p-medianas) para uma estrutura completa, a cada geração.

No apêndice A, mostramos um conjunto de tabelas para os problemas da tabela 4.1, em que as soluções ótimas foram obtidas. Nas tabelas, as medianas e as respectivas atribuições são exibidas.

Como o conjunto de instâncias da biblioteca OR pode ser considerado composto por problemas fáceis, pois suas soluções ótimas podem ser obtidas diretamente usando relaxação Lagrangeana (Senne e Lorena, 1997; Beasley, 1993), nós decidimos usar o AGC no conjunto de instâncias de Galvão *et al.*, 1996, que apresentam

"gap" de dualidade para alguns valores de *p*. Nós também realizamos uma comparação com a heurística de permutação HP, usada diretamente num conjunto de estruturas.

Na tabela 4.2 mostramos os resultados computacionais. Para usar a HP sozinha, iniciamos com um conjunto de estruturas completas de tamanho *n*. Então a HP foi aplicada a cada estrutura e repetimos o processo com 5 diferentes conjuntos de estruturas geradas aleatoriamente. O uso direto da HP produz diferenças que variam de 3.57% a 14.88%, sendo que a média das diferenças é 7,9%.

Tabela 4.2: Resultados para instâncias de Galvão *et al.* 1996.

Núme-ro de vértices	Número de media-nas	*"gap"* Galvão *et al.(1996)* (%)	Solução ótima	Solução AGC	Diferen-ça AGC (%)	Solu-ção HP	Diferen-ça HP(%)
100	5	0.346	5703	5703.00	0	6065.6	6.36
100	10	3.728	4426	4426.00	0	4900.6	10.72
100	15	0.895	3893	3898.33	0.136	4212.3	8.20
100	20	0.093	3565	3575.67	0.299	3781.6	6.07
100	25	0.067	3291	3299.00	0.243	3426.0	4.10
100	30	0.056	3032	3044.90	0.425	3145.0	3.73
100	40	0	2542	2553.00	0.432	2632.6	3.57
100	50	0	2083	2085.00	0.096	2168.6	4.11
150	5	1.404	10839	10839.00	0	11298.	4.24
150	10	3.158	8729	8729.00	0	9446.0	8.21
150	15	4.906	7390	7403.67	0.184	8287.6	12.15
150	20	2.975	6454	6489.33	0.547	7414.6	14.88
150	25	1.009	5875	5887.00	0.204	6650.6	13.20
150	30	0.208	5495	5514.00	0.345	6187.0	12.59
150	40	0.068	4907	4924.00	0.346	5319.3	8.40
150	50	0.062	4374	4380.00	0.137	4647.0	6.24

Os resultados obtidos por Galvão *et al.*, 1996, apresentam *"gap"* de dualidade, obtido através da aplicação às instâncias do software CPLEX e uma relaxação Lagrangeana.

As *diferenças* para o AGC variaram de 0% a 0,54% e a média das diferenças foi 0,21%. Isto demonstra que o AGC consegue obter soluções de alta qualidade, mesmo quando existe um *gap* de dualidade.

Também podemos comparar os resultados do algoritmo HP e o AGC. Visivelmente os resultados do AGC são melhores. O que demonstra que a mutação sozinha não é capaz de boas soluções, e evidência a importância do processo construtivo na geração de estruturas completas.

5 – PROBLEMA DE AGRUPAMENTO CAPACITADO

Neste capítulo usamos o AGC para obter soluções ao problema de agrupamento capacitado (“capacitated clustering problem” – CCP). O CCP é o problema no qual, dado um conjunto de objetos com diferentes pesos, deseja-se particionar este conjunto em diferentes agrupamentos, de tal forma que o peso total dos objetos em cada agrupamento seja menor ou igual a um dado valor. O objetivo é minimizar a dispersão total dos objetos em relação a um centro do agrupamento ao qual foram atribuídos. Veja figura 5.1.

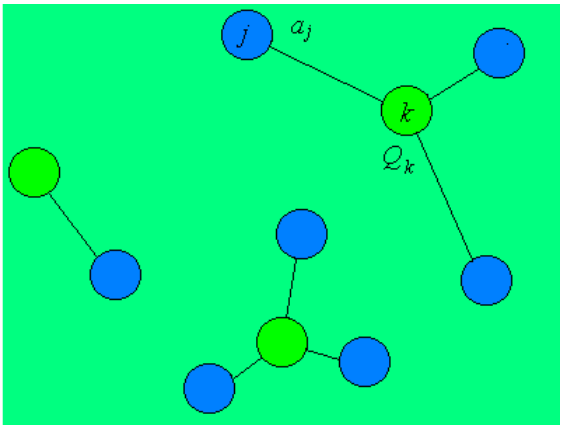


Figura 5.1: O problema de agrupamento capacitado.

5.1 – O PROBLEMA

O problema de agrupamento capacitado, que é NP-completo (Garey e Johnson, 1979) pode ser formalizado conforme segue: Dado um conjunto de n objetos ou clientes (vértices), $V = \{1, ...,n\}$, com uma matriz nxn de custo, μ_{kl}, indicando a distância entre um par de vértices de V. Assumimos que $\mu_{kl} \geq 0$, $\mu_{kk}=0$ e $\mu_{kl}=\mu_{lk}$ para todo $k,l \in V$. Para cada vértice (cliente) k existe uma demanda positiva a_k. O conjunto V pode ser particionado em $\Gamma=\{C_1,C_2,...,C_p\}$ agrupamentos para um dado p inteiro, $2 \leq p < n$, e $I=\{1,...,p\}$ o conjunto dos índices dos agrupamentos. Considere $V_1 =\{\zeta_1,...,\zeta_p\}$ como o conjunto de centros de Γ, onde $V_1 \subseteq V$ e ζ_i é o centro (vértice) do qual a soma de suas distâncias a todos os outros vértices no agrupamento C_i é minimizada. Considere também que Q_i denota a capacidade do agrupamento C_i e assumimos que $Q_i = Q, \ \forall i \in I$. Observe que a igualdade assumida não é estritamente necessária e poderia ser relaxada, sem qualquer perda de generalidade.

Uma solução genérica possível para CCP pode ser representada por:

$$V = \bigcup_{i=1}^{p} C_i ; C_i \cap C_j = \varnothing \ \forall i, j \in I \ e \ i \neq j; \tag{5.1}$$

$$\sum_{j \in C_i} a_j \leq Q_i \quad \forall i \in I; \tag{5.2}$$

$$Z(\Gamma) = \sum_{i=1}^{p} \left(\sum_{j \in C_i} \mu_{j\zeta_i} \right), \tag{5.3}$$

Onde $Z(\Gamma)$ é o valor da função objetivo da solução $\Gamma=\{C_1,...,C_p\}$.

Uma formulação para o problema de agrupamento não-capacitado (“uncapacitated clustered problem” – UCP) pode ser encontrada em Vinod (1969); Rao (1971); Mulvey e Crowder (1979). Uma abordagem usando algoritmo “branch and bound” foi sugerida por Jarvinen *et al.* (1972); Koontz *et al.* (1975); Christofides and Beasley (1982); klein and Aronson (1991) e existem várias heurísticas para resolver o problema (Teitz and Bart, 1968; Whitaker, 1983; Golden and Skiscim, 1986; Rahman and Smith, 1991).

Mulvey and Beck (1984) foram os primeiros a estender de UCP para CCP, adicionando restrições de capacidade a cada agrupamento. Eles desenvolveram um heurística primal e um método híbrido heurística-subgradiente para obter boas soluções para o CCP. A heurística primal inicia com um conjunto aleatório de *p* centros e atribui clientes aos seus centros mais próximos, em ordem decrescente de valores previamente definidos. Estes valores são definidos como o valor absoluto da diferença entre a distância do primeiro e segundo centros. Após todas atribuições serem completadas, o centro para cada agrupamento é recalculado. Se um ou mais novos centros são obtidos e a função objetivo diminui para um valor pré-estabelecido, o processo é repetido, reatribuindo os clientes aos novos centros. O processo continua até que não sejam obtidos novos centros de uma iteração à outra. A heurística-subgradiente é repetida para um pré-determinado número de iterações. A cada iteração, os centros dos agrupamentos são obtidos resolvendo um problema de relaxação que exclui as restrições de atribuição. Então um procedimento heurístico procura uma possível atribuição dos clientes aos centros. A qualidade das soluções é melhorada, usando-se ainda, um procedimento de permutação de clientes entre agrupamentos.

Osman e Christofides (1994) propuseram diferentes métodos heurísticos: uma heurística construtiva simples, um mecanismo de λ-permutações, uma implementação híbrida, usando busca tabu e “simulated annealing” e realizaram a comparações entre os métodos.

A seguir, veremos como o problema foi tratado usando o Algoritmo Genético Construtivo.

5.2 – REPRESENTAÇÃO

A representação é a mesma usada para o problema das *p*-medianas. Vejamos um exemplo de representação. A cadeia s_k=(2 1 # 2 2 2 1 2 2 1 2 2), onde cada posição na cadeia representa um vértice, representa as atribuições mostrados na figura 5.2. O conjunto *V* de vértices pode ser dividido em três: o conjunto V_1 das medianas, o conjunto V_2 dos vértices que não são medianas e o conjunto $V_3=V-(V_1(s_k)UV_2(s_k))$ de

curingas. No exemplo, V_1={2,7,10}, V_2={1,4,5,6,8,9,11,12} e V_3={3}. Vemos na figura 5.2, que existe uma tabela com as respectivas demandas para cada cliente (vértice) e outra tabela mostrando a demanda total obtida por agrupamento. Considerando que a demanda (capacidade) máxima por agrupamento é 100 unidades, vemos que embora o vértice 4 esteja mais próximo do agrupamento com centro no vértice 10, não foi atribuído a este agrupamento para evitar que a capacidade fosse excedida. O vértice 4 foi atribuído a mediana 2, pois se constituía na melhor atribuição possível (que respeitava a capacidade máxima do agrupamento). Percebemos, então, que a representação usada requer uma heurística (regra) de transição, para que mostre as atribuições dos vértices às medianas correspondentes.

A heurística de transição, *HT1*, apresenta os seguintes passos:

HT1 {Heurística de transição}

Para *i*=1 até *p* f**azer**

$Q_i := Q$;

$C_i = \{\zeta_i\}$;

Fim_para

Para *j=1 até n* e j$\in V_2$ **fazer**

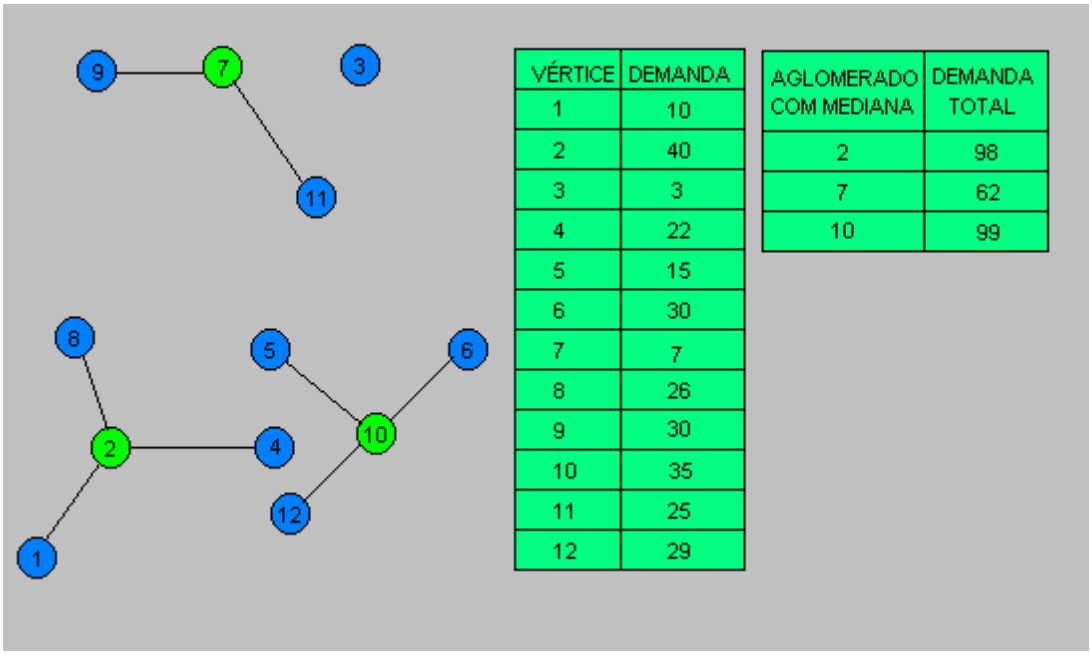
k = índice {mínimo $\mu_{j\zeta i}$ | $Q_i - a_j \geq 0$, *i*=1,...,*p*};

$C_k := C_k \; U \; \{j\}$;

$Q_k := Q_k - a_j$;

Fim_para

Algoritmo 5.1: Heurística de transição para o CCP.



VÉRTICE	DEMANDA
1	10
2	40
3	3
4	22
5	15
6	30
7	7
8	26
9	30
10	35
11	25
12	29

AGLOMERADO COM MEDIANA	DEMANDA TOTAL
2	98
7	62
10	99

Figura 5.2: Representação gráfica para o problema de agrupamento capacitado.

5.3 – POPULAÇÃO INICIAL

A população inicial, P_o, é formada de esquemas que recebem exatamente p etiquetas com número 1 em posições aleatórias e uma porcentagem das posições recebe aleatoriamente etiquetas com número 2, e as posições restantes recebem #. Foram empregados os mesmos critérios usados para o problema das p-medianas (seção 4.2).

5.4 – RECOMBINAÇÃO

Também como no problema das p-medianas, as estruturas da população P_α estão numa ordem não-crescente, usando a seguinte chave:

$$\Delta(s_k) = \frac{1 + d_k}{|V_1(s_k)| + |V_2(s_k)|} \tag{5.4}$$

As estruturas completas ($V_3(s_k)=\varnothing$), esquemas com $|V_3(s_k)|$ pequenos, e estruturas apresentando pequenos d_k são melhores e aparecem nas primeiras posições.

A recombinação ocorre entre uma estrutura *base* e uma estrutura *guia*, exatamente como foi proposto no problema das *p*-medianas. Uma dúvida que poderia surgir, refere-se ao limite de capacidade dos agrupamentos. No entanto, não ocorre qualquer problema nesta fase, uma vez que o limite de capacidade é tratado na heurística de transição.

5.5 – MUTAÇÃO

Quando s_{base} é uma estrutura completa ($V_3(s_k)=\varnothing$), usa-se a heurística de permutação HP apresentada em 4.4. O mesmo comentário do ítem 5.4 se aplica aqui, quanto ao atendimento das capacidades.

5.6 – FUNÇÕES DE AVALIAÇÃO

Considere uma estrutura $s_k \in P_\alpha$, então usando a heurística de transição HT1 a função *g* pode ser definida como:

$$g(s_k) = \sum_{i=1}^{p} \sum_{j \in C_i(s_k)} \mu_{j\zeta_i} \tag{5.5}$$

e a função *f* pode ser definida:

$$f(s_k) = \sum_{j=1}^{p} \lambda_j . [\,|\, C_j(s_k) \,|\, -1] \tag{5.6}$$

onde:

$$\lambda_j = \min_{i \in V_2(s_k)} \{\mu_{ij}\},\ j \in V_1(s_k) \tag{5.7}$$

Claramente $f(s_k) \leq g(s_k)$, para todas $s_k \in P_\alpha$. Idealmente a diferença $g(s_k)-f(s_k)$ precisa ser a menor possível.

O limite superior g_{max} é obtido no início do algoritmo, através da geração de uma estrutura completamente aleatória e usando a mesma avaliação de g. Como a estrutura é gerada aleatoriamente, em geral não consegue uma boa avaliação. Para assegurar que g_{max} seja sempre um limite superior, após a recombinação, cada nova estrutura gerada, s_{nova}, é rejeitada se $g_{max} \leq g_{nova}$.

5.7 - RESULTADOS COMPUTACIONAIS

Dois conjuntos de problemas testes com $n \in \{50,100\}$ e $p \in \{5,10\}$ do artigo de Osman e Christofides (1994) foram usados. Um conjunto contém 10 problemas de tamanho 50x5 e o outro conjunto contém 10 problemas de tamanho 100x10. Os vértices estão localizados num plano bidimensional e suas coordenadas foram geradas aleatoriamente através de uma distribuição normal no intervalo [1,100]. μ_{kl} é a distância euclidiana entre o vértice k e l. O valor da demanda, a_k, é gerado da distribuição uniforme no intervalo [1,20]. A capacidade Q dos agrupamentos, para um dado problema, é escolhida de tal forma que $\tau \in [0.82, 096]$, onde τ é a razão:

$$\tau = \frac{\sum_{k \in V} a_k}{\sum_{i \in I} Q_i} \tag{5.8}$$

Os problemas testes estão em ordem crescente de τ. Ainda se desconhece as soluções ótimas para estas instâncias, entretanto tem-se as melhores soluções obtidas por Osman e Christofides (1994) para cada problema.

O algoritmo foi codificado na linguagem de programação C e executado num computador com processador Pentium 166 MHz.

Os resultados resumidos do AGC podem ser vistos na tabela 5.1. Nas primeiras colunas da tabela, também são mostrados resultados de outros algoritmos do artigo de Osman e Christofides (1994).

O algoritmo denominado H.OC é uma heurística construtiva proposta para obter uma solução inicial ao problema. Esta heurística apresenta três estágios. No

primeiro estágio é escolhido um conjunto inicial de centros. O segundo estágio consiste em atribuir clientes aos centros encontrados na primeira fase. O estágio final, consiste em recalcular os novos centros, de acordo com as atribuições ocorridas.

O algoritmo H1+FI é uma heurística que inicia com o algoritmo H.OC, seguida por um procedimento de busca local denominado *mecanismo 1-permutação* e uma estratégia de seleção denominada "*first-improve"*. O algoritmo H1+BI apresenta os mesmos passos, apenas substituindo a estratégia de seleção pela denominada "*best-improve"*. A diferença entre as duas estratégias de seleção é a seguinte: No "*best-improve*" todas as soluções de uma determinada vizinhança são examinadas e a melhor, que satisfaz um critério de aceitação, é escolhida. No *"first-improve"* a primeira solução que satisfaz um critério de aceitação é escolhida.

Os algoritmos HSS.OC e HSS.C usam a heurística "simulated annealing - SA". A diferença das duas formas, está no procedimento de *resfriamento* da temperatura do SA. O algoritmo TS1+FBA, usa a busca tabu com estratégia de seleção que admite o primeiro *movimento* que melhora a qualidade da solução corrente, ou que escolhe o primeiro movimento que menos prejudica a qualidade da solução, quando não existir movimento melhor na vizinhança.

Os valores da tabela para o AGC foram obtidos executando 5 vezes o programa e calculando-se a sua média. Na tabela 5.2, são mostrados as diferenças percentuais em relação a melhor solução conhecida, ou seja:

$$\text{Diferença}\,(\%) = \frac{(\text{solução da heurística} - \text{melhor solução conhecida}) * 100}{\text{solução da heurística}} \qquad (5.9)$$

O controle de evolução usado foi: ε=0.05 para $0 \leq \alpha \leq 1$ e ε=0.025 para $\alpha > 1$. E o desvio absoluto admitido é d = 0.1.

Tabela 5.1: Resultados da função objetivo do AGC e de outros métodos.

Problema	Vértices	Medianas	Melhor solução	H.OC	H1+FI	H1+BI	HSS.OC	HSS.C	TS1+FBA	AGC
1	50	5	713	786	780	818	713	734	734	713
2	50	5	740	816	762	778	740	740	740	740
3	50	5	751	972	811	816	751	751	751	751
4	50	5	651	891	651	652	651	651	651	651
5	50	5	664	804	746	677	664	664	664	664
6	50	5	778	882	841	847	778	778	778	778
7	50	5	787	968	852	824	787	805	787	787
8	50	5	820	945	834	837	820	820	821	826
9	50	5	715	752	735	734	715	715	715	715
10	50	5	829	1017	844	891	829	829	829	834
11	100	10	1006	1761	1020	1019	1006	1006	1009	1014
12	100	10	966	1567	1004	974	966	966	968	969
13	100	10	1026	1847	1144	1053	1026	1026	1026	1026
14	100	10	982	1635	998	1054	985	982	985	987
15	100	10	1091	1517	1098	1138	1091	1091	1096	1091
16	100	10	954	1780	1063	993	954	954	957	955
17	100	10	1034	1665	1104	1092	1039	1037	1040	1034
18	100	10	1043	1345	1089	1136	1045	1045	1045	1045
19	100	10	1031	1634	1105	1125	1031	1032	1034	1032
20	100	10	1005	1872	1036	1030	1005	1019	1005	1039

Na figura 5.3 mostramos um gráfico comparando os resultados do AGC com a melhor solução conhecida. Podemos observar que para 11 instâncias a melhor solução conhecida foi obtida. Para as demais instâncias, as diferenças variaram de 0,09% a 3,38%. A média das diferenças foi 0,37%.

Tabela 5.2: Diferenças percentuais em relação a melhor solução.

Problema	H.OC	H1+FI	H1+BI	HSS.OC	HSS.C	TS1+FBA	AGC	Tempo (s)
1	10,23	9,39	14,72	0	2,94	2,94	0	2
2	10,27	2,97	5,13	0	0	0	0	2
3	29,42	7,98	8,65	0	0	0	0	12
4	36,86	0	0,15	0	0	0	0	2
5	21,08	12,34	1,95	0	0	0	0	8
6	13,36	8,09	8,86	0	0	0	0	2
7	22,99	8,25	4,70	0	2,28	0	0	3
8	15,24	1,70	2,07	0	0	0,12	0,73	11
9	5,17	2,79	2,65	0	0	0	0	2
10	22,67	1,80	7,47	0	0	0	1,44	14
11	75,04	1,39	1,29	0	0	0,29	0,79	614
12	62,21	3,93	0,82	0	0	0,20	0,31	296
13	80,01	11,50	2,63	0	0	0	0	327
14	66,49	1,62	7,33	0,30	0	0,30	0,50	303
15	38,91	0,54	4,21	0	0	0,36	0	315
16	86,58	11,42	4,08	0	0	0,31	0,10	271
17	61,02	6,76	5,60	0,48	0,29	0,58	0	332
18	28,95	4,41	8,91	0,19	0,19	0,19	0,19	380
19	58,48	7,17	9,11	0	0,09	0,29	0,09	410
20	86,26	3,08	2,48	0	1,39	0	3,38	503

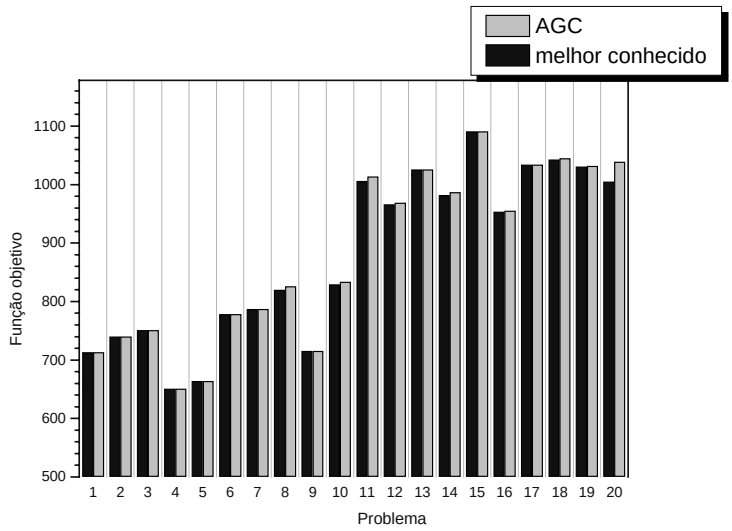


Figura 5.3: Gráfico comparando o AGC com a melhor solução conhecida.

No apêndice B são mostradas as figuras das soluções dos problemas testes. Como os resultados mostrados nas tabelas 5.1 e 5.2 são oriundos da média da execução de 5 vezes do AGC, mostramos apenas as figuras nas quais sempre a melhor solução conhecida foi obtida, pois, em geral, nos outros problemas, a solução variava de execução para execução.

6 – PROBLEMA CAPACITADO DE PARTICIONAMENTO DE GRAFOS

Usamos também o AGC no problema capacitado de particionamento de grafos - PCPG. O problema consiste em particionar os vértices de um grafo em *k* agrupamentos, tal que a soma dos pesos dos vértices de cada agrupamento tenha um limite inferior e superior, enquanto maximiza (minimiza) a soma dos custos das arestas dentro (fora) de cada agrupamento. Vamos ver na seção seguinte a formalização do problema.

6.1 - O PROBLEMA

Dado um grafo não direcionado *G(V,E)* com pesos não negativos a_k, $k \in V$, $V=\{1,2,...,n\}$, arestas de custo μ_e, $e \in E$, e um inteiro *p*, o problema capacitado de particionamento de grafos é obter um particionamento $\Gamma=\{C_1, C_2,...,C_p\}$ de *V* que resolve:

$$Max \sum_{i=1}^{p} \sum_{e \in E(C_i)} \mu_e \tag{6.1}$$

$$Q_{\min} \leq \sum_{u \in C_m} a_u \leq Q_{\max}, m = 1,2,..., p, \tag{6.2}$$

onde $E(C_i)=\{(i,j)\in E:i,j\in C_i\}$. Nós nos referimos a C_i, $i=1,...,p$, como agrupamentos correspondentes ao particionamento de Γ de V. Em outras palavras, o problema consiste em particionar os vértices de um grafo em p agrupamentos, de tal forma que a soma dos pesos dos vértices de cada agrupamento tenha um limite inferior dado por Q_{min} e um limite superior dado por Q_{max}, enquanto maximiza a soma dos custos das arestas no interior dos agrupamentos.

Johnson *et al.* (1992) apresentaram uma formulação de programação inteira ao PCPG, para uma aplicação na construção de compiladores. Um compilador consiste de diversos módulos, onde cada módulo é formado por um conjunto de procedimentos e subrotinas com suas necessidades de memória associadas. Os módulos são combinados para formar agrupamentos, que possuem um limite de memória disponível. Limites típicos para cada agrupamento são 450K e 512K. O custo de comunicação entre módulos do mesmo agrupamento é muito pequeno e pode ser ignorado e o custo de comunicação entre módulos de agrupamentos diferentes é significativo, pois pode existir a necessidade de permutação de áreas de memória. O objetivo na construção de compiladores é atribuir módulos a agrupamentos, de tal forma que o limite de memória seja satisfeito e o custo de comunicação total entre os diferentes módulos seja minimizado. No PCPG, os módulos são representados por vértices e a memória necessária por cada módulo são representadas pelos pesos. O custo de comunicação é representado pelo custo das arestas do grafo.

Nós assumimos que o número de agrupamentos a serem formados é p e que o limite inferior dos pesos de cada agrupamento é 0, enquanto o limite superior dos pesos é igual a Q.

6.2 – REPRESENTAÇÃO

Assim como nos outros problemas, o primeiro passo consiste em obter uma representação adequada. No PCPG uma estrutura pode ser representada por uma cadeia. Cada posição na cadeia representa um vértice do grafo do problema e usa as seguintes etiquetas:

- #: vértice que ainda não participa das soluções;
- 1: vértice semente;
- 2: vértice que é atribuído a algum agrupamento formado por um vértice semente.

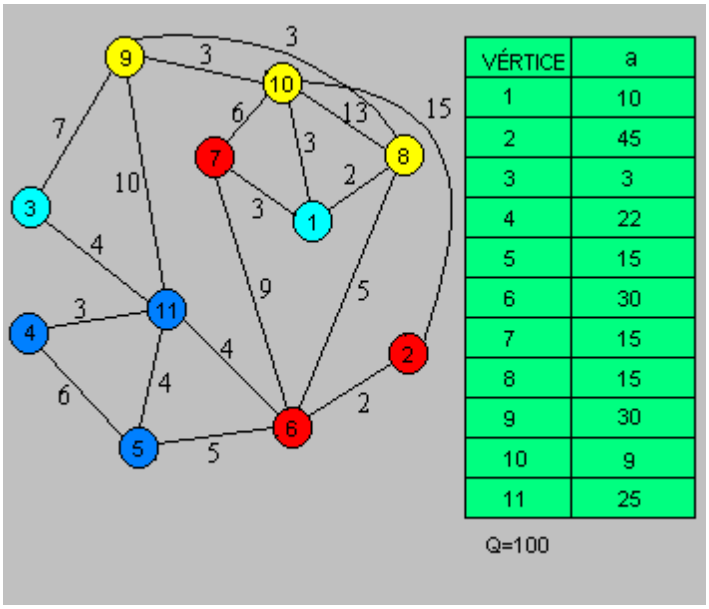


Figura 6.1: Representação da estrutura s=(#2#12122122).

Assim, o conjunto de vértices V pode ser dividido em três subconjuntos: o conjunto V_1 de vértices sementes, o conjunto V_2 de vértices que estão atribuídos a algum agrupamento e o conjunto V_3 dos vértices que ainda não fazem parte da solução.

Por exemplo, a cadeia s_k=(#,2,#,1,2,1,2,2,1,2,2) apresenta os subconjuntos: V_1={4,6,9}, V_2={2,5,7,8,10,11} e V_3={1,3}. Observamos que sempre teremos $p=|V_1|$, ou seja, o número de vértices sementes corresponde ao número de agrupamentos que desejamos formar. Assim denotamos $V_1=\{\zeta_1,...,\zeta_p\}$. Na figura 6.1 mostramos um grafo com 11 vértices e com as respectivos pesos das arestas (uma solução mostra os vértices estão agrupados conforme sua cor). Após a identificação dos vértices sementes é necessário a atribuição dos vértices do conjunto V_2 aos agrupamentos. Esta atribuição está baseada na heurística de transição do algoritmo 6.1.

HT2 {Heurística de transição}

Para i=1 até p **fazer**

$Q_i := Q$;

$C_i = \{\zeta_i\}$;

Fim_para

Para $j=1$ *até* n e $j \in V_2$ **fazer**

k = índice {máximo $\chi_i \mid Q_i - a_j \geq 0$, i=1,...,p};

$$\text{onde: } \chi_i = \frac{\sum_{e \in E(C_i U \{j\})} \mu_e}{|C_i|}$$

$C_k := C_k \, U \, \{j\}$;

$Q_k := Q_k - a_j$;

Fim_para

Algoritmo 6.1: Heurística de transição para o PCPG.

No algoritmo 6.1 podemos observar que a atribuição dos vértices do conjunto V_2 aos agrupamentos depende do peso das arestas e da capacidade dos agrupamentos.

6.3 – POPULAÇÃO INICIAL

A população inicial é formada por $|P_0|$ esquemas com exatamente p etiquetas 1. Estas etiquetas são colocadas em posições aleatórias na cadeia. Um percentual dos demais vértices recebe etiquetas 2 (também em posições aleatórias) e o restante dos vértices recebe etiqueta #.

6.4 – RECOMBINAÇÃO

A recombinação ocorre da mesma forma como para o problema das p-medianas.

6.5 – MUTAÇÃO

Como forma de mutação, percebemos que alterando o vértice semente, podiamos construir diferentes soluções. Portanto, algoritmo de mutação pode ser igual ao dos problemas anteriores.

6.6 – FUNÇÕES DE AVALIAÇÃO

Como trata-se de um problema de maximização, a função *g* passa a ser um limite superior na avaliação das estruturas. Desta forma, podemos definir a função *g* como:

$$g = \sum_{i=1}^{p} \lambda_i \mid E(C_i) \mid \tag{6.11}$$

onde:

$$\lambda_i = max\, \mu_e, e \in E(C_i) \tag{6.12}$$

A função *f* pode ser definida como:

$$f = \sum_{i=1}^{p} \sum_{e \in E(C_i)} \mu_e \tag{6.13}$$

E o limite superior, $g_{max,}$ pode ser definido como:

$$g_{\max} = \lambda \mid E \mid \tag{6.14}$$

onde: $\lambda = max\, \mu_e, e \in E$ e $|E|$ = número de arestas do grafo.

6.7 – TESTES COMPUTACIONAIS

O AGC foi implementado para instâncias de (Johnson *et al.*, 1992). Existem dois conjuntos de instâncias. O primeiro com Q=450 e o segundo com Q=512.

O algoritmo foi codificado na linguagem de programação C e executado num computador com processador Pentium 166 MHz. Os resultados resumidos, podem ser vistos na tabela 6.1.

Tabela 6.1: Resultados para o algoritmo genético construtivo.

Problema		Q	Vértices	p	Arestas	Solução ótima	AGC	Diferença (%)	Tempo (s)
1	Cb450.30.6.47	450	30	6	47	1099	1099	0	20
2	Cb450.48.8.98	450	48	8	98	2928	2928	0	31
3	Cb450.47.8.99	450	47	8	99	1837	1826	0,59	86
4	Cb450.47.9.101	450	47	9	101	3574	3451	3,44	210
5	Cb450.61.9.187	450	61	9	187	22245	21459	3,53	118
6	Cb512.30.5.47	512	30	5	47	1174	1142	2,72	63
7	Cb512.45.7.98	512	45	7	98	3238	3209	0,89	49
8	Cb512.47.7.99	512	47	7	99	1993	1940	2,65	79
9	Cb512.47.8.101	512	47	8	101	3969	3713	6,40	322
10	Cb512.61.8.187	512	61	8	187	23564	22762	3,40	359

Para dois problemas o AGC conseguiu obter as soluções ótimas. Nas outras 8 instâncias , as diferenças em relação ao ótimo variaram de 0,59 à 6,40. Na figura 6.2 mostramos um gráfico comparativo entre as soluções ótimas e as solucões obtidas com o AGC.

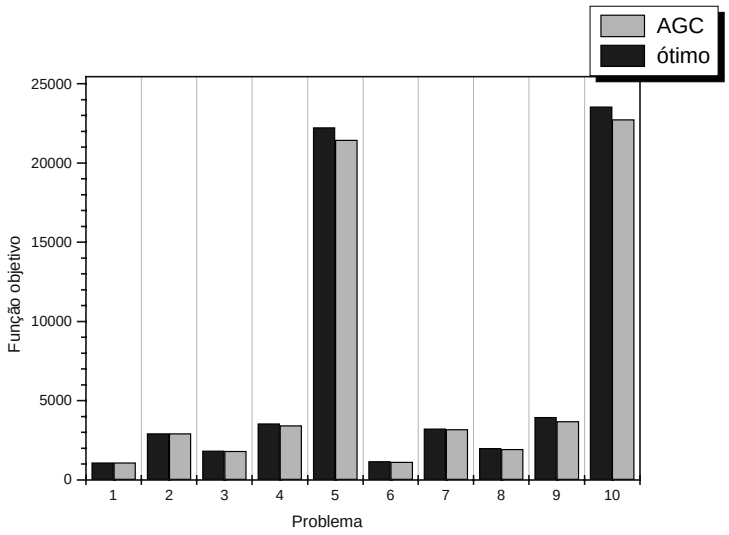


Figura 6.2: Gráfico comparando os resultados do AGC com as soluções ótimas.

7 – CONSIDERAÇÕES FINAIS E CONCLUSÕES

7.1 – CONCLUSÕES

Nesta tese apresentamos uma nova heurística de otimização, que usa alguns conceitos do algoritmo genético e do algoritmo A*. O AGC apresenta-se como um método promissor na busca de soluções a problemas de otimização combinatória e assim como o AGT, faz parte de um movimento na ciência da computação que busca inspiração biológica para a resolução de problemas.

Quando o algoritmo genético foi proposto por Holland 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, também compartilhamos desta idéia e criamos uma forma de avaliar diretamente esquemas, o que ainda não tinha sido realizado com sucesso. Para a avaliação são usadas duas funções de forma semelhante ao algoritmo A*.

Para as três aplicações desta tese, as funções *f* e *g* foram definidas de forma a tornar uma das funções igual a avaliação da função objetivo, e a outra um limite (superior ou inferior). Desta forma, para uma estrutura completa, estamos minimizando os "*gaps*" entre a relaxação e solução. Isso porém nem sempre será necessário, como no

caso do problema de coloração de grafos. Neste problema, para uma estrutura completa, $f=g$, quando uma solução é obtida (veja equações 7.1 e 7.2).

Podemos observar que nos três problemas de agrupamentos examinados nesta tese, usamos a mesma representação, ou seja, os símbolos:1,2 e #. Concluimos que, para outros problemas de agrupamento, isto sempre será possível, bastando apenas, obter uma heurística de transição adequada.

A heurística de transição é importante, porque permite uma busca local implícita e garante que as estruturas, mesmo com uma representação simples (usando apenas 3 símbolos) sempre satisfaçam as restrições do problema (a menos de incluir toda instância do problema). Outras heurísticas poderiam ter sido usadas nos problemas examinados.

Devido a sua natureza e através da heurística de transição é possível a incorporação de novas restrições aos problemas estudados pelo AGC, sem que haja necessidade de alterações no método. Todas as restrições podem ser tratadas na heurística de transição.

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.

A mutação, inicialmente considerada operador de fundo por Holland (1975), têm-se tornado importante nos recentes trabalhos sobre computação evolutiva. Nesta tese foi implementada como uma forma de busca local para estruturas completas. O fato de existir esta busca local apenas fortalece o AGC, mas não retira o seu mérito, uma vez que quando as estruturas completas são obtidas, estas já representam soluções de alta qualidade. A mutação (busca local) apenas direciona a busca para um ótimo local.

Os resultados obtidos com o AGC para os três problemas de agrupamentos mostraram que o novo método é eficiente na obtenção de boas soluções em intervalos reduzidos de tempo. Na maioria das instâncias analisadas, as soluções ótimas ou estando

a menos de 1% das soluções ótimas foram obtidas. Para algumas instâncias, foi possível obter soluções de qualidade superior aos encontrados por outros métodos propostos para os problemas.

A partir do problema das *p*-medianas, os processos de recombinação, mutação e parâmetros mais convenientes foram obtidos, além de observar a importância da heurística de transição. Desta forma, os dois problemas seguintes tiveram implementação facilitada.

Objetivamos inserir o AGC no Sistema de Informações Geográficas SIG ARC/Info, que é parte integrante do Projeto ARSIG – Análise de Redes com Sistemas de Informações Geográficas (http://www.lac.inpe.br/~lorena/ArsigIndex.html), pois a nova heurística é flexível o suficiente para permitir a incorporação de restrições aos problemas e pela qualidade das soluções obtidas, particularmente no problema das *p*-medianas.

As heurísticas de transição, os parâmetros de evolução, o processo de recombinação e a mutação usados nesta tese foram resultados de observações empíricas. Uma análise mais aprofunda destes componentes do AGC seria conveniente e ainda pode ser desenvolvida.

7.2 – TRABALHOS FUTUROS

Nesta tese desenvolvemos as principais idéias, conceitos e fundamentos do AGC e aplicamos o novo método a três diferentes problemas de otimização combinatória. Entretanto, existem ainda muitas outras possibilidades de aplicação do método, e que merecem ser exploradas. Assim, vamos mostrar como poderiamos usar o AGC em outras três aplicações: coloração de grafos, roteamento de veículos e problema de "*bin-packing*".

7.1.1 – Problema de coloração de vértices de grafo

O problema pode ser definido conforme segue. Um grafo não direcionado $G = (V,E)$ consiste de um conjunto finito V de vértices e uma família E de arestas, que são

pares não ordenados *(x,y)* de vértices distintos. Uma *k-coloração* de *G* é uma partição do conjunto *V* em *k* conjuntos independentes $C_1, C_2, ..., C_k$. Um conjunto *C* de vértices é chamado *independente* quando não há dois vértices em *C* ligados por uma aresta. O menor *k* para o qual existe uma *k-coloração* de *G* é chamado o *número cromático* de *G*, e é indicado por $\chi(G)$.

O particionamento descrito acima equivale a atribuir a cada vértice do grafo *G* uma cor *i*, $1 \leq i \leq k$, de modo que dois vértices adjacentes não tenham a mesma cor. Esta abordagem é conhecida como *coloração de vértices* do grafo.

Ribeiro Filho (1997), em sua dissertação de mestrado, implementou o AGC usando uma representação, onde a cadeia mostra explicitamente a atribuição das cores. A figura 7.1 mostra esta representação.

Usando uma heurística de transição, poderíamos alterar esta representação e os símbolos 1, 2 e # seriam suficientes. Acreditamos que os resultados poderiam ser melhorados a partir desta nova representação, uma vez que existe uma busca local associada a heurística de transição.

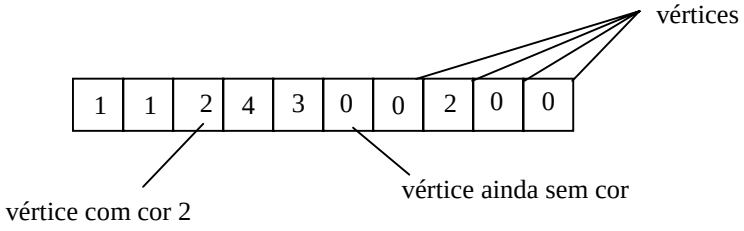


Figura 7.1: Representação usada no problema de coloração de grafos.

Para obter as funções de avaliação, um grafo completo é formado e as funções podem ser definidas assim:

$$g = \sum_{p=1}^{k} [\frac{(|C_p| - 1).|C_p|}{2}] \qquad (7.1)$$

e

$$f = \sum_{p=1}^{k} \{[\frac{(|C_p|-1).|C_p|}{2}] - |E(C_p)|\} \quad (7.2)$$

onde

C_p: conjunto de vértices com a cor *p*;

$|C_p|$: número de elementos do conjunto C_p; e

E(C_p*)* é o conjunto de arestas de *G* ligando os vértices pertencentes a C_p.

Observe-se que quanto menor o número de arestas de *G* ligando vértices pertencentes a C_p, maior o valor de *f*, e melhor a solução aproximada de *k*-coloração. Quando para uma estrutura completa *f=g*, obtemos uma solução para o problema.

Para calcular o limite superior, assumimos que cada conjunto independente apresenta o mesmo número de vértices. Assim:

$$g_{\max} = \omega.k[\frac{([\frac{m}{k}]-1).(\frac{m}{k})}{2}] \quad (7.3)$$

em que *m* é o número de vértices de *G*, ω é um fator para garantir que $g_{max} > g$, e que é um limite superior para o valor da função que avalia a solução aproximada de *k*-coloração para o problema original, isto é, todo o grafo *G*.

Para usar a nova representação (com os símbolos 1,2 e #), poderíamos utilizar uma forma semelhante a usada no PCPG. Por exemplo, s=(2 1 # 1 2 1 2 2) mostra que os vértices 2, 4 e 6 são sementes e receberam cores distintas. Os demais vértices podem receber atribuições de cores, de tal forma a minimizar o número de conflitos, isto é, o menor número de arestas ligando os vértices.

7.1.2 – Problema de roteamento de veículos

Considere a situação em que existem *n* clientes, onde cada um demanda uma certa quantidade, q_i, de mercadorias de um depósito. As mercadorias são entregues através de uma frota homogênea de veículos. Cada veículo inicia a viajem no depósito,

entrega mercadorias a um sub-conjunto de clientes, que recebem integralmente a mercadoria solicitada, e retorna ao depósito. A rota percorrida por cada veículo precisa satisfazer um conjunto de restrições. Por exemplo, a quantidade de mercadorias que um veículo precisa transportar não pode exceder sua capacidade e a distância máxima de uma rota poderia ser limitada. O problema de roteamento de veículos consiste em decidir quais veículos devem ser usados para atender quais clientes, e em que ordem isto deve ocorrer, de tal forma que todos clientes sejam atendidos e as restrições não sejam violadas, e alguma medida seja otimizada (normalmente a distância total percorrida pelos veículos).

Para este problema, podemos usar uma representação onde, inicialmente, um cliente é atribuído a cada uma das rotas (vértices sementes). Os demais vértices são atribuídos as rotas formadas pelos vértices sementes, de tal forma, a minimizar a distância percorrida pelo veículo. Podemos observar que variando os vértices sementes, novos agrupamentos, que determinam as rotas, são formados.

Após a formação dos agrupamentos, algum procedimento (como o descrito por Lin e Kernighan,1973) poderia ser usado como uma forma de busca local, para incrementar a qualidade das soluções.

7.1.3 – Problema de "*bin-packing*"

O problema consiste de um conjunto de caixas ("bins") $B=\{b_1,...,b_p\}$, iguais e dimensões fixas iguais a υ; e de um conjunto de objetos $O=\{o_1,...,o_m\}$, de diferentes dimensões que devem ser introduzidas nas caixas. O objetivo é minimizar o número de caixas utilizadas. Consequentemente, os objetos devem ocupar ao máximo a capacidade de cada caixa.

Aquino H. (1996) na sua proposta de dissertação de mestrado apresenta uma representação, conforme o seguinte exemplo. A estrutura s = (# 5 3 2 # 4 1) representa que o objeto número 1 (primeira posição na cadeia) ainda não foi atribuído a nenhuma caixa, o objeto número 2 (segundo posição na cadeia) pertence a caixa número 5, e assim segue.

As funções de avaliação podem ser as seguintes:

$$f = \sum_{i=1}^{p} \sum_{j \in b_i} o_j \tag{7.4}$$

$$g = \sum_{i=1}^{p} \sum_{j \in b_i} \lambda_i \mid b_i \mid \tag{7.5}$$

onde:

$$\lambda_i = \max\{o_j\},\ j \in b_i \tag{7.6}$$

A função f calcula o somatório da ocupação das caixas e g fornece um limite superior para cada caixa.

A função g_{max} pode ser dado como:

$$g_{\max} = \upsilon.m \tag{7.7}$$

A representação usada também poderia ser alterada para a nova forma (com os três símbolos:1,2 e #) e criada uma heurística de transição apropriada.

8 – REFERÊNCIAS BIBLIOGRÁFICAS

Alander, J.T. An indexed bibliography of genetic algorithms: Years 1957-1993. Technical Report 94-1, University of Vaasa, Finland, 1994. Disponível em *PostScript* via ftp de http://alife.santafe.edu.

Antonisse, J. A new interpretation of schema notation that overturns the binary encoding constraint. **Proceedings of the Third International Conference on Genetic Algorithms**. Morgan Kaufmann, 1989.

Aquino, H.Q. **Uma heurística construtiva para o problema de "bin-packing",** Proposta de dissertação de mestrado, Computação Aplicada, INPE, dezembro de 1996.

Automatic Control and Systems Engineering Department, University of Sheffield, http://www.shef.ac.uk/uni/projects/gaipp/index.html.

Bäck, Th. Self-Adaptation in Genetic Algorithms. **Procs. of the 1st European Conference on Artificial Life, F. J. Varela, P. Bourgine (eds.) MIT Press**, p. 263-271, 1992.

Bäck, Th.; Schwefel, H.P. An Overview of evolutionary algorithms for parameter optimization, **Evolutionary Computation**, v. 1, p. 1-23, 1993.

Baker, J.E. Adaptive selection methods for genetic algorithms. In J.J. Grefenstette, ed., **Proceedings of the First International Conference on Genetic Algorithms and Their Applications**. Erbaum. 1985.

Beasley, J.E. A note on solving large p-median problems. **European Journal of Operational Research**. v. 21, p. 270-273, 1985.

Beasley, J.E. OR-Library: distribution test problems by electronic mail**. Journal of Operational Research Society.** v. 41, p. 1069-1072, 1990.

Beasley, J.E. Lagrangean heuristics for Location problems. **European Journal of Operational Research**. v. 65, p. 383-399, 1993.

Bersini, H.; Seront, G. In Search of a good Evolution-Optimization Crossover. **Procs. 2nd Conf. On Parallel Problem Solving from Nature** (PPSN II), R. Männer, B. Manderick (eds.), Elsevier Science Publishers, p. 479-488, 1992.

Cerny, V. A thermodynamical approach to the travelling salesman problem: an efficient simulated annealing algorithm, **Journal of Optimization Theory and Applications**. v. 45, p. 41-51, 1985.

Cristofides, N.; Beasley, J.E. A tree search algorithm for the p-median problems. **European Journal of Operational Research**. v. 10, p. 196-204, 1982.

Davis, L.D. **Handbook of Genetic Algorithms.** Van Nostrand Reinhold, 1991.

De Jong, K. **An analysis of the behaviour of a class of genetic adaptive systems**. Ph.D. thesis, University of Michigan, Ann Arbor, MI, 1975.

De Jong, K.A.; Sarma, J. Generation gaps revisited. In L.D. Whitley, ed., **Foundations of Genetic Algorithms 2. Morgan Kaufmann,** 1993.

Efroymson, M.A.; Ray, T.L. A branch-and-bound algorithm for plan location. **Operational Research**. v. 14, p. 361-368, 1966.

Eiben, A.E.; Raué, P.E.; Ruttkay, Z. Genetic algorithms with multi-parent recombination. **Procs. 3rd Conf. On Parallel Problem Solving from Nature** (PPSN III), Y. Davidor, H.P. Schwefel, R. Männer (eds.), Springer LNCS 866, p. 78-87, 1994.

Evolutionary Computation Group, Department of Artificial Intelligence, University of Edinburgh, http://www.dai.ed.ac.uk.

Evolutionary Computing Group, University of the West of England, Bristol, U.K., http://www.btc.uwe.ac.uk/evol/index.html.

Fogarty, T.C. Varying the Probability of Mutation in the Genetic Algorithm. **Procs. of the 3rd. Int. Conf. on Genetic Algorithms (ICGA), J.D. Schaffer (ed.), Morgan Kaufmann Publ.,** p. 104-109, 1989.

Fogel, L.J.; Owens, A.J.; Walsh, M.J. **Artificial Intelligence through Simulated Evolution**, New York: Wiley, 1966.

Fogel, D.B. **System identification through simulated evolution: A Machine Learning Approach to Modeling**. Needham Heights: Ginn Press, 1991.

Galvão, R.D.; Raggi, L.A. A method for solving to optimality uncapacitated location problems. **Annals of Operations Research**. v. 18, p. 225-244, 1989.

Galvão, R.D.; Ferreira Filho, V.J.M.; Rivas, M.P.A. Some computational aspects of p-median type location problems**. VIII Congresso CLAIO/XXVIII SBPO**, Rio de Janeiro, p. 1266-1271, 1996.

Garey, M. R.; Johnson, D. S. **Computers and Intractability: a Guide to the Theory of NP-Completeness.** San Francisco: Freeman, 1979.

Genetic Algorithms Group (GAG), George Mason University, Fairfax, Virginia, http://www.cs.gmu.edu/research/gag/

Genetic Algorithms Research and Applications Group, Michigan State University, http://isl.msu.edu//GA/.

Glover, F. Tabu search – Part I. **ORSA Journal on Computing**. v. 1, p. 190-206, 1989.

Glover, F. Tabu search – Part II. **ORSA Journal on Computing**. v. 2, p. 4-32, 1990.

Goldberg, D.E. Genetic algorithms in search, optimization and machine learning. **Addison-Wesley***,* Reading, MA, p. 11-172, 1989.

Goldberg, D.E.; Korb, B.; Deb, K. Messy genetic algorithms: Motivation, analysis, and first results**. Complex Systems** v. 3: p. 493-530, 1989.

Goldberg, D.E. A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. **Complex Systems** v. 4: p. 445-460, 1990.

Goldberg, D.E.; Deb, K. A comparitive analysis of selection schemes used in genetic algorithms . In G. Rawlins, ed**., Foundations of Genetic Algorithms, Morgan Kaufmann**, 1991.

Goldberg, D.E.; Deb, K.; Kargupta, H.; Harik, G. **Rapid, accurate optimization of difficult problems using fast messy genetic algorithms**. IlliGAL Report No. 93004, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1993.

Golden, B.; Skiscim, C. Using simulated annealing to solve routing and location problems. **Naval Research Logistics Quarlerly***,* v. 33, p. 261-279, 1986.

Grefenstette, J.J. Optimization of control parameters for genetic algorithms, **IEEE Transactions on Systems, Man and Cybernetics** SMC v. 16(1), p. 122-128, 1986.

Grefenstette, J.J. **Proceedings of the second int'l conference on genetic algorithms and their applications**, Hillsdale, NJ:Lawrence Erlbaum, 1987.

Hakimi, S.L. Optimum distribution of switching centers and the absolute centers and the medians of a graph. **Operations Research.** v. 12, p. 450-459, 1964.

Hakimi, S.L. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. **Operations Research.** v.13, p. 462-475, 1965.

Hart, P.E.; Nilsson, J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Transactions on SSC** 4: p. 100-107, 1968.

Hart, P.E.; Nilsson, J.; Raphael, B. Correction to "A formal basis of the heuristic determination of minimum cost paths". **SIGART Newsletter** 37: p. 28-29, 1972.

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

Illinois Genetic Algorithm Laboratory. *IlliGAL*, University of Illinois at Urbana-Champaign, http://gal4.ge.uiuc.edu.

Institut für Prozeβ und Anlagentechnik, Arbeitsgruppe Bionik und Evolutionstechnik, TU Berlin, http://lautaro.fb10.tu-berlin.de.

Janikow, C. Z.; Michalewicz, Z. An experimental comparison of binary and floating point representations in genetic algorithms. In R. K. Belew and L.B. Booker, eds., **Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan kaufmann**, 1991.

Jarvinen, P.; Rajala, J. ; Sinervo, H. A branch and bound algorithm for seeking the p-median. **Operations Research**, v. 20, p. 173-178, 1972.

Jarvinen, P.J.; Rajala, J. A branch and bound algorithm for seeking the p-median. **Operations Research**. v. 20, p. 173-178, 1972.

Johnson, E.L.; Mehrotra, A. Min-Cut Clustering. Working paper. **School of Industrial& Systems Engineering, Georgia Institute of Technology, Atlanta**, 1992.

Julstrom, B.A. What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm, **Procs. of the 6th Int. Conf. on Genetic Algorithms (ICGA), L. J. Eshelman (ed.), Morgan Kaufmann Publ.** p. 81-87, 1995.

Kargupta, H. **Search, Polynomial Complexity, and The Fast Messy Genetic Algorithm**, Ph.D. thesis, IlliGAL Report No. 95008, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, 1995.

Kirkpatrick, S.; Gelat, C.D.; Vecchi, M. P. Optimization by simulated annealing. **Science**, v. 220, p. 671-680, 1983.

Klein, K. ; Aronson, J.E. Optimal clustering: a model and method. **Naval Research Logistics**, v. 38, p. 447-461, 1991.

Koontz, W.L.G.; Patrenahall, M.N. ; Fukunaga, K. A branch and bound clustering algorithm. **IIE Transactions on Computers**, v. 24, p. 908-915, 1975.

Koza, J.R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**, MIT Press*,* 1992.

Koza, J.R. **Genetic Programming II: Automatic Discovery of Reusable Programs.** MIT Press*,* 1994.

Lin, S.; Kernighan, B.W. An effective heuristic algorithm for the traveling-salesman problem. **Operations Research,** 21, p. 498-516, 1973.

Lorena, L.A.N.; Lopes, F.B. A dynamic list heuristic for 2D-cutting. In J. Dolezal and J. Fidler eds., **System Modelling and Optimization, Chapman-hall,** p. 481-488, 1996a.

Lorena, L.A.N.; Lopes, L.S. Computational experiments with genetic algorithms applied to set covering problems**. Pesquisa Operacional**, 16, p. 41-53, 1996b.

Lorena, L.A.N.; Lopes, L.S. Genetic algorithms applied to computationally difficult set covering problems. **Journal of the Operational Research Society**, 48, p. 440-445, 1997.

Mitchell, M. **An Introduction to Genetic Algorithms**. MIT Press, Cambridge, England, 1996.

Mühlenbein, H.; Schlierkamp-Voosen, D. The science of breeding and its application to the breeder genetic algorithm BGA**. Evolutionary Computation***,* v. 1, n. 4, p. 335-360, 1994.

Mulvey, J.M.; Crowder, H.P. Cluster analysis: an application of Lagrangean relaxation. **Management Science**, Vol. 24, pp. 329-340, 1979.

Mulvey, J. M.; Beck, M.P. Solving capacitated clustering problems. **European Journal of Operational Research**, v. 18, p. 339-348, 1984.

Navy Center for Applied Research in Artificial Intelligence, Washington, D.C., http://www.aic.nrl.navy.mil.

Neebe, A.W. A branch and bound algorithm for the p-median transportation problem. **Journal of the Operational Research Society**. v. 29, p. 989-995, 1978.

Osman, I. H.; Christofides, N. Capacitated clustering problems by hybrid simulated annealing and tabu search, **International Transactions in Operational Research**, v. 1, n. 3, p. 317-336, 1994.

Pearl, J. A_{ϵ}* - An algorithm using search effort estimates, **IEEE Transactions on Pattern Analysis and Machine Intelligence**, vol. 4, n. 4, p. 392-399, 1982.

Pearl, J. **Heuristics – Intelligent search strategies for computer problem solving**, Addison-Wesley Publishing Company, MA, 1985.

Prügel-Bennett, A.; Shapiro, J.L. An analysis of genetic algorithms using statistical mechanics, **Physical Review Letters** 72, n. 9, p. 1305-1309, 1994.

Rahman, S.; Smith, D.K. A comparison of two methods for the p-median problem with and without maximum distance constraints**. International Journal of Operations and Production Management**, v. 11, p. 76-84, 1991.

Rao, M.R. Clustering analysis and mathematical programming. **Journal of the American Statistical Association**, v. 66, p. 622-626, 1971.

Rechenberg, I. Cybernetic solution path of an experimental problem, **Roy. Aircr. Establ., libr. transl**. 1122, Fornborough, Hants., UK, 1965.

Research Group Adaptive Systems, GMD, St. Augustin, H. Mühlenbein, http://borneo.gmd.de/AS/pages/as.html.

Ribeiro Filho, G. **Uma heurística construtiva para coloração de grafos**. Dissertação de mestrado, INPE, 1997.

Roberts, F.S. Meaningfulness of conclusions from combinatorial optimization. **Discrete Applied Mathematics**, 29, p. 221-242, 1990.

Santa Fe Institute, New Mexico, http://www.santafe.edu.

Schwefel, H.P. **Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik**, Diploma thesis, Technical University of Berlin, 1965.

Senne, E.L.F.; Lorena, L.A.N. A lagrangean/surrogate approach to p-median problems. **European Journal of Operational Research** - submitted. 1997.

Smith, J.; Fogarty, T.C. Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm, **Procs. IEEE Int. Conf. on Evolutionary Computation (ICEC'96), IEEE Press**, p. 318-323, 1996.

Spears, W.M. Crossover or mutation? In L.D. Whitley, ed., **Foundations of Genetic Algorithms 2. Morgan Kaufmann**, 1993.

Syswerda, G. A study of reproduction in generational and steady-state genetic algorithms. In G. Rawlins, ed., **Foundations of Genetic Algorithms, Morgan Kaufmann**, 1991.

Teitz, M.B.; Bart, P. Heuristic methods for estimating the vertex median of a weighted graph. **Operations Research**. v. 16, p. 955-961, 1968.

Vinod, H. D. Integer programming and the theory of groups. **Journal of the American Statistical Association**, v. 64, p. 506-519, 1969.

Wang, P.Y. Two algorithms for constrained two-dimensional cutting stock problems, **Operations Research** 31, p. 573-586, 1983.

Whitaker, R.A. A fast algorithm for the greedy interchange for large scale clustering and median location problems. **Canadian Journal of Operational Research and Information Processing**, v. 21, p. 95-108, 1983.

Wright, A.H. Genetic algorithms for real parameter optimization. In G. Rawlins, ed., **Foundations of Genetic Algorithms. Morgan Kaufmann**, 1991.

APÊNDICE A

A seguir temos um conjunto de tabelas que mostram os resultados para algumas instâncias da biblioteca OR (Beasley, 1990). Todos resultados abaixo representam soluções ótimas para os problemas. Após a identificação do problema, as tabelas mostram os vértices que são medianas e esclarece como os vértices estão atribuídos às medianas. O custo de cada agrupamento também é informado.

Tabela A.1: Medianas, atribuições e custos para o problema pmed1.

IDENTIFICAÇÃO	
Problema	*Pmed1*
Número total de vértices	*100*
Número de medianas	*5*
Custo total	*5819*

MEDIANA	ATRIBUIÇÕES	CUSTO
7	2,3,4,5,6,8,19,20,21,22,27,35,36,39,45,46,49,50,51,52,53,59,60,61,62,68,80,93, 96	1665
13	10,11,12,14,15,16,17,18,30,37,38,40,41,42,43,44,47,54,55,56,57,58,69,70,71, 72,73,81,82,84,85,86	2147
65	63,64,66,67,94	241
91	33,34,74,75,76,77,78,79,87,88,89,90,92	701
99	1,9,23,24,25,26,28,29,31,32,48,83,95,97,98,100	1065
CUSTO TOTAL:		5819

Tabela A.2: Medianas, atribuições e custos para o problema pmed2.

IDENTIFICAÇÃO	
Problema	*Pmed2*
Número total de vértices	*100*
Número de medianas	*10*
Custo total	*4093*

MEDIANA	ATRIBUIÇÕES	CUSTO
6	5,25,26,27,92,93	109
8	7,9,55,86	148
12	10,11,13,14,62,63,80,81,82	419
37	15,16,17,18,19,22,23,24,35,36,38,49,50,60,61,64,65,71,72,75,76,77,79,84,87, 88	1358
41	4,28,39,40,42,51,52,53,73,74,85,98,99	640
45	43,44,46,47,78	330
58	1,2,3,20,21,48,56,57,59,69,70,83,89,90,91	640
67	29,30,31,34,66,68	281
95	32,33,94,96,97	150
99	54,100	18
CUSTO TOTAL:		4093

Tabela A.3: Medianas, atribuições e custos para o problema pmed3.

IDENTIFICAÇÃO	
Problema	*Pmed3*
Número total de vértices	*100*
Número de medianas	*10*
Custo total	*4250*

MEDIANA	ATRIBUIÇÕES	CUSTO
5	4,6,57,58,74	100
9	7,8,10,11,61,62,63,64,75,76,77,78,79,80	439
13	12,14,15,16,17,30,43,44,66,82	517
21	2,33,41,45,83,84	292
26	22,27,31,32,42,59,81,93,94	469
36	1,24,25,26,28,29,34,35,37,38,73,85,86,89,92	669
48	40,46,47,49,65,72,96,97	344
55	18,19,20,21,23,54,56,60,95	384
69	3,39,50,53,67,68,70,71	409
99	51,52,87,88,90,91,98,100	627
CUSTO TOTAL:		4250

Tabela A.4: Medianas, atribuições e custos para o problema pmed4.

IDENTIFICAÇÃO	
Problema	*Pmed4*
Número total de vértices	*100*
Número de medianas	*20*
Custo total	*3034*

MEDIANA	ATRIBUIÇÕES	CUSTO
1	100	10
5	6	24
8	7,70,73	95
10	9,11,85	103
13	12,14	98
22	15,18,20,21,36,42,43,54,55,80,81	437
26	16,17,23,25,27,28,29,30,31,32,57,58,68	591
34	33,35,46	97
38	37,39,63,89	114
51	50	19
55	4,5,19,40,41,44,45,56,99	275
60	59,61	37
66	24,65,67	160
72	62,71	86
77	74,75,76,78,79,98	216
83	82,84	125
87	52,53,69,86,88	183
91	64,90	68
93	92,94	85
96	2,3,47,48,49,95,97	211
CUSTO TOTAL:		3034

Tabela A.5: Medianas, atribuições e custos para o problema pmed6.

IDENTIFICAÇÃO	
Problema	*Pmed6*
Número total de vértices	*200*
Número de medianas	*5*
Custo total	*7824*

MEDIANA	ATRIBUIÇÕES	CUSTO
16	4,5,8,9,14,15,17,18,24,28,29,39,59,60,67,72,73,74,75,78,79,118,124,125,126, 139,151,168,171,177,179,182,183,190,191,192,197,198	1557
86	12,20,27,31,32,33,35,41,44,48,55,56,57,65,76,77,80,85,87,88,91,95,97,99,138, 148,158,161,165,166,178,187,188,189,193,194	1449
101	2,3,7,11,21,34,37,38,52,53,61,69,89,90,94,98,100,105,106,107,113,114,116, 117,122,128,134,135,136,137,145,146,152,153,154,159,160,163,174,184,185, 186,195,196	1617
111	1,6,10,13,22,23,25,26,43,47,49,51,54,58,62,63,66,68,70,71,84,92,93,102,103, 109,110,112,115,119,120,121,129,133,143,144,147,149,150,156,157,164,167, 169,172,173,199,200	2110
126	19,30,36,40,42,45,46,50,64,81,82,83,96,104,108,123,127,130,131,132,140,141, 142,155,162,170,175,176,180,181	1091
CUSTO TOTAL:		7824

Tabela A.6: Medianas, atribuições e custos para o problema pmed7.

IDENTIFICAÇÃO	
Problema	*Pmed7*
Número total de vértices	*200*
Número de medianas	*10*
Custo total	*5631*

MEDIANA	ATRIBUIÇÕES	CUSTO
3	4,19,93,94,97,105,106,141,142,161,162,163	451
10	8,9,11,12,13,14,15,42,52,74,75,76,81,102,103,125,172,179,184,195	647
72	7,25,71,73,82,86,87,100,120,124,168,169,188,193	380
87	34,35,36,53,62,85,88,89,98,143	252
131	6,16,57,59,90,91,95,96,110,130	343
142	1,31,37,44,45,50,63,65,84,101,107,111,121,122,123,144,153,176,183,194,197	625
181	5,17,20,22,23,24,26,27,32,33,38,40,47,48,51,58,60,61,68,69,70,108,112,114, 127,128,129,137,138,140,150,152,156,160,171,175,178,180,182,185,186,189, 198,199	1152
186	21,39,43,49,54,55,56,67,83,113,145,146,177,187	421
191	2,3,18,41,46,66,77,78,99,104,119,126,147,149,154,155,165,170,173,190,192	605
199	28,29,30,64,79,80,92,109,115,116,117,118,132,133,134,135,136,139,148,151, 157,158,159,164,166,167,174,196,200	755
CUSTO TOTAL:		5631

Tabela A.7: Medianas, atribuições e custos para o problema pmed11.

IDENTIFICAÇÃO	
Problema	*Pmed11*
Número total de vértices	*300*
Número de medianas	*5*
Custo total	*7696*

MEDIANA	ATRIBUIÇÕES	CUSTO
24	4,10,13,17,29,45,57,61,62,73,74,97,98,102,107,111,116,125,145,151,155,171, 176,178,179,183,198,202,219,220,225,232,241,250,251,254,255,259,264,269, 275,276,299	925
31	30,32,34,46,49,70,71,72,87,94,104,105,108,138,175,186,204,206,209,210,214, 224,229,233,236,244,245,247,248,256,257,262,274,282,284	952
98	1,2,6,8,11,15,20,23,24,25,33,40,43,44,47,50,64,65,67,68,75,76,77,81,82,83,84, 90,92,93,95,96,99,106,112,113,115,117,118,119,124,129,131,134,136,137,141, 144,150,160,161,162,163,164,166,167,168,169,170,173,177,185,187,188,189, 190,196,197,200,201,205,207,211,212,213,215,217,218,221,222,226,227,228, 239,242,243,246,252,253,266,267,277,283,288,292,293,294,295,300	2408
167	5,9,16,18,19,35,42,55,56,69,79,80,91,103,109,110,121,122,123,128,130,133, 140,148,157,158,174,184,192,193,194,203,249,281,285,286,289,296,298	928
201	3,7,12,14,21,22,26,27,28,36,37,38,39,41,48,51,52,53,54,58,59,60,63,66,78,85, 86,88,89,100,101,114,120,126,127,132,135,139,142,143,146,147,149,152,153, 154,156,159,165,172,180,181,182,191,195,199,208,216,223,230,231,234,235, 237,238,240,258,260,261,263,265,268,270,271,272,273,278,279,280,287,290, 291,297	2483
CUSTO TOTAL:		7696

Tabela A.8: Medianas, atribuições e custos para o problema pmed16.

IDENTIFICAÇÃO	
Problema	*Pmed16*
Número total de vértices	*400*
Número de medianas	*5*
Custo total	*8162*

MEDIANA	ATRIBUIÇÕES	CUSTO
20	9,15,16,18,21,29,30,38,48,49,55,58,61,69,71,73,89,91,97,121,122,134,137,138, 139,142,143,146,148,156,159,160,172,176,179,181,185,203,204,205,208,215, 217,219,220,221,222,230,232,233,244,250,264,271,274,278,279,283,293,296, 298,302,305,307,310,321,333,342,344,366,371,387,391	1543
229	3,4,5,6,8,39,40,42,44,64,75,82,85,87,95,96,98,114,116,117,119,125,128,136, 151,153,155,168,169,170,175,182,183,190,191,192,193,195,196,216,242,247, 248,256,258,259,262,268,273,284,286,287,297,300,301,304,306,311,314,315, 316,318,319,322,327,328,331,335,336,337,338,339,340,341,343,345,347,349, 356,357,373,374,378,379,392,395,398	1775
267	7,12,17,19,20,27,32,33,34,36,37,41,47,50,51,54,57,59,60,62,63,66,68,70,74,76, 77,79,86,90,92,93,94,103,104,107,109,110,115,118,120,126,127,131,132,133, 135,140,141,149,152,157,161,162,163,164,165,166,171,173,174,177,178,184, 187,188,197,198,201,202,206,207,210,211,212,212,214,218,224,226,227,234, 235,236,237,238,240,245,246,251,252,253,255,257,260,265,269,270,272,277, 280,281,285,288,289,290,292,295,303,308,317,323,326,329,332,334,346,348, 353,359,362,363,364,365,369,370,372,376,382,383,386,389,390,394,396,397, 399,400	2831
374	1,11,14,22,23,24,28,31,43,52,53,56,65,72,78,81,83,84,106,108,111,112,144, 145,147,186,194,199,200,209,223,225,228,229,243,263,275,282,294,309,320, 350,351,354,355,360,375,388,393	887
379	2,10,13,25,26,35,45,46,67,80,88,99,100,101,102,105,113,123,124,129,130,150, 154,158,167,180,189,231,239,241,249,254,261,266,267,276,291,299,312,313, 324,325,330,352,358,361,367,368,377,380,381,384,385	1126
CUSTO TOTAL:		8162

Tabela A.9: Medianas, atribuições e custos para o problema pmed21.

IDENTIFICAÇÃO	
Problema	*Pmed21*
Número total de vértices	*500*
Número de medianas	*5*
Custo total	*9138*

MEDIANA	ATRIBUIÇÕES	CUSTO
71	1,5,7,14,20,21,22,23,29,33,34,38,47,54,58,67,70,72,79,89,90,95,96,103,107, 108,128,130,132,142,143,147,164,169,170,171,175,180,181,187,193,196,197, 203,208,211,219,220,221,223,224,225,228,230,236,245,248,253,258,263,270, 274,279,280,288,294,295,296,304,310,311,324,332,333,336,337,338,341,348, 357,359,361,369,373,374,375,376,392,404,409,420,424,430,443,450,467,470, 479,496,497,500	1865
138	9,13,15,17,32,41,42,43,44,45,46,56,57,69,75,76,84,86,91,92,98,105,109,110, 117,121,124,136,137,140,141,144,148,149,159,160,161,166,174,176,186,198, 199,200,205,212,213,215,216,222,229,231,234,237,241,244,246,264,265,271, 273,276,277,278,286,292,293,298,300,321,322,327,347,349,354,360,378,382, 384,386,387,394,395,400,401,410,411,419,432,435,452,462,473,498,499	1828
161	3,18,24,35,39,40,48,49,50,59,63,64,73,77,78,81,82,88,94,97,99,100,101,102, 111,112,113,118,127,134,135,145,151,162,163,201,202,204,217,226,227,232, 239,251,259,261,272,275,281,287,289,305,308,316,317,320,323,325,328,329, 330,339,342,345,346,351,355,358,367,368,370,379,380,381,398,414,415,421, 422,427,428,431,436,437,440,441,442,449,451,455,456,457,459,460,461,464, 468,469,477,478,481,483,484,485,486,487,488,490,491,492,495	1999
285	2,10,11,12,16,31,37,68,74,83,85,116,122,123,139,146,150,156,157,165,172, 173,182,185,189,207,209,210,214,235,247,249,250,255,257,260,267,283,284, 290,291,297,299,303,307,315,326,335,343,344,356,362,363,364,371,372,385, 391,393,396,397,402,406,407,408,412,417,418,423,425,426,439,444,445,448, 454,463,471,472,474,475,476	1511
494	4,6,8,19,25,26,27,28,30,36,51,52,53,55,61,62,65,66,80,87,93,104,106,114,115, 119,120,125,126,129,131,133,152,153,154,155,158,167,168,177,178,179,183, 184,188,190,191,192,194,195,206,218,233,238,240,242,243,252,254,256,262, 266,268,269,282,301,302,306,309,312,313,314,318,319,331,334,340,350,352, 353,365,366,377,383,388,389,390,399,403,405,413,416,429,433,434,438,446, 447,453,458,465,466,480,482,489,493	1935
CUSTO TOTAL:		9138

APÊNDICE B

A seguir temos um conjunto de tabelas e figuras que mostram os resultados para algumas instâncias do artigo de Osman e Christofides (Beasley, 1990). Os resultados representam situações em que a melhor solução conhecida foi encontrada. Para cada problema, as tabelas mostram quais os vértices que são medianas e esclarece como as atribuições ocorreram. O custo e demanda de cada agrupamento também é informado.

As figuras mostram as posições de cada vértice (cliente) e ilustram o processo de atribuição dos vértices às medianas.

Tabela B.1: Medianas, atribuições, demandas e custos para o problema pmedc1.

IDENTIFICAÇÃO	
Problema	*pmedc1.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*713*
Demanda total	*490*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
10	3,7,11,13,17,23,25,30,38,45,46,49	114	219
12	2,6,8,9,20,35,40,43	109	109
19	4,5,22,24,27,28,29,31,37,47	107	141
21	1,14,15,18,32,36,39,41,42,44,50	107	192
48	16,26,33,34	53	52
TOTAL		490	713

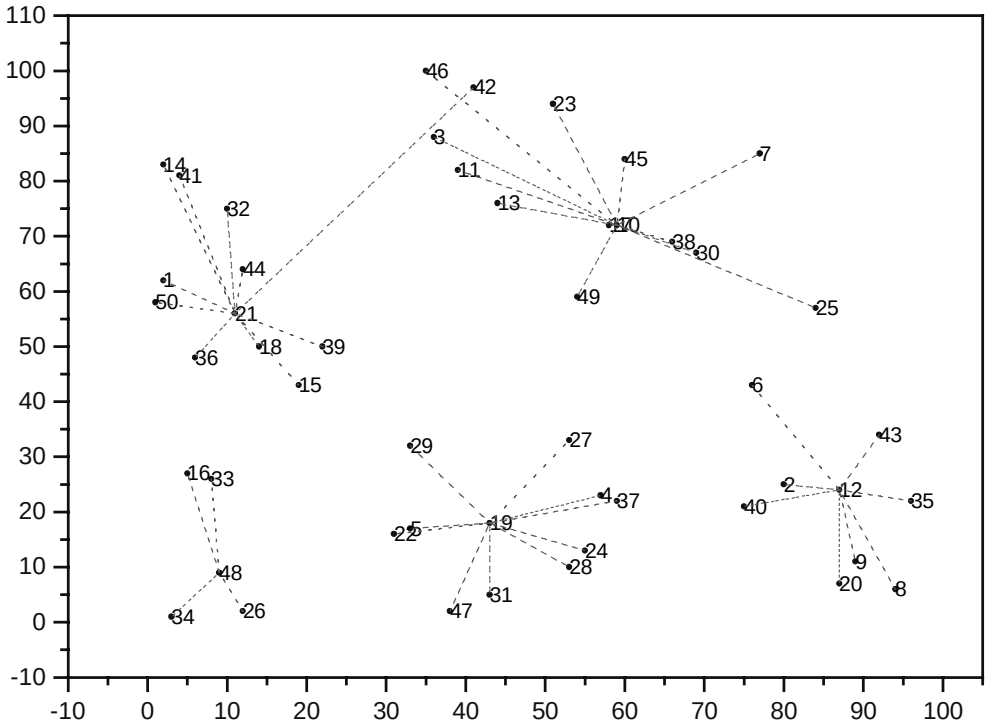


Tabela B.2: Medianas, atribuições, demandas e custos para o problema pmedc2.

IDENTIFICAÇÃO	
Problema	*pmedc2.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*740*
Demanda total	*502*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
16	6,10,18,24,27,30,37,40,42	71	163
22	2,4,5,11,12,14,31,32	109	138
26	1,7,9,15,17,39,43,48	90	137
33	3,19,25,29,34,36,41,44,46,49	117	178
47	8,13,20,21,23,28,35,38,45,50	115	124
TOTAL		502	740

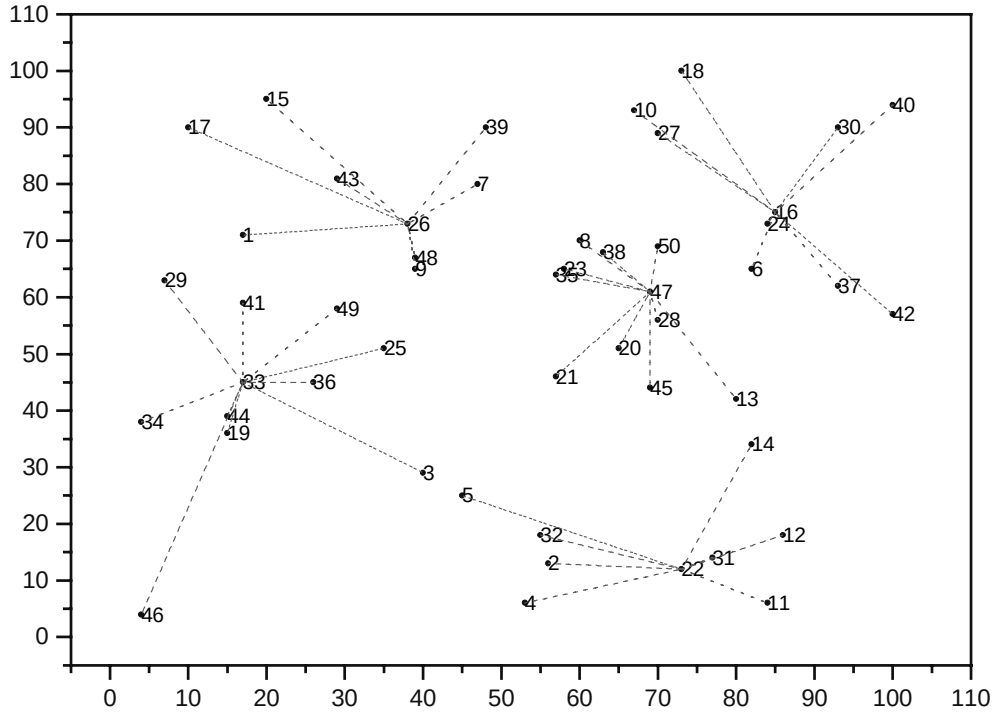


Tabela B.3: Medianas, atribuições, demandas e custos para o problema pmedc3.

IDENTIFICAÇÃO	
Problema	*pmedc3.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*751*
Demanda total	*512*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
15	1,5,9,24,28,43,49	74	145
20	3,11,14,18,21,23,34,36,45	107	133
38	2,25,26,32,37,42,44	116	118
39	4,8,10,12,13,17,19,29,31,41,46,47	120	181
48	6,7,16,22,27,30,33,35,40,50	95	174
TOTAL		512	751

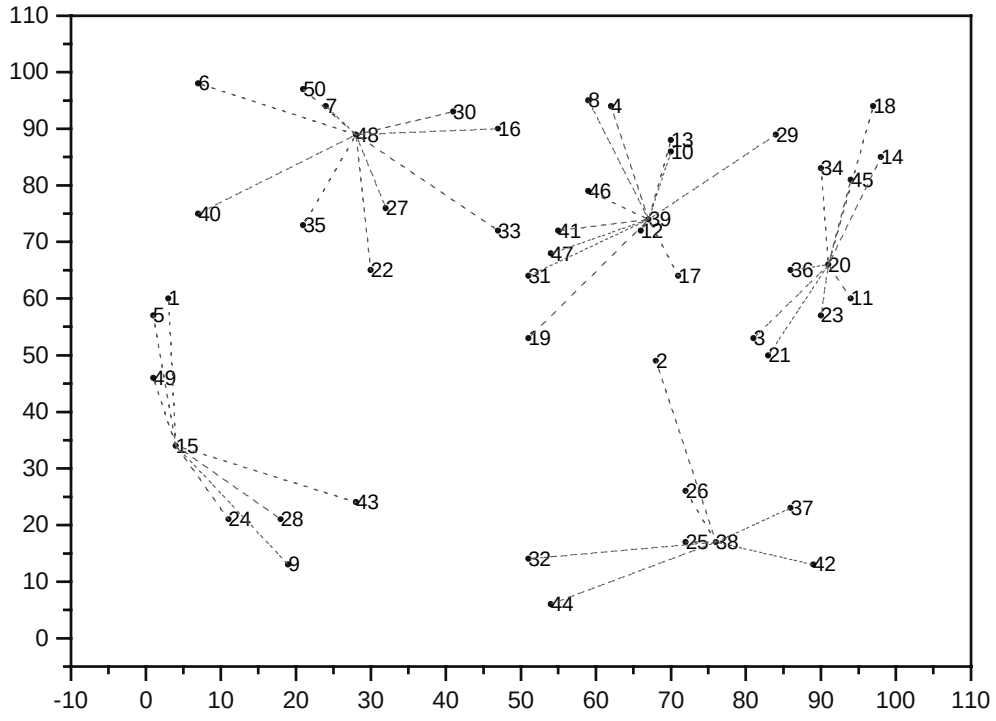


Tabela B.4: Medianas, atribuições, demandas e custos para o problema pmedc4.

IDENTIFICAÇÃO	
Problema	*pmedc4.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*651*
Demanda total	*517*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
3	5,6,15,24,31,34,36,38,47,48	119	187
9	4,7,28,32,35,37,39,40,44,46	105	98
29	11,16,19,20,27,30,33,41,42,45	119	99
43	1,2,8,12,13,14,21,49	84	124
50	10,17,18,22,23,25,26	90	143
TOTAL		517	651

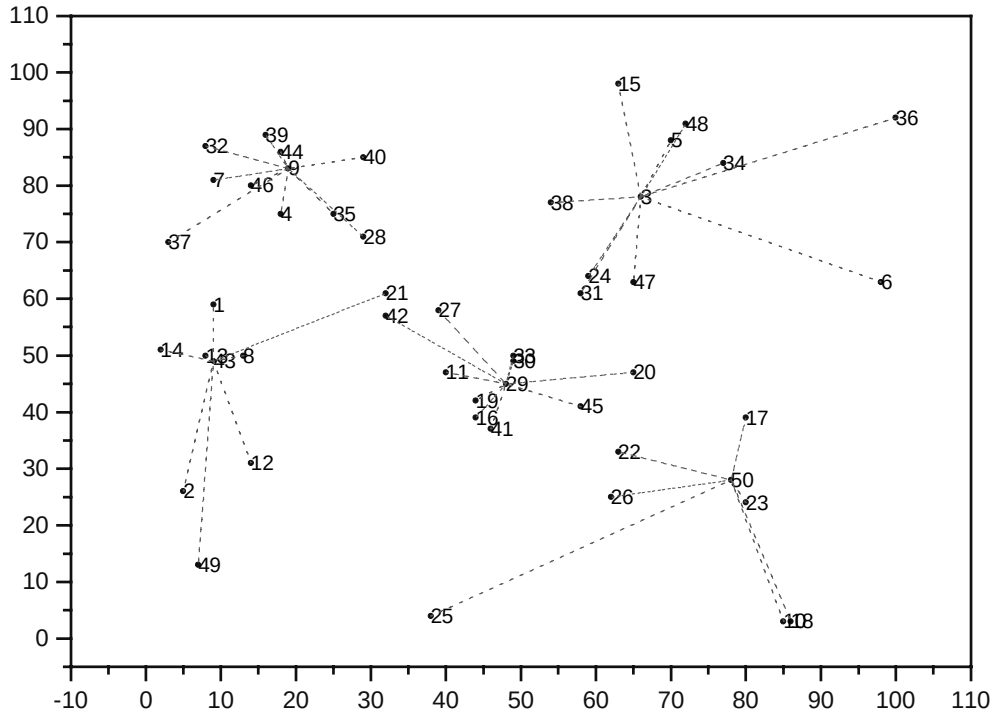


Tabela B.5: Medianas, atribuições, demandas e custos para o problema pmedc5.

IDENTIFICAÇÃO	
Problema	*pmedc5.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*664*
Demanda total	*541*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
13	4,5,6,11,15,16,25,30,48	85	147
22	1,18,19,23,24,31,34,46	106	111
29	8,10,14,21,33,35,39,44	114	111
36	7,9,17,20,37,42,49	116	98
40	2,3,12,26,27,28,32,38,41,43,45,47, 50	120	197
TOTAL		541	664

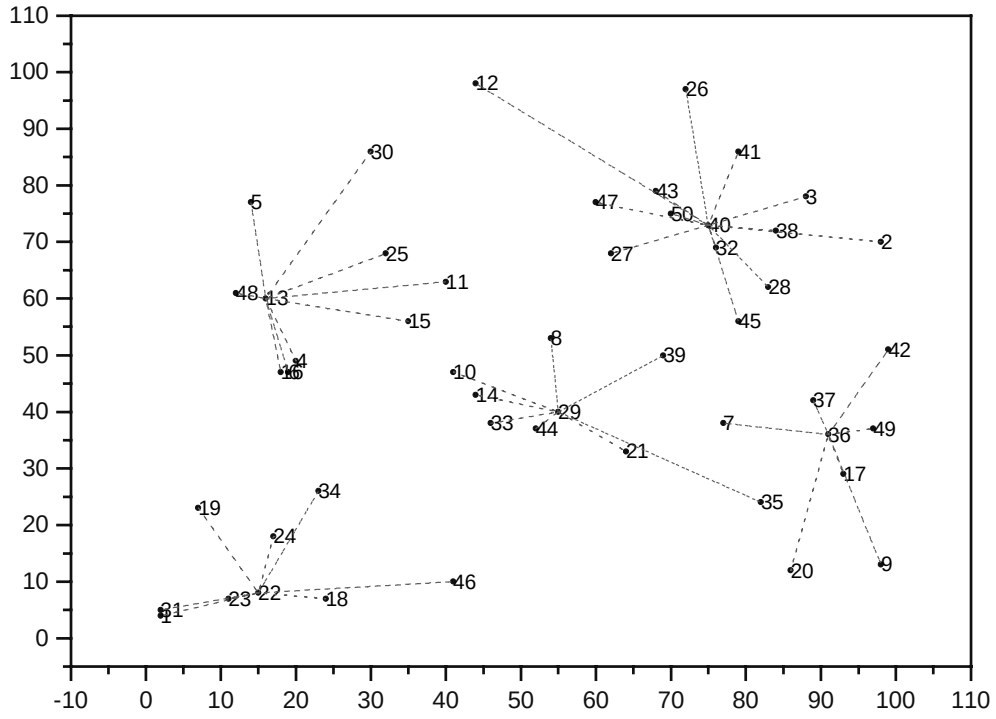


Tabela B.6: Medianas, atribuições, demandas e custos para o problema pmedc6.

IDENTIFICAÇÃO	
Problema	*pmedc6.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*778*
Demanda total	*550*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
7	9,12,19,23,24,34,43,44,47	108	122
17	2,8,16,21,25,26,29,30,50	112	151
20	4,11,14,22,33,35,36,37,41,49	119	194
42	1,3,6,18,27,28,31,40,45	93	129
46	5,10,13,15,32,38,39,48	118	182
TOTAL		550	778

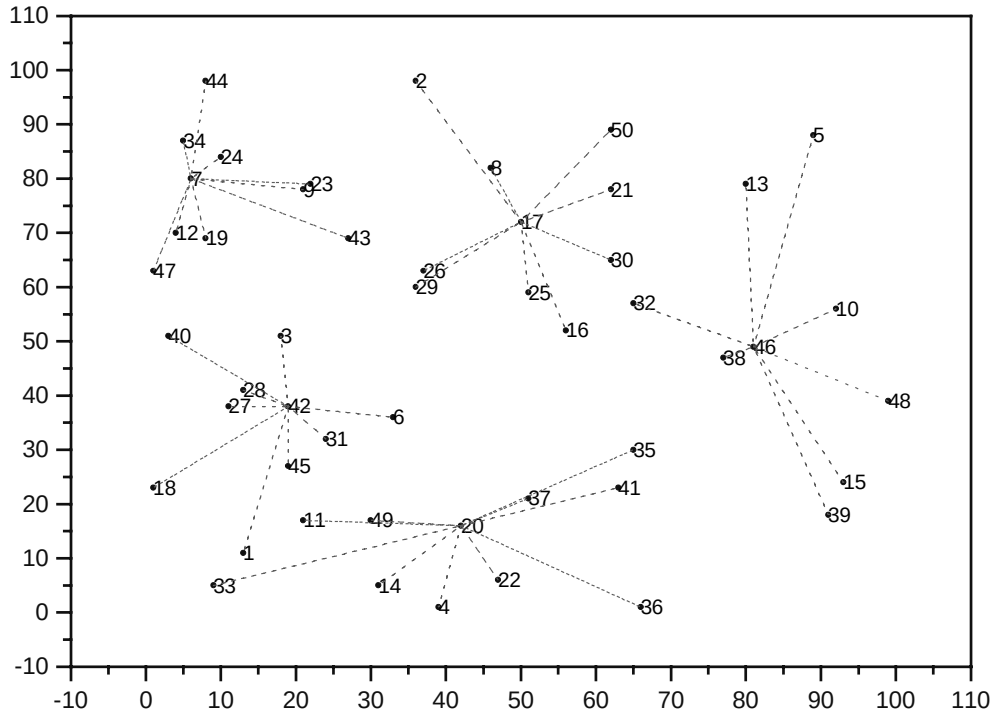


Tabela B.7: Medianas, atribuições, demandas e custos para o problema pmedc7.

IDENTIFICAÇÃO	
Problema	*Pmedc7.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*787*
Demanda total	*551*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
6	2,7,8,19,26,32,41,45	102	153
13	3,5,27,28,30,40,46,50	114	241
20	4,18,25,31,35,37,42,47	115	209
24	9,12,15,16,23,29,43,44,48,49	102	111
36	1,10,11,14,17,21,22,33,34,38,39	118	73
TOTAL		551	787

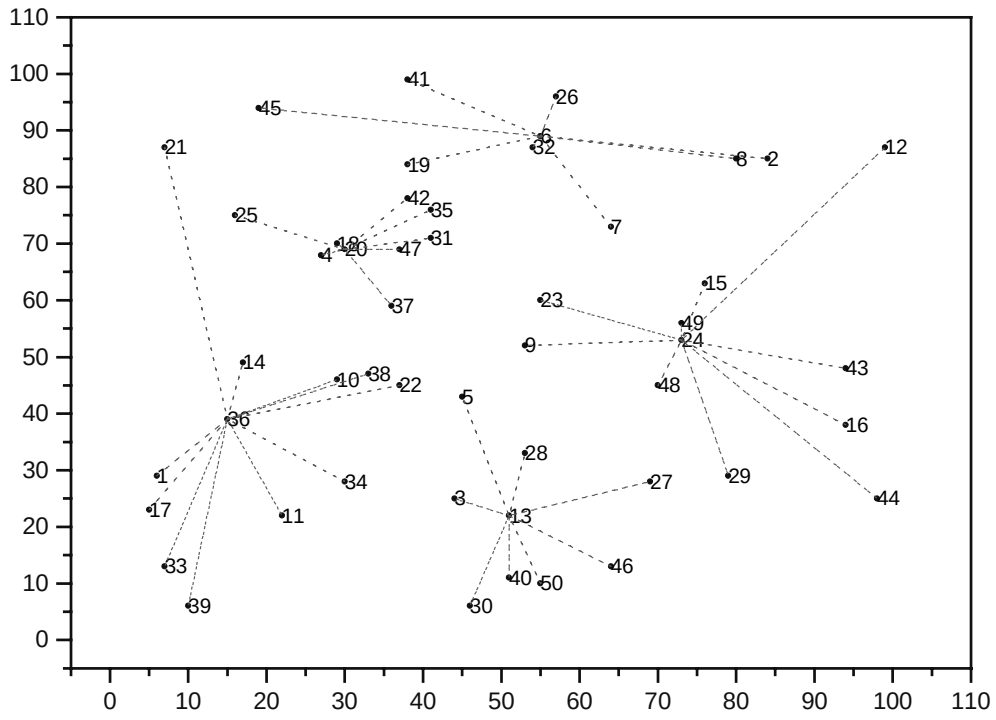


Tabela B.8: Medianas, atribuições, demandas e custos para o problema pmedc9.

IDENTIFICAÇÃO	
Problema	*Pmedc9.txt*
Número total de vértices	*50*
Número de medianas	*5*
Capacidade máxima do agrupamento	*120*
Custo total	*715*
Demanda total	*559*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
1	12,13,14,28,29,30,31,46,49,50	106	150
7	3,5,8,10,19,36,39,42	116	144
11	6,17,23,25,32,33,34,37,40	115	159
22	2,15,16,20,21,27,35,47,48	104	122
38	4,9,18,24,26,41,43,44,45	118	140
TOTAL		559	715

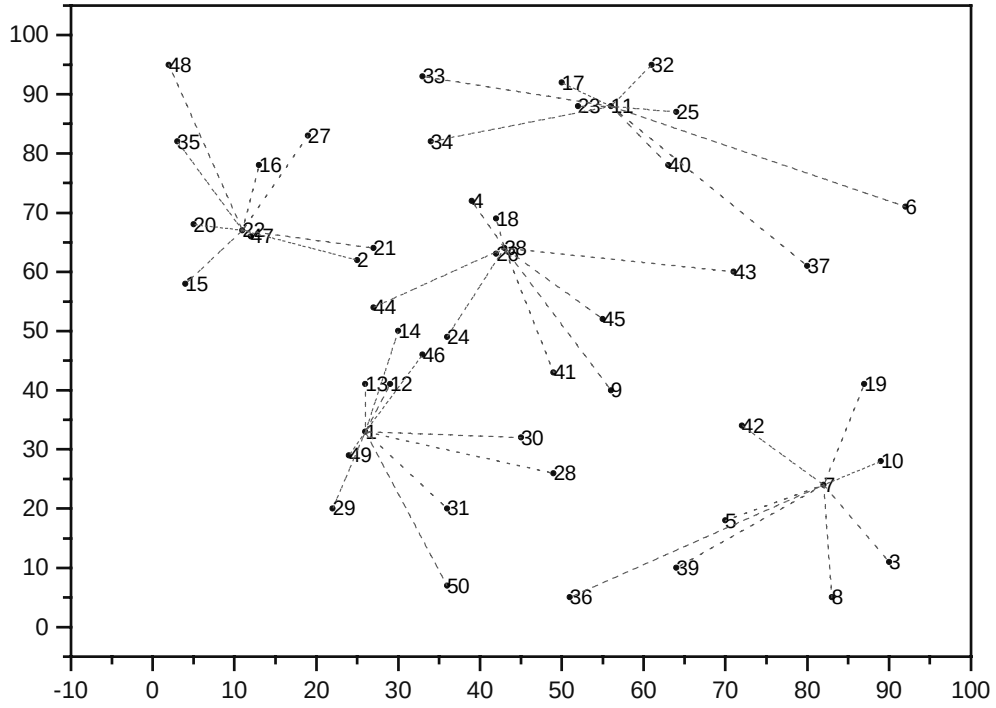


Tabela B.9: Medianas, atribuições, demandas e custos para o problema pmedc13.

IDENTIFICAÇÃO	
Problema	*Pmedc13.txt*
Número total de vértices	*100*
Número de medianas	*10*
Capacidade máxima do agrupamento	*120*
Custo total	*1026*
Demanda total	*1033*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
12	11,22,31,52,58,63,64,72,93,94	117	119
17	1,15,61,69,70,85,88,99	80	84
36	25,29,33,41,44,49,65,67,80,83,92	119	123
51	7,9,10,26,39,43,48,57,77,87,89	119	136
54	8,20,21,23,28,35,38,47,50,78,96	106	116
59	2,3,4,5,32,55,60,71	107	91
74	19,34,46,56,62,95,100	105	105
75	18,27,30,40,84,90,91,98	105	92
79	13,14,45,53,73,81,86,97	114	73
82	6,16,24,37,42,66,68,76	61	87
TOTAL		1033	1026

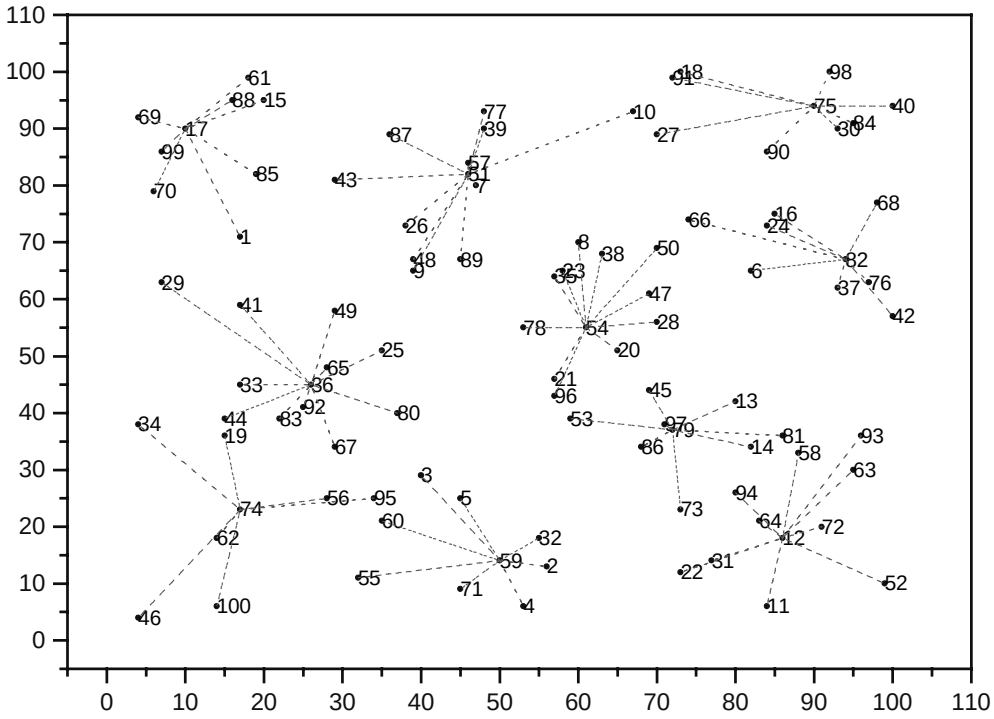


Tabela B.10: Medianas, atribuições, demandas e custos para o problema pmedc15.

IDENTIFICAÇÃO	
Problema	*pmedc15.txt*
Número total de vértices	*100*
Número de medianas	*10*
Capacidade máxima do agrupamento	*120*
Custo total	*1091*
Demanda total	*1050*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
5	10,13,52,56,68,91,98	66	96
8	2,17,21,50,66,72,74,76,95	63	115
22	4,14,20,36,37,64,71,89,90	120	107
45	1,6,11,18,27,28,31,33,42,49,55,57, 75	115	171
53	26,29,59,65,73,86	115	42
62	35,41,58,67,77,79,83,93,100	119	125
85	16,25,30,32,38,46,63,94,97	120	96
88	15,39,48,51,60,61,70,80,87	118	127
92	7,23,24,34,44,54,69,78,81,84,99	116	114
96	3,9,12,19,40,43,47,82	98	98
TOTAL		1050	1091

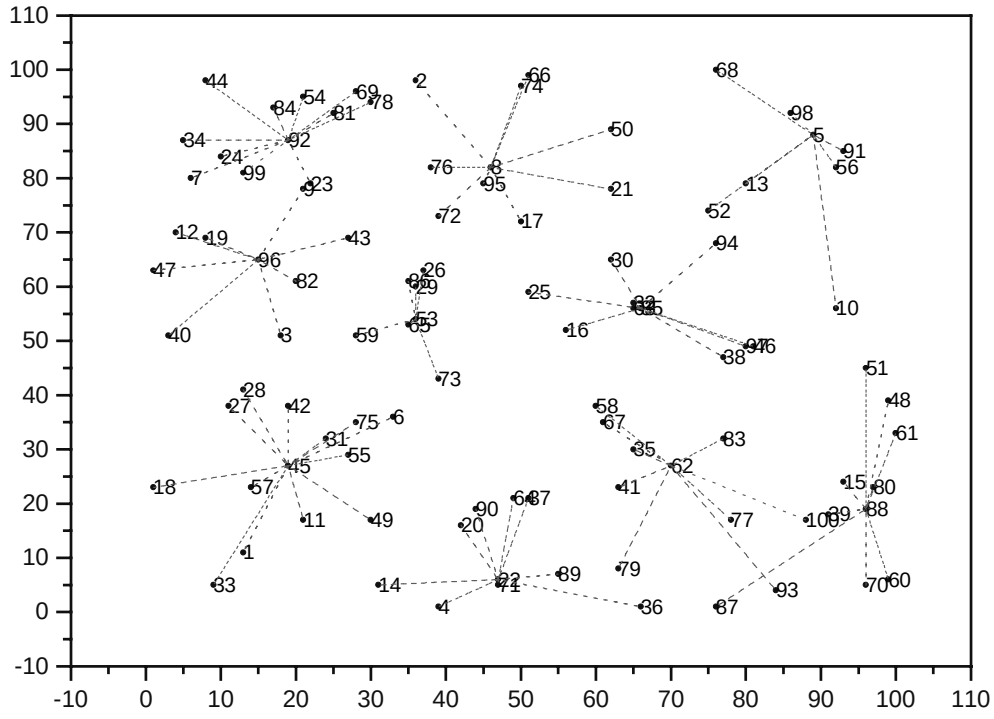


Tabela B.11: Medianas, atribuições, demandas e custos para o problema pmedc17.

IDENTIFICAÇÃO	
Problema	*pmedc17.txt*
Número total de vértices	*100*
Número de medianas	*10*
Capacidade máxima do agrupamento	*120*
Custo total	*1034*
Demanda total	*1073*

MEDIANA NO VÉRTICE	VÉRTICES ATRIBUÍDOS	DEMANDA	CUSTO
1	9,13,34,38,79,86,100	109	59
21	17,18,49,60,76,88	112	58
25	23,28,58,59,78,80	92	56
32	16,20,22,27,41,48,64,65,74,75,97	119	159
46	7,8,19,26,30,33,36,67,87,99	86	95
56	11,29,42,45,52,62,66,82,85,89,91, 95	115	190
61	2,3,5,15,37,39,94	119	66
71	24,35,40,53,55,68,70,72,90,92,93	110	95
73	6,10,12,47,50,57,63,77,98	115	89
81	4,14,31,43,44,51,54,69,83,84,96	96	167
TOTAL		1073	1034