UNIVERSIDADE CANDIDO MENDES – UCAM
PROGRAMA DE PÓS-GRADUAÇÃO EM PESQUISA OPERACIONAL E INTELIGENCIA COMPUTACIONAL
CURSO DE MESTRADO EM PESQUISA OPERACIONAL E INTELIGENCIA COMPUTACIONAL

Maycon Guedes Cordeiro

DESENVOLVIMENTO DE ALGORITMOS GENÉTICOS PARA O PROBLEMA DAS P-MEDIANAS UTILIZANDO OPERADORES DE CRUZAMENTO CONVENCIONAIS E NÃO-CONVENCIONAIS

CAMPOS DOS GOYTACAZES, RJ
ABRIL DE 2011

UNIVERSIDADE CANDIDO MENDES – UCAM
PROGRAMA DE PÓS-GRADUAÇÃO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL
CURSO DE MESTRADO EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL

Maycon Guedes Cordeiro

DESENVOLVIMENTO DE ALGORITMOS GENÉTICOS PARA O PROBLEMA DAS P-MEDIANAS UTILIZANDO OPERADORES DE CRUZAMENTO CONVENCIONAIS E NÃO-CONVENCIONAIS

Dissertação apresentada ao programa de Pós-Graduação em Pesquisa Operacional e Inteligência Computacional, da Universidade Candido Mendes – Campos/RJ, para a obtenção de grau de MESTRE EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL.

Orientador Prof. D.Sc. Dalessandro Soares Vianna
Co-orientadora: Prof[a]. M.Sc. Marcilene de Fátima Dianin Vianna

CAMPOS DOS GOYTACAZES, RJ
ABRIL DE 2011

MAYCON GUEDES CORDEIRO

DESENVOLVIMENTO DE ALGORITMOS GENÉTICOS PARA O PROBLEMA DAS P-MEDIANAS UTILIZANDO OPERADORES DE CRUZAMENTO CONVENCIONAIS E NÃO-CONVENCIONAIS

Dissertação apresentada ao programa de Pós-Graduação em Pesquisa Operacional e Inteligência Computacional, da Universidade Candido Mendes – Campos/RJ, Para a obtenção de grau de MESTRE EM PESQUISA OPERACIONAL E INTELIGÊNCIA COMPUTACIONAL.

Aprovada em 01 de abril de 2011.

BANCA EXAMINADORA

_____

Prof. Dalessandro Soares Vianna, D.Sc.
Universidade Federal Fluminense

_____

Profª. Marcilene de Fátima Dianin Vianna, M.Sc.
Universidade Federal Fluminense

_____

Prof. Fermín Alfredo Tang Montané, D.Sc.
Universidade Estadual Norte Fluminense

_____

Prof. Edwin Benito Mitacc Meza, D.Sc.
Universidade Federal Fluminense

CAMPOS DOS GOYTACAZES, RJ
ABRIL DE 2011

Dedico esse trabalho a todas as pessoas em que tive grande convívio, aos meus amigos e família por me apoiar e acreditar em mim, e aos que me desmotivaram e desacreditaram por me darem a vontade e garra de provar o contrário.

AGRADECIMENTOS

A minha mãe por todo apoio e amor que ela me deu durante este percurso.

Ao meu pai e avó por ser exemplo de bondade e dignidade em que me espelho.

Ao meu orientador, Dalessandro, por acreditar no meu trabalho, por sua valiosa contribuição nesta dissertação, pelas palavras de motivação verdadeiras e pelo seu profissionalismo em que me inspiro.

Aos meus grandes amigos Marcos Patrão e José Arildo por me ajudarem e servirem de exemplo na minha vida profissional.

Aos professores Tang e Marcilene, onde fico muito honrando por seus elogios e reconhecimento a este trabalho e que contribuíram muito com suas observações e correções minuciosas nesta dissertação.

Aos meus amigos Giovani, Diego, Fábio, Jackeline, Karla, Kelly, Márcio e Renata por acreditarem em mim e sempre me motivarem.

Aos meus amigos Thiago e Rodrigo que me ajudaram e me divertiram muito durante o período do mestrado em Campos.

Aos meus amigos do mestrado, por me proporcionar amizade, companheirismo, e companhia nos churrascos e sextas de Lord Pup.

A todos os profissionais que já trabalhei e que gostam de mim e me ajudaram de alguma forma, especialmente a Silvana.

RESUMO

DESENVOLVIMENTO DE ALGORITMOS GENÉTICOS PARA O PROBLEMA DAS P-MEDIANAS UTILIZANDO OPERADORES DE CRUZAMENTO CONVENCIONAIS E NÃO-CONVENCIONAIS

Neste trabalho são propostos algoritmos genéticos (AGs) para o problema das p-medianas utilizando diferentes operadores genéticos de cruzamento. Foram desenvolvidos cinco operadores de cruzamento, sendo dois deles variações de operadores clássicos da literatura – operadores convencionais – e outros três (não-convencionais). Dois destes operadores são baseados na técnica de reconexão por caminhos, a qual foi proposta inicialmente como estratégia de intensificação para explorar trajetórias entre soluções de elite obtidas por heurísticas busca tabu e busca dispersa (*scatter search*). O último é uma versão "reativa", a qual visa explorar de forma eficiente os diferentes operadores desenvolvidos. Estes AGs foram avaliados utilizando problemas testes disponíveis na literatura, para os quais já foram definidos limites inferiores. Os resultados obtidos demonstram a eficiência de se utilizar a técnica de reconexão por caminhos como operador de cruzamento em AGs.

PALAVRAS-CHAVE: Problema das p-medianas; algoritmos genéticos; reconexão por caminhos; algoritmos reativos.

ABSTRACT

DEVELOPMENT OF GENETIC ALGORITHMS FOR THE PROBLEM OF P-MEDIAN CROSSING USING CONVENTIONAL OPERATORS AND NON-CONVENTIONAL

The aim of this study is propose genetic algorithms (GA) for the p-median problem using different genetic operators of crossover. Were developed five crossover operators, two of them variations of classic operators on the literature – conventional operators – and three others (non-conventional). Two of these operators are based on the path-relinking technique, which was initially proposed as a strategy to explore paths of intensification between elite solutions obtained by tabu search and scatter search. The latter is a "reactive" approach, which seeks to explore efficiently the different operators developed. These GA´s were evaluated using test problems available in literature, for which have already been set lower limits. The results demonstrate the efficiency of using the technique of path-relinking as genetic operators of GA´s.

KEYWORDS: p-median problem, genetic algorithms; path-relinking; reactive algorithms.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

No clássico problema das p-medianas pretende-se localizar facilidades para melhor servir a clientes de forma a otimizar um certo critério (DREZNER, 1995). “Facilidades” é um termo genérico que pode ser substituído por postos de combustível, escolas, hospitais, indústrias, antenas de telecomunicações, etc. Já o termo “clientes” refere-se a motoristas, estudantes, pacientes, revendedores, receptores de ondas ou qualquer termo a ser servido pelas facilidades.

O problema das p-medianas é NP-difícil (GAREY; JOHNSON, 1979) e, desta forma, o tempo para se obter a solução ótima cresce exponencialmente à medida que se aumenta os dados de entrada (número *p* de facilidades e o número *n* de clientes). Na literatura científica são encontrados diferentes artigos que abordam o problema das p-medianas (e suas variações) utilizando heurísticas/metaheurísticas. Dentre eles estão: Teitz e Bart (1968) e Resende e Werneck (2003) que propuseram heurísticas baseadas em busca local; Heurísticas busca tabu foram propostas por Goncharov e Kochetov (2002) e Mladenović *et al*. (1996); Lorena *et al*., (1999), Fleszar e Hindi (2008) e Crainic *et al*. (2004) desenvolveram métodos VNS (*Variable Neighborhood Search*); algoritmos genéticos foram desenvolvidos por Alp *et al*. (2003) e Chaudhry *et al*. (2003); estratégias neurais foram propostas por Dominguez e Munoz (2008) e Domínguez Merino *et al*. (2003); Levanova e Loresh (2004) e García-López *et al*. (2003) propuseram, respectivamente, métodos baseados em colônia de formigas e *scatter search*.

A ligação cliente/facilidade requer grandes investimentos e possui despesas operacionais muito altas, principalmente no que se refere a distâncias percorridas em autovias e qualidade do sinal em telecomunicações. Por este motivo, uma redução no custo cliente/facilidade pode representar uma diferença considerável na despesa final às empresas que a custeiam. Visando

resolver esse problema, há alguns estudos na literatura para tentar diminuir ao máximo este custo, reduzindo assim as despesas e aumentando a qualidade desses serviços.

Dos trabalhos pesquisados na literatura científica, pode-se destacar: Teitz e Bart (1968) e Lorena *et al*. (1999). O primeiro apresenta uma heurística eficiente para o problema, a qual é usada, neste trabalho, na construção de soluções iniciais. No trabalho de Lorena *et al.* (1999), além de uma heurística para o problema, limites inferiores para um conjunto de problemas testes são apresentados. O objetivo deste trabalho foi desenvolver heurísticas eficientes que alcançasse resultados iguais ou melhores aos obtidos pelas heurísticas propostas por Teitz e Bart (1968) e Lorena *et al*. (1999). Para alcançar tais resultados, dentre diversos metodologias presentes na literatura, foi escolhida a metaheurística de algoritmos genéticos (AG). Utilizando-se desta metaheurística, foram realizadas modificações específicas para o tratamento do problema das p-medianas em sua estrutura, assim desenvolvendo uma heurística específica para o tratamento deste problema.

Esta pesquisa foi desenvolvida na linguagem C, a qual foi escolhida por ser comumente usada no meio científico e acadêmico bem como pelo seu desempenho com o uso de ponteiros, que proporciona uma estabilidade superior em tempo de processamento elevado às demais linguagens estruturadas.

Esta dissertação está organizada da seguinte forma:

- no Capítulo 2 será discutido detalhadamente o problema das p-medianas, apresentando dados relevantes, grafos e matrizes.

- no Capítulo 3 serão discutidos algumas soluções existentes na literatura científica para o problema das p-medianas. Os Algoritmos Genéticos serão discutidos detalhadamente neste capítulo onde serão apresentadas suas aplicações, história e funcionamento.
- no Capítulo 4 serão apresentado os AGs desenvolvidos, descrevendo as especificações que modificam o AG simples proposto por Holland (1975), visando o ganho de desempenho para o tratamento do problema das p-medianas;

- no Capítulo 5 serão apresentados os resultados computacionais obtidos pelos AGs desenvolvidos. Estes resultados serão avaliados sob um conjunto de instâncias descritas em Lorena *et al*., (1999), para as quais os limites inferiores são conhecidos; e

- as conclusões e os pontos para desenvolvimentos futuros serão enumerados no Capítulo 6.

2 DEFINIÇÃO DO PROBLEMA

O problema das p-medianas é um problema da área de teoria dos grafos. A seguir são descritas as definições referentes a grafos e ao problema das p-medianas.

2.1 GRAFOS

A teoria dos grafos é um ramo da matemática que estuda as relações entre os objetos abstratos de um determinado conjunto.
Grafo é uma estrutura *G(V,A)* onde:

- *V* é um conjunto não vazio de objetos denominados **vértices**; e
- *A* é um conjunto de pares não ordenados de *V*, chamados **arestas**.

Exemplo:

- *V* = {a, b, c, d, e}
- *A* = {(a, b), (a, c), (b, c), (b, d), (c, d), (c, e), (d, e)}

A representação gráfica de um grafo é feita por pontos (vértices) e linhas (arestas) unindo estes pontos conforme ilustra a Figura 1.

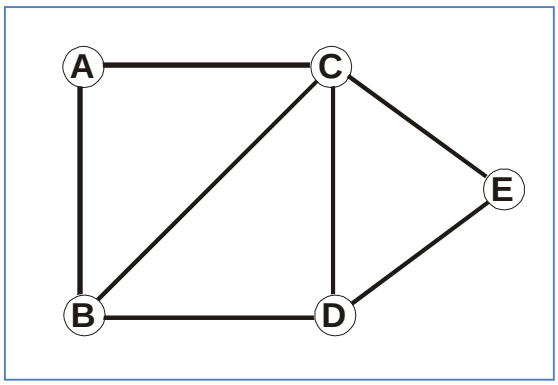

Figura 1: Grafo.

Dependendo da aplicação, arestas podem ou não ter direção; pode ser permitido ou não arestas ligarem um vértice a ele próprio; e vértices e/ou arestas podem ter um peso numérico associado.

Dos problemas que envolvem os estudos dos grafos pode-se citar: problema do caminho mínimo, problema do Caixeiro Viajante, problema das p-medianas dentre outros.

2.2 O PROBLEMA DAS P-MEDIANAS

Mediana ou centróide de um grafo *G(V, A)* é um vértice para o qual a soma das distâncias aos demais vértices é mínima. Existem problemas que têm como solução uma única mediana (um único vértice), chamadas de 1-mediana, porém, há casos em que existem mais de uma mediana como solução, chamados de 2-mediana, 3-mediana, ou p-mediana, de um modo geral.

No problema das p-medianas abordado neste trabalho, pretende-se localizar *p* facilidades (medianas) para melhor servir a clientes, de forma a minimizar o custo total.

Os dados relevantes para um problema das p-medianas são:

- um número finito de pontos (vértices) com valores conhecidos de demanda, denominados clientes ou pontos de demanda;
- um número finito de locais candidatos para a instalação de facilidades (medianas). Neste trabalho considera-se que os pontos de demanda (vértices) são candidatos para a instalação de facilidades;
- a distância entre cada ponto de demanda e os locais candidatos (arestas). Essas distâncias podem ser calculadas sobre a rede de caminhos que conectam os pontos (ex.: ruas, dutos e cabos) ou como distâncias euclidianas, ou seja, uma simples reta que liga dois pontos; e
- o número *p* de facilidades a serem instaladas.

Considerando *G(V,A)* um grafo não orientado, deve-se encontrar um conjunto de vértices $V_p \subseteq V$ (V_p é o conjunto das medianas do problema) com cardinalidade *p*, tal que a soma das distâncias de cada vértice pertencente a *V* (conjunto de clientes do problema) até seu vértice mais próximo em V_p seja a mínima possível.

Como um exemplo prático, é possível ver na Figura 2 um conjunto *V* onde todos os vértices são candidatos a mediana.

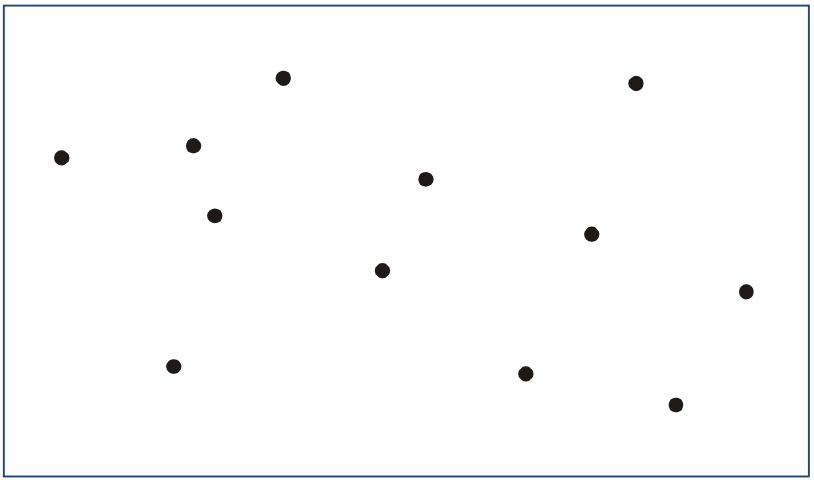

Figura 2: Conjunto de vértices candidatos.

Conforme ilustra a Figura 3, foram definidos aleatoriamente 2 vértices como medianas onde seu custo é de 768 milímetros. Este custo é calculado somando as distâncias de todos os vértices (clientes) à mediana (facilidade) mais próxima.

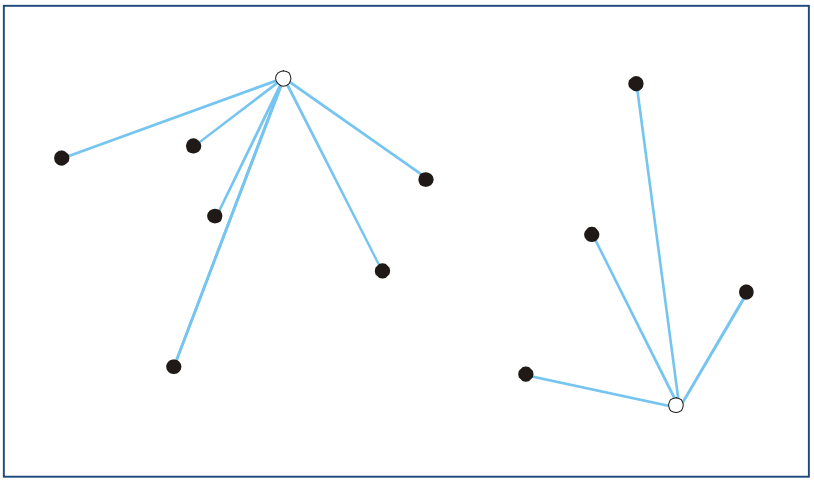

Figura 3: Medianas aleatórias.

Otimizando as 2 medianas no grafo proposto, foi possível diminuir o custo para 621 milímetros, conforme ilustra a Figura 4.

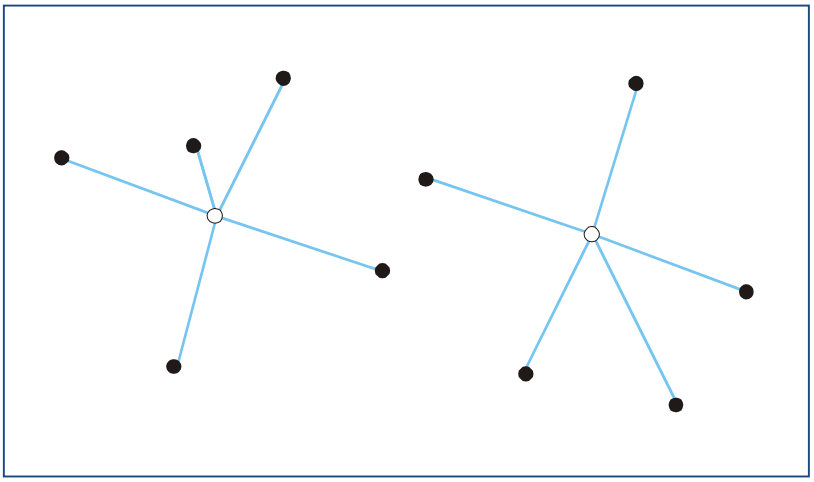

Figura 4: Medianas otimizadas.

3 DESENVOLVIMENTO DA SOLUÇÃO PROPOSTA

Problemas do tipo NP-difícil são considerados problemas intratáveis, onde o espaço de solução é tão amplo que pode-se considerar que o problema não pode ser resolvido através de métodos de solução exata. Em termos práticos, um problema é tratável se o seu limite superior de complexidade é polinomial, e é intratável se sua complexidade é exponencial, isto é, se seu tempo de execução é da ordem de uma função exponencial (TOSCANI; VELOSO, 2005). Assim, para tratar a maioria dos problemas do tipo NP-difícil, um dos meios mais viáveis é a utilização de alguma técnica de otimização que usa algoritmos exploratórios para encontrar a solução de problemas em buscas por aproximações sucessivas, até que o problema seja resolvido. Esses tipos de algoritmos são chamados de **heurística** (SOUZA, 2008).

Metaheurísticas são estratégias direcionadas à solução de problemas de otimização combinatória, as quais usam heurísticas genéricas que se adaptam facilmente às estruturas paralelas e são direcionadas à otimização global de um problema, podendo conter diferentes procedimentos heurísticos de busca local na solução a cada passo.

Existem na literatura diversas metaheurísticas que tratam de problemas do tipo NP-Difícil. Algumas das mais amplamente divulgadas são: Algoritmos Genéticos, *Simulated Annealing,* Busca Tabu, *GRASP,* Busca em Vizinhança Variável (VNS) e Colônia de Formigas.

Neste trabalho foi usada a metaheurística dos **Algoritmos Genéticos**, que é uma técnica de busca que utiliza procedimentos iterativos que simulam o processo de evolução de uma população de possíveis soluções de um determinado problema. O processo de evolução é aleatório, porém, guiado por um mecanismo de seleção baseado na adaptação de estruturas individuais “vetor de solução”. A cada iteração do algoritmo (uma geração), um novo

conjunto de estruturas é criado através da troca de informações (*bits* ou blocos) entre estruturas bem adaptadas selecionadas da geração anterior. Novas estruturas são geradas aleatoriamente com uma dada probabilidade e incluídas na população. O resultado tende a ser um aumento da adaptação de indivíduos ao meio, podendo acarretar também em um aumento global da aptidão da população a cada nova geração. Neste caso, a população evolui a cada geração aproximando-se de uma solução ótima (SOBRINHO, 2003).

Este capítulo é organizado da seguinte forma:

- na Seção 3.1 é apresentado a analogia e história dos algoritmos genéticos;
- na Seção 3.2 são descritos os conceitos de um AG; e
- na Seção 3.3 é descrito o ciclo de um AG básico.

3.1 ANALOGIA E HISTÓRIA DOS ALGORITMOS GENÉTICOS (AG)

Em meados do século XIX surgiu um dos mais importantes princípios no campo da evolução da vida dos organismos vivos, a teoria da Seleção Natural de Darwin (1859) defendia a ideia de que na natureza os seres vivos com melhores características, ou seja, os mais adaptados, tendem a sobreviver frente aos demais.

Baseado nestes princípios, em 1975 John Holland publicou o livro intitulado *"Adaptation in Natural and Artificial Systems"* onde formulou técnicas de busca, chamadas de Algoritmos Genéticos (AG), para utilização em processos de otimização e resolução de problemas (HOLLAND, 1975). Nos anos 80 David E. Golderg, um dos alunos de Holland, implementou com sucesso, pela primeira vez, está técnica numa aplicação industrial. Com este trabalho, publicou o livro *“Genetic Algorithms in Search, Optimization and Machine Learning”* uma das referências mais citadas na área de ciência da computação (GOLDBERG; DEB, 1989).

3.2 PRINCIPAIS CONCEITOS DE UM ALGORITMO GENÉTICO (AG)

Os principais conceitos de um AG são cromossomos, gene e população.

- **Cromossomo**: A representação do cromossomo é fundamental para o AG e consiste de uma maneira de modelar a informação do problema em um formato viável de ser tratado pelo computador. Em AGs, um cromossomo é uma estrutura de dados que representa uma das possíveis soluções do espaço de busca de um problema, o qual geralmente é representado por um vetor.

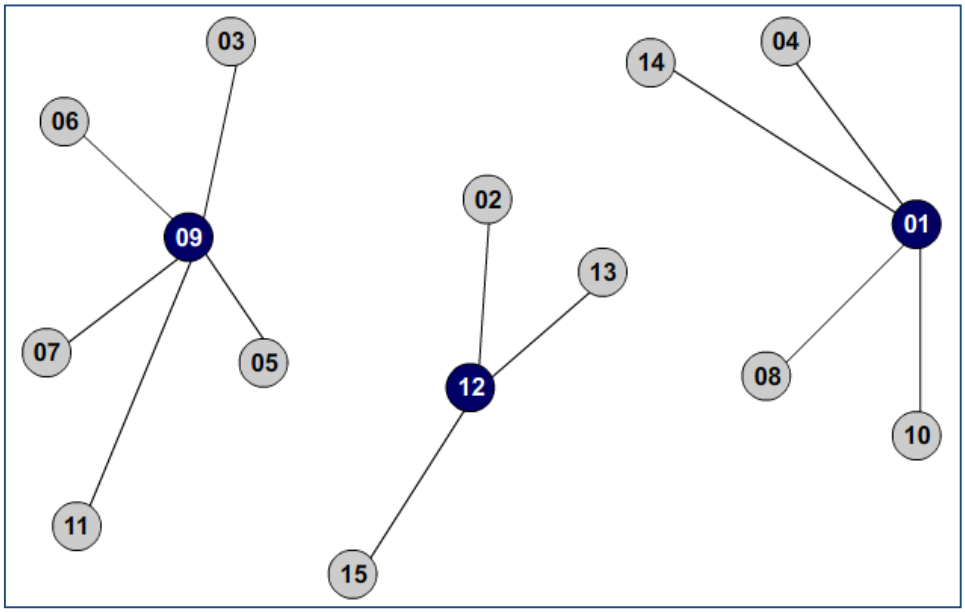

Figura 5: Representação gráfica do problema das p-medianas.

Como pode ser observado na Figura 5, os vértices escolhidos como medianas são 01, 09 e 12. Logo para se representar essa solução em forma de cromossomos em um AG, pode-se utilizar um vetor de 3 posições.

01	09	12

Figura 6: Vetor “cromossomo”.

Um outro modo de se representar esse cromossomo seria através de um vetor binário com o números de posições iguais ao número de vértices do grafo, onde 0 “zero” representaria um vértice a ser atendido por uma mediana e 1 “um” representaria a própria mediana, conforme observa-se na Figura 7.

1	0	0	0	0	0	0	0	1	0	0	1	0	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figura 7: Cromossomo com vetor binário.

- **Gene:** Um cromossomo é composto de genes, os quais descrevem a solução candidata. Os genes de um cromossomo podem ser compostos por uma *string* de *bits* “11011110” ou por outros valores “inteiros, reais ou caracteres” como ilustra a Figura 8.

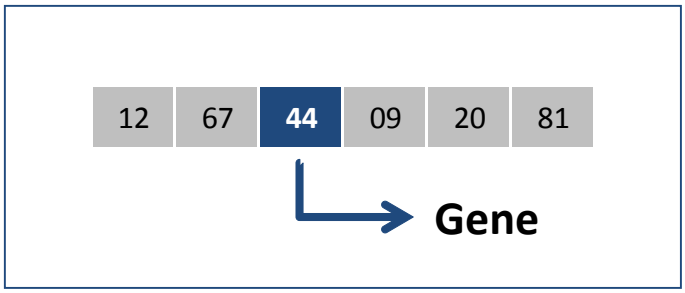

Figura 8: Gene destacado em um cromossomo.

- **População:** É o conjunto de todas as soluções “cromossomos” a serem armazenadas. Cada cromossomo deste conjunto representa uma solução possível. Cada uma das possíveis soluções é avaliada pelo seu valor de aptidão para o problema. A Figura 9 Ilustra ima população de 10 cromossomos

Cromossomo 01	54	41	38	88	71	4	74	54	30	25
Cromossomo 02	46	22	88	91	77	48	44	57	7	73
Cromossomo 03	64	9	94	49	9	18	63	32	100	44
Cromossomo 04	93	87	89	60	61	41	53	92	89	23
Cromossomo 05	11	83	87	34	8	92	14	8	85	3
Cromossomo 06	4	97	63	20	69	9	16	30	48	24
Cromossomo 07	53	46	21	90	91	47	71	7	35	1
Cromossomo 08	30	58	88	47	78	5	21	55	26	30
Cromossomo 09	2	50	77	23	29	26	35	36	12	87
Cromossomo 10	44	86	61	23	80	99	53	70	90	98

Figura 9: População de cromossomos.

3.3 CICLO BÁSICO DE UM ALGORITMO GENÉTICO

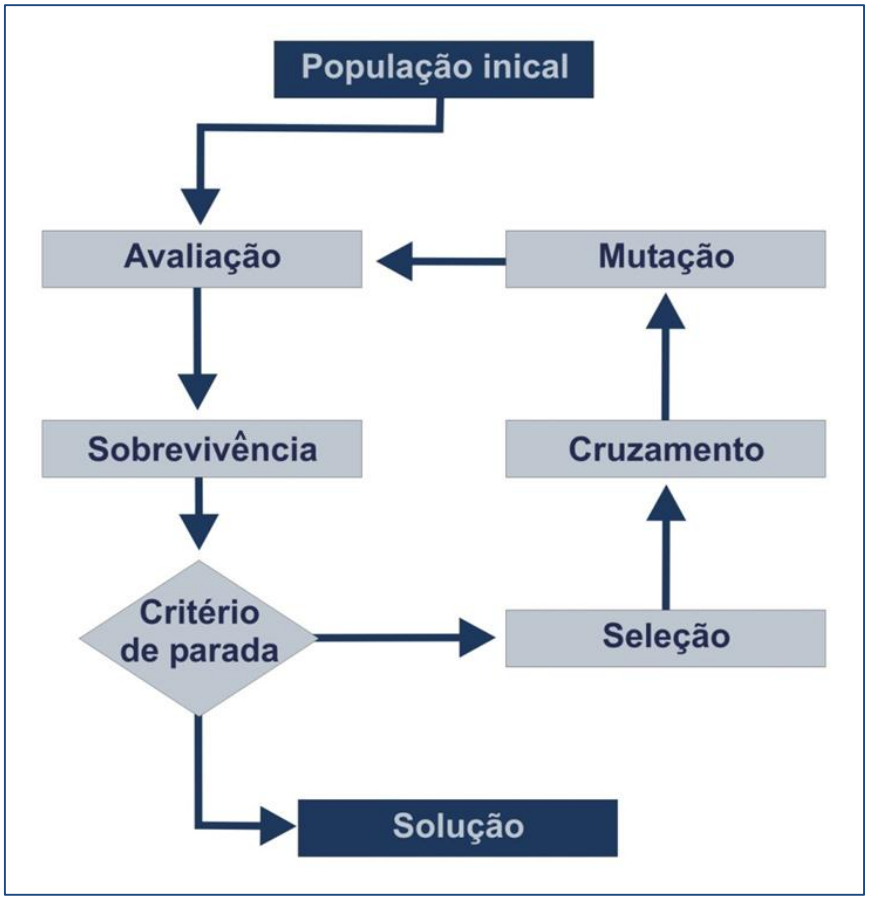

Figura 10: Fluxograma de um Algoritmo Genético básico.

Com referência ao diagrama da Figura 10, observa-se que cada iteração do Algoritmo Genético corresponde à aplicação de um conjunto de cinco operações básicas: avaliação, sobrevivência, seleção, cruzamento e mutação. Com exceção da primeira iteração, quando não é executado o operador de sobrevivência. Ao fim destas operações, cria-se uma nova população onde espera-se representar uma melhor aproximação da solução do problema de otimização que a população anterior.

3.3.1 População inicial

A população inicial é gerada atribuindo-se aleatoriamente valores aos genes de cada cromossomo ou usando algum tipo de inteligência artificial ou heurística para gerar uma população de melhor qualidade, seguindo um critério de aptidão que é medido através da função objetivo do problema de otimização.

3.3.2 Avaliação

Geralmente a avaliação de uma população é determinada através do cálculo da função objetivo, que depende das especificações de otimização. Cada cromossomo é uma entrada para uma função objetivo (algoritmo que calcula o desempenho do cromossomo), cuja saída fornece o valor de aptidão "*fitness*" do cromossomo.

3.3.3 Sobrevivência

Nesta etapa é definida a porcentagem de indivíduos substituídos em cada iteração do algoritmo genético ou a cada *N iterações* realizadas. (GOLDBERG; DEB, 1989). Dentre os métodos encontrados na literatura, destacam-se:

- ***Simple Genetic Algorithm* (SGA):** proposto por Holland, (1975) foi a primeira forma de implementação de AGs. Nesta, a população de uma geração é completamente substituída pela geração seguinte.

- ***Simple Genetic Algorithm* (SGA) com elitismo:** o elitismo é uma estratégia de implementação onde os *N* melhores indivíduos de cada geração são preservados para a geração seguinte (LINDEN, 2006). Isto garante que ao menos o(s) melhor(es) indivíduo(s) de uma nova geração seja(m) equivalente(s) ao melhor(es) que a geração anterior.

- ***Steady State Genetic Algorithm* (SSGA):** nesse algoritmo, usualmente, somente um ou dois descendentes são produzidos em cada geração. Após o cruzamento, são definidos os indivíduos da população que serão substituídos pelos novos descendentes (LO-ZANO; HERRERA; CANO, 2008).

3.3.4 Critério de parada

Na etapa de avaliação do critério de parada é verificado se o AG deve continuar ou não evoluindo sua população (GOLDBERG, 1989). Dentre os métodos encontrados na literatura, destacam-se:

- **Número de Iterações:** o número máximo de iterações é um parâmetro definido pelo usuário. Quando o AG alcança este valor, o algoritmo é finalizado.

- **Convergência da População:** o melhor valor encontrado para a função objetivo dentro da população é subtraído da média dos valores das

funções objetivo de todos os indivíduos da população. Se este cálculo não exceder um valor pré-determinado pelo usuário, a população convergiu e o algoritmo é finalizado.

- **Tempo de Computação:** neste método, o usuário define o tempo máximo de computação para a execução do algoritmo. Após decorrido este tempo, o algoritmo é finalizado (CHOU; PREMKUMAR; CHU, 1999).

- **Aptidão Alvo:** um valor alvo para a aptidão é definido pelo usuário. O algoritmo finaliza quando, em uma geração, o cálculo da função objetivo de um dos indivíduos da população resultar em um valor melhor ou igual a aptidão alvo pré-definida (RESENDE; RIBEIRO, 2003) (RIBEIRO; VIANNA, 2003).

3.3.5 Seleção

A ideia principal da seleção em um algoritmo genético é oferecer aos melhores indivíduos da população corrente a preferência para o processo de reprodução, permitindo que estes indivíduos passem as suas características às próximas gerações. Dentre os métodos de seleção encontrados na literatura, destacam-se:

- **Torneio:** neste método um subconjunto de indivíduos (normalmente um par) é escolhido aleatoriamente. Contudo, somente o indivíduo com a melhor aptidão é selecionado para participar do cruzamento (BLICKLE; THIELE, 1995).

- **Roleta:** cada indivíduo tem uma probabilidade de ser selecionado proporcional à sua aptidão. Para visualizar este método considere um círculo representando a população sendo dividido em setores, onde a área de cada setor é proporcional à aptidão de cada indivíduo. Este método também é denominado amostragem universal estocástica. A Figura 11 faz uma representação gráfica deste método de seleção, onde

ilustra 10 cromossomos e suas respectivas probabilidades de serem escolhidos para o cruzamento representado pelos seus setores no círculo.

Fitness - Cromossomo 01	1348
Fitness - Cromossomo 02	1725
Fitness - Cromossomo 03	1501
Fitness - Cromossomo 04	1114
Fitness - Cromossomo 05	1425
Fitness - Cromossomo 06	1692
Fitness - Cromossomo 07	1288
Fitness - Cromossomo 08	1420
Fitness - Cromossomo 09	1385
Fitness - Cromossomo 10	1718

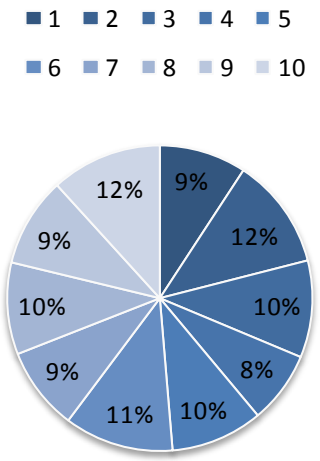

Figura 11: Método roleta de seleção.

3.3.6 Cruzamento

Uma das principais características dos algoritmos genéticos que os distinguem de outras técnicas de busca é o operador cruzamento (MITCHELL, 1996). Cruzamento é a troca de segmentos entre "casais" de cromossomos selecionados, com a finalidade de originar novos indivíduos que poderão ser incluídos na próxima geração. A ideia do cruzamento é a propagação das características dos indivíduos mais aptos da população. Dentre os métodos de cruzamentos encontrados na literatura, os mais conhecidos são:

- **Cruzamento por ponto de corte unico (*single-point crossover*):** neste método de reprodução um ponto de corte no cromossomo é escolhido de forma aleatória sobre a longitude do vetor que o representa e a partir desse ponto se realiza a troca de material cromossômico entre os dois indivíduos (HOLLAND, 1975). Na Figura 12 é possível ver um esquema da representação desse tipo de cruzamento, na qual o porto de corte foi definido após o quarto gene.

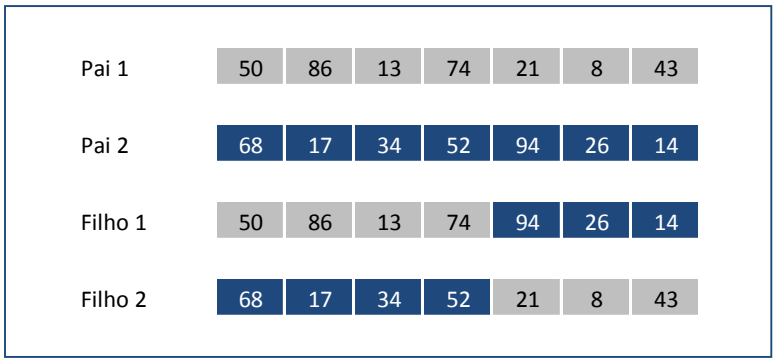

Figura 12: Cruzamento por ponto de corte.

- **Cruzamento uniforme (*uniform crossover*):** neste método cada gene do descendente é criado através da cópia de um gene dos pais, escolhido de acordo com uma máscara de cruzamento gerada aleatoriamente. Onde houver 0 na máscara de cruzamento, o gene correspondente será copiado do primeiro pai e onde houver 1 será copiado do segundo. O processo é repetido com os pais trocados para produzir o segundo descendente. Uma nova máscara de cruzamento é criada para cada par de pais. A Figura 13 demonstra graficamente o processo.

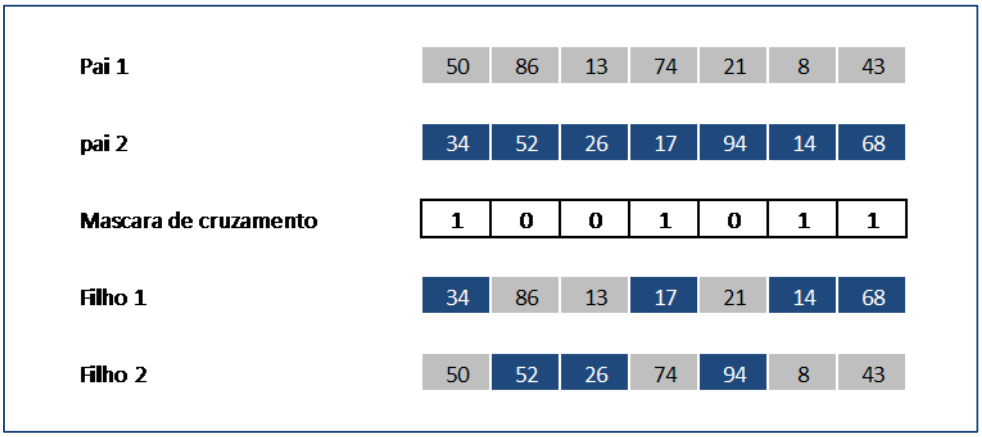

Figura 13: Cruzamento uniforme.

3.3.7 Mutação

A mutação é vista como o operador responsável pela introdução e manutenção da diversidade genética na população. Ela trabalha alterando arbitrariamente, logo após o cruzamento, um ou mais genes de um cromossomo filho escolhido aleatoriamente, fornecendo dessa forma meios para a introdução de novos elementos na população (HOLLAND, 1975). Na maioria dos casos o operador de mutação é aplicado aos indivíduos com uma probabilidade dada por uma taxa de mutação definida pelo usuário. A Figura 14 ilustra a representação gráfica do operador de mutação, onde através de uma máscara são selecionados os genes a serem trocados. Geralmente os novos valores atribuídos aos genes mutados são aleatórios, porem nada impede de haver algum processo de inteligência ou otimização nisso.

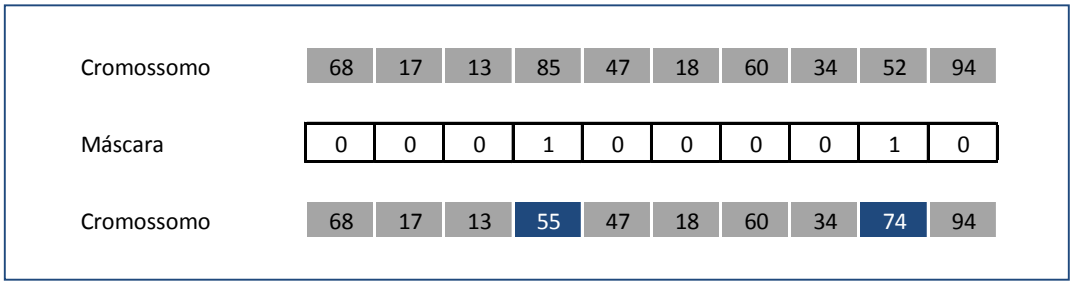

Figura 14: Mutação.

4 ESPECIFICAÇÕES DO ALGORITMO GENÉTICO (AG) DESENVOLVIDO

Para este trabalho, que trata unicamente o problemas das p-medianas, foram feitas diversas alterações na estrutura básica do algoritmo genético, descritas no capítulo anterior, a fim de otimizar seu desempenho, melhorando assim consideravelmente o resultado final. Foram feitas as seguintes modificações:

- Inserção do operador “Detecção de clones”;
- Modificação no operador de seleção;
- Adaptação e criação de novos operadores de cruzamento;
- Modificação no operador de mutação; e
- Adaptação no operador de sobrevivência.

4.1 PARÂMETROS DA METODOLOGIA UTILIZADA.

A teoria sobre AG não auxilia muito sobre a escolha de seus parâmetros, uma vez que tais configurações dependem do problema abordado. Usualmente são utilizados valores testados empiricamente (SOARES, 1997). Neste trabalho foram testados alguns parâmetros, onde utilizando de testes práticos e alguns estudos encontrados na literatura sobre o ajuste dos parâmetros do AG (JONG, 1975; GREFENSTETTE, 1986; GOLDBERG, 1989) foi definido tais parâmetros:

- **Tamanho da população *(numPop)*:** a fim de não ter uma amostragem insuficiente do espaço de busca com uma população muito pequena e não definir uma população muito grande tendo um tempo de cálculo computacional inaceitável, foi proposto para este trabalho uma população de 50 indivíduos.

- **Critério de parada:** foi adotado para este trabalho o critério de parada por tempo de execução, sendo atribuídos diferentes tempos para cada instancia tratada. Esses valores de tempos são dados em segundos, calculado pela expressão abaixo.

$$Tempo = \frac{n \times p}{4}$$

Onde *n* é o numero de vertices do grafo e *p* o numero de medianas.

- **Probabilidade de Mutação *(chanceMut)*:** A probabilidade de mutação indica a taxa em que haverá a mutação de cromossomos nas populações ao longo da evolução. Pequenos valores de *chanceMut* não proporcionam o devido aumento da diversidade populacional, limitando assim o espaço de busca. Por outro lado, um alto valor de *chanceMut* conduz os AGs à procura aleatória. Na literatura são encontrados valores como: 0,001 (JONG, 1975), 0,005 a 0,001 (SCHAFFER et ai, 1989) e 0,01 (GREFENSTETTE, 1986). Para este trabalho foi definido a probabilidade de mutação em 0,01 (1%).

- **Nível de Mutação (*nivelMutacao*):** Quando há uma mutação, ao invés de ser alterado um único gene no cromossomo serão alterados uma porcentagem de genes do cromossomo definido por *nivelMutacao.* Depois de alguns testes foi definido para este trabalho um nível de mutação de 10%, ou seja, 10% dos genes de um cromossomo serão alterados caso ocorra a mutação.

4.2 POPULAÇÃO INICIAL

Ao invés de gerar uma população inicial aleatória, foi proposto a esse trabalho o uso de um algoritmo construtivo de otimização para gerar cada cromossomo da população. Foi escolhido o algoritmo de Teitz And Bart (1968) devido ao seu bom desempenho.

4.2.1 Algoritmo de Teitz And Bart

Proposto por Teitz e Bart (1968), é um método aproximado para a determinação de *p* medianas entre *n* vértices candidatos à mediana do problema, baseado na substituição de vértices. Escolhe-se, inicialmente, *p* vértices aleatórios para formar um conjunto *S* inicial. *S* é considerado uma aproximação inicial de V_p (conjunto ideal de facilidades). Sendo *V* o conjunto de todos os vértices do grafo, verifica-se se pode haver um vértice v_i pertencente ao conjunto *V* - *S* que possa substituir um vértice v_j pertencente a S, produzindo um novo conjunto *S*' (*S*' = *S* + {v_i} - {v_j}), tal que o somatório das distâncias entre cada cliente (vértices de *V*) e a mediana mais próxima deste seja menor em *S*' do que em *S*. Se isto for possível, é feita a substituição de v_j por v_i e *S*' passa a ser a melhor aproximação, até o momento, para o conjunto V_p. O processo continua até obter um conjunto *S** onde nenhuma substituição de vértice de *S** por outro vértice em *V* – *S** produza melhora no somatório das distâncias entre os clientes e as medianas. Na Figura 15 é ilustrado o pseudocódigo do algoritmo de Teitz e Bart.

```
1.  TeitzAndBart(matDist[][],numMed,numVert,cromossomo[],fitnessMelhor)
2.  Entrada
3.      matDist[][] - matriz de distância
4.      numMed - número de medianas
5.      numVert – número de vértices do grafo
6.      cromossomo[] – uma solução construída
7.  Saida
8.      cromossomo[] – uma solução otimizada construída
9.  Início
10.     fitnessMelhor <- AcharFitness(matDist[][],numMed,numVert,
    cromossomo[]);
11.     croMelhor[] <- cromossomo[];
12.     Para i<-1 até numMed faça
13.         Para j<-1 até numVert faça
14.             cromossomo[i] <- j;
15.             fitness <- AcharFitness(matDist[][],numMed,numVert,
    cromossomo[]);
16.             Se fitness < fitnessMelhor
17.                 fitnessMelhor <- fitness;
18.                 croMelhor[i] <- cromossomo[i];
19.             Fim-se
20.         Fim-para
21.         cromossomo[i] <- croMelhor[i];
22.     Fim-para
23.     Retorne cromossomo[];
24. Fim-TeitzAndBart
```

Figura 15: Pseudocódigo do algoritimo de Teitz And Bart.

A Figura 15 apresenta o pseudocódigo do algoritmo de Teitz and Bart, o qual recebe como parâmetros de entrada a matriz de distâncias, o cromossomo a ser otimizado e outras variáveis necessárias para sua execução. Como saída, retorna o cromossomo otimizado por seu processo. Na linha 10 se obtém o *fitness* do cromossomo antes da sua otimização . O duplo laço nas linhas 12-22 faz com que cada posição do vetor seja trocada com todos os vértices; a cada troca, o procedimento de avaliação é chamado como se pode observar na linha 15. Sempre que há uma melhoria no desempenho, este estado é armazenado através da condicional na linha 16, assim otimizando a solução.

4.2.2 Gerando a população inicial

A população inicial é composta de *numPop* cromossomos, onde cada cromossomo é codificado como um vetor de *p* posições, nas quais são

armazenadas as medianas escolhidas através do algoritmo de Teitz And Bart, como se pode observar em seu pseudocódigo ilustrado na Figura 16.

```
1.  CriaGeracao(matDist[][],numMed,numPop,numVert,vetFitness[])
2.  Entrada
3.      matDist[][] - matriz de distância
4.      numMed - número de medianas
5.      numPop – número de cromossomos da população
6.      numVert – número de vértices do grafo
7.  Saída
8.      populacao[][] – matriz de cromossomos "soluções"
9.  Início
10.       Para i<-1 até numPop faça
11.             populacao[i][] <- CromossomoAleatorio(numMed,numVert);
12.           populacao[i][] <- TeitzAndBart(matDist[][],numMed,numVert,
    vetFitness[i]);
13.     Fim-para
14.     Retorne populacao[][];
15. Fim-CriaGeracao
```

Figura 16: Pseudocódigo do algoritmo gerador da população inicial.

A Figura 16 apresenta o pseudocódigo do algoritmo gerador da população inicial, o qual recebe como parâmetros a matriz de distâncias e outras variáveis necessárias para sua execução. Como saída, retorna a matriz de população do AG. O laço nas linhas 10-13 garante que o processo de criação dos cromossomos ocorra até que o número de indivíduos definidos pelo usuário seja atingido. Na linha 11 é criado um cromossomo com valores aleatórios e na linha 12 esse cromossomo é refinado através do algoritmo de Teitz and Bart.

4.3 AVALIAÇÃO

A avaliação ou aptidão do cromossomo é calculada através do somatório das distâncias euclidianas entre cada cliente e a mediana, pertencente à solução, que está mais próxima dele. A seguir é apresentado na Figura 17 o pseudocódigo que realiza esta avaliação em toda população.

```
1.  AcharFitness (matDist[][],numMed, numVert,cromossomo[])
2.  Entrada
3.      matDist[][] - matriz de distância
4.      numMed - número de medianas
5.      numVert – número de vértices do grafo
6.      cromossomo[] – uma solução construída
7.  Saída
8.      fitness – Fitness do cromossomo avaliado
9.  Início
10.     fitness <- 0;
11.     Para i<-1 até numVert faça
12.         menorDistancia <- maior distancia entre dois vértices;
13.         Para j<-1 até numMed faça
14.             Se menorDistancia > matDist[i][ cromossomo[j]] então
15.                 menorDistancia <- matDist[i][ cromossomo[j]];
16.             Fim-se
17.         Fim-para
18.         fitness <- fitness + menorDistancia;
19.     Fim-para
20. Retorne fitness;
21. Fim-AchaFitness
```

Figura 17: Pseudocódigo do algoritmo que calcula a aptidão.

A Figura 17 apresenta o pseudocódigo do algoritmo que calcula a aptidão, o qual recebe como parâmetros o cromossomo a ser avaliado, a matriz de distâncias e outras variáveis necessárias para execução deste operador. Como saída, retorna o *fitness* do cromossomo avaliado. O laço duplo nas linhas 11-19 tem a função de achar a distância entre um vértice do grafo à mediana mais próxima, realizando esta operação para todos os vértices do grafo e armazenando o seu somatório, como se pode observar na linha 18.

4.4 SELEÇÃO

Os pais são selecionados por um método conhecido como “roleta viciada” (GOLDBERG, 1989), no qual quanto melhor a avaliação do cromossomo, maior será a chance deste ser escolhido para o cruzamento. O cálculo da probabilidade $prob_i$ do cromossomo *i* da população *P* foi definido empiricamente. A equação a seguir mostra como é realizado o cálculo de $prob_i$:

$$prob_i = \left(f_i{}^{-4} \times 100\right) / \left(\sum_{i=1}^{numPop} f_i{}^{-4}\right)$$

onde f_i representa a avaliação ou aptidão do cromossomo *i* e *numPop* representa o número de cromossomos da população *P*.

É importante citar algumas observações:

- valor de aptidão deve ser sempre elevado a um número negativo para priorizar as menores avaliações, já que o que se quer é minimizar o custo x;
- elevando a -4 aumenta-se consideravelmente a chance dos melhores cromossomos serem escolhidos, daí o nome de roleta viciada para função, conforme pode-se observar no pseudocódigo ilustrado na Figura 18.

```
1.  RoletaViciada(vetFitness[],numPop)
2.  Entrada
3.      vetFitness[] – Vetor com os fitness da população
4.      numPop – número de cromossomos da população
5.  Saída
6.      vetPorcent[] – vetor com as porcentagens da roleta para os
    cromossomos da população
7.  Início
8.      somatorio <- 0;
9.      Para i<-1 até numPop faça
10.         somatorio <- somatorio + vetFitness[i]^-4;
11.     Fim-para
12.     Para i<-1 até numPop faça
13.         Se i = 1 então
14.             vetPorcent[i] <- vetFitness[i]^-4*100/ somatorio;
15.         Senão
16.             vetPorcent[i] <- vetFitness[i]^-4*100/ somatorio +
    vetPorcent[i-1];
17.         Fim-se
18.     Fim-para
19.     Retorne vetPorcent[];
20. Fim-RoletaViciada
```

Figura 18: Pseudocódigo da roleta viciada.

A Figura 18 apresenta o pseudocódigo da roleta viciada, o qual recebe como parâmetros o vetor de *fitness* e o número de indivíduos da população. Como saída ele retorna um vetor contendo as probabilidades de cada individuo da população a ser escolhido. O laço nas linhas 9-11 obtém o somatório de todos os *fitness* elevado a -4. O laço nas linhas 12-16 atribui a probabilidade de cada indivíduo ser escolhido para o cruzamento fazendo uma simples regra de três com os *fitness* elevados a -4.

4.5 CRUZAMENTO

O operador de cruzamento ou recombinação cria novos cromossomos através da combinação de dois ou mais cromossomos. A ideia intuitiva por trás do operador de cruzamento é a troca de informação entre diferentes soluções candidatas (VIANNA & RIBEIRO, 2004).

Com o intuito de avaliar diferentes operadores de cruzamento para o problema das p-medianas, foram implementados cinco operadores:

- Ponto de corte (GOLDBERG, 1989);
- Ponto mais próximo (desenvolvido neste trabalho);
- *Path relinking* aleatório (adaptado de RIBEIRO & VIANNA, 2003);

- *Path relinking* melhor (adaptado de Ribeiro e Vianna, 2003); e
- Reativo (proposto neste trabalho, adaptando a estratégia de Prais e Ribeiro, 2000, proposta para uma heurística GRASP).

Estes operadores serão descritos com mais detalhes nas próximas seções.

4.5.1 Ponto de corte

Baseado no clássico operador de ponto de corte (GOLDBERG, 1989), cada cromossomo pai é dividido em um ponto (chamado ponto de corte) definido aleatoriamente. Dois novos cromossomos são gerados permutando-se a metade inicial de um cromossomo pai com a metade final do outro. A seguir, é ilustrado na Figura 19 o pseudocódigo do cruzamento por pontos de corte.

A Figura 19 apresenta o pseudocódigo do cruzamento ponto de corte, o qual recebe como parâmetros os cromossomos pais definidos pelo operador de seleção, a matriz distância necessária para calcular o *fitness* dos cromossomos filhos e algumas variáveis e estruturas necessárias para realizar este método cruzamento. Como saída, este operador retorna o cromossomo filho gerado. Na linha 12 é definido aleatoriamente o ponto de corte, o laço nas linhas 13-21 realiza a troca de genes entre os cromossomos pais definida pelo ponto de corte, atribuindo esses genes aos filhos. Nas linhas 22 e 23 os cromossomos filhos são avaliados. Na condicional entre as linhas 24 a 30 é selecionado para entrar na população o cromossomo filho de melhor desempenho. A seguir na Figura 20 encontra-se uma ilustração dessa operação

```
1.  CrossoverPC(pai1[],pai2[],numMed,numVert,matDist[][],filho[],
    fitnessFilho)
2.  Entrada
3.      pai1[] – 1° cromossomo pai escolhido
4.      pai2[] – 2° cromossomo pai escolhido
5.      numMed - número de medianas
6.      numVert – número de vértices do grafo
7.      matDist[][] - matriz de distância
8.      fitnessFilho – Fitness do filho gerado
9.  Saída
10.     filho[] – filho gerado no cruzamento
11. Início
12.     corte <- numero aleatório compreendido entre 1 a (numMed-1);
13.     Para i<-0 até numMed faça
14.         Se corte <= i então
15.             filho1[i] <- pai1[i];
16.             filho2[i] <- pai2[i];
17.         Senão
18.             filho1[i] <- pai2[i];
19.             filho2[i] <- pai1[i];
20.         Fim-se
21.     Fim-para
22.     fitnessFilho1 <- AcharFitness(matDist[][],numMed,numVert,
    filho1[]);
23.     fitnessFilho2 <- AcharFitness(matDist[][],numMed,numVert,
    filho2[]);
24.     Se fitnessFilho1 < fitnessFilho2 então
25.         filho[] <- filho1[];
26.         fitnessFilho <- fitnessFilho1;
27.     Senão
28.         filho[] <- filho2[];
29.         fitnessFilho <- fitnessFilho2;
30.     Fim-se
31.     Retorne filho[];
32. Fim-CrossoverPC
```

Figura 19: Pseudocódigo do cruzamento ponto de corte.

.

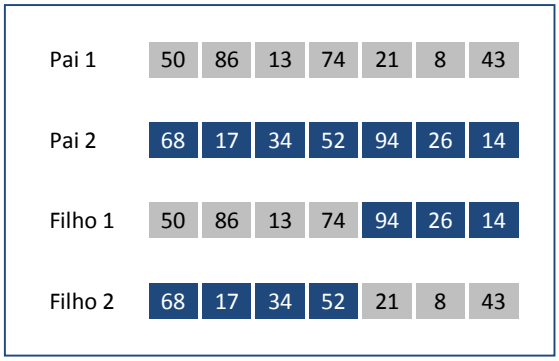

Figura 20: Ponto de corte.

4.5.2 Mediana mais próxima

Quando os cromossomos pais são selecionados, um deles é reorganizado (posição de seus genes são alteradas) posicionando os genes mais próximos dos seus respectivos no outro pai utilizando a distância entre os vértices. Em seguida uma máscara de cruzamento é gerada aleatoriamente. Onde houver 1 na máscara de cruzamento, o gene correspondente será copiado do primeiro pai e onde houver 0 será copiado do segundo. O segundo filho é gerado utilizando a mesma máscara, mas alterando a maneira que esta é interpretada: onde houver 1 na máscara, o gene correspondente será copiado do segundo pai e onde houver 0 será copiado do primeiro. Este operador de cruzamento foi desenvolvido pela primeira vez neste trabalho. A Figura 21 ilustra seu pseudocódigo.

A Figura 21 apresenta o pseudocódigo do cruzamento Ponto mais próximo, o qual recebe como parâmetros os cromossomos pais definidos pelo operador de seleção, a matriz distância necessária para calcular o *fitness* dos cromossomos filhos e algumas variáveis e estruturas necessárias para realizar este método cruzamento. Como saída, este operador retorna o cromossomo filho gerado. O duplo laço nas linhas 14-24 reorganiza o posicionamento dos genes de um dos pais os colocando na mesma posição do gene mais próximo através da matriz distância. No laço nas linhas 25-34 é criada uma máscara de cruzamento e realizada a troca de genes entre os pais. Nas linhas 35 e 36 os cromossomos filhos são avaliados. Na condicional entre as linhas 37 a 42 é selecionado para entrar na população o cromossomo filho de melhor desempenho.

A Figura 22 ilustra um exemplo de cruzamento utilizando este operador. Note que o cromossomo “pai 2” foi reorganizado, onde o gene do “pai 2” mais próximo do primeiro gene do “pai 1” é o gene de valor 34, o gene do “pai 2” mais próximo do segundo gene do “pai 1” é o gene de valor 52; e assim por diante.

```
1.  CrossoverPMP(pai1[],pai2[],numMed,matDist[][],numVert,cromossomo[],
    fitnessFilho)
2.  Entrada
3.      pai1[] – 1° cromossomo pai escolhido
4.      pai2[] – 2° cromossomo pai escolhido
5.      numMed – número de medianas
6.      matDist[][] – matriz de distância
7.      numVert – número de vértices do grafo
8.      cromossomo[] – uma solução construída
9.      fitnessFilho – Fitness do filho gerado
10. Saída
11.     filho[] – filho gerado no cruzamento
12. Início
13.     menorDist <- maior distancia entre dois vértices do grafo;
14.     Para i<-0 até numMed faça
15.         Para j<-0 até numMed faça
16.             Se matDist[pai1[i]][pai2[j]] < menorDist então
17.                 menorDist <- matDist[pai1[i]][pai2[j]];
18.                 pMenor <- j;
19.             Fim-se
20.         Fim-para
21.         aux <- pai2[i];
22.         pai2[i] <- pai2[pMenor];
23.         pai2[pMenor] <- aux;
24.     Fim-para
25.     Para i<-0 até numMed faça
26.         mascara <- número aleatório compreendido entre 0 e 1;
27.         Se mascara = 0 então
28.             filho1[i] <- pai1[i];
29.             filho2[i] <- pai2[i];
30.         Senão
31.             filho1[i] <- pai2[i];
32.             filho2[i] <- pai1[i];
33.         Fim-se
34.     Fim-para
35.     fitnessFilho1 <- AcharFitness(matDist[][],numMed,numVert,
    cromossomo[]);
36.     fitnessFilho1 <- AcharFitness(matDist[][],numMed,numVert,
    cromossomo[]);
37.     Se fitnessFilho1 < fitnessFilho2 então
38.         filho[] <- filho1[];
39.         fitnessFilho <- fitnessFilho1;
40.     Senão
41.         filho[] <- filho2[];
42.         fitnessFilho <- fitnessFilho2;
43.     Fim-se
44.     Retorne filho[];
45. fim-CrossoverPMP
```

Figura 21: Pseudocódigo do Cruzamento Ponto mais próximo.

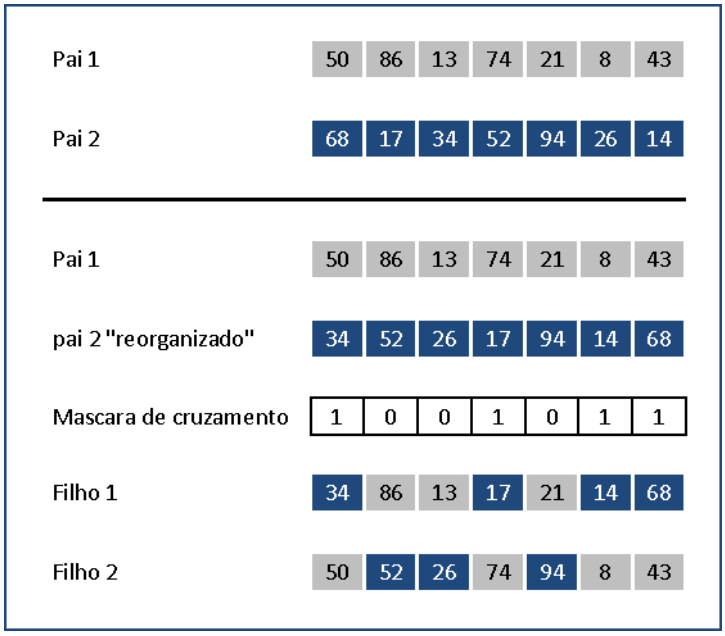

Figura 22: Ponto mais próximo.

4.5.3 *Path relinking* aleatório

O uso da técnica de reconexão por caminhos (*path relinking*) como operador de cruzamento foi proposta inicialmente por Ribeiro e Vianna (2003).

O intuito é explorar a trajetória entre dois cromossomos pais, retornando como filho o melhor indivíduo neste trajeto. Neste operador, dois cromossomos são selecionados como pais, sendo um deles considerado como “guia” e o outro como “partida” (que é o cromossomo corrente inicial). O objetivo é partir do cromossomo partida e chegar ao cromossomo guia por meio de substituição no cromossomo corrente de genes do cromossomo guia.

A cada passo, uma mediana (gene) g_i da solução corrente que não existe na solução guia é escolhida aleatoriamente para ser substituída por alguma mediana do cromossomo guia que se encontra na mesma posição no vetor. Será escolhida para substituir g_i aquela mediana presente no cromossomo guia que acarretar o menor custo no cromossomo corrente. Este processo é repetido até que o cromossomo corrente se iguale ao cromossomo guia.

É definido como filho do cruzamento o cromossomo intermediário de melhor avaliação no trajeto entre o cromossomo de partida e o guia.

A Figura 23 ilustra o pseudocódigo do cruzamento por *Path relinking* aleatório.

```
1.  CrossoverPRa(pai1[],pai2[],numMed,numVert,matDist[][],fitnessFilho)
2.  Entrada
3.      pai1[] – 1° cromossomo pai escolhido
4.      pai2[] – 2° cromossomo pai escolhido
5.      numMed - número de medianas
6.      numVert – número de vértices do grafo
7.      matDist[][] - matriz de distância
8.      fitnessFilho – Fitness do filho gerado
9.  Saída
10.     filho[] – filho gerado no cruzamento
11. Início
12.     fitnessPai1 <- AcharFitness(matDist[][],numMed,numVert,pai1[]);
13.     filho[] <- pai1[];
14.     guia[] <- filho[];
15.     fitnessfilho <- fitnessPai1;
16.     fitnessGuia <- fitnessfilho;
17.     Para i<-0 até numMed faça
18.         pos <- numero aleatório compreendido entre 1 a numMed sem
    repetir;
19.         guia[pos] <- pai2[pos];
20.         fitnessGuia <- AcharFitness(matDist[][],numMed,numVert,
    guia[]);
21.         Se i = 1 então
22.             filho[pos] <- guia[pos];
23.             fitnessfilho <- fitnessGuia;
24.         Senão
25.             Se fitnessfilho > fitnesGuia então
26.                 filho[pos] <- guia[pos];
27.                 fitnessfilho <- fitnesGuia;
28.             Fim-se
29.         Fim-se
30.     Fim-para
31. Retorne filho[];
32. Fim-CrossoverPRa
```

Figura 23: Pseudocódigo do Cruzamento *Path relinking* aleatório.

A Figura 23 apresenta o pseudocódigo do cruzamento *Path relinking* aleatório, o qual recebe como parâmetros os cromossomos pais definidos pelo operador de seleção, a matriz distância necessária para calcular o fitness de cromossomos intermediários entre os cromossomos de partida e guia e algumas variáveis e estruturas necessárias para realizar este método cruzamento. Como saída, este operador retorna o cromossomo filho gerado. Entre as linhas 13 e 16 são atribuídos valores de variáveis globais a variáveis locais para não alterar o conteúdo global dos ponteiros. O laço nas linhas 17-31 percorre todo cromossomo guia trocando genes escolhidos aleatoriamente no

cromossomo de partida; logo a cada iteração deste laço é formado um cromossomo intermediário que é avaliado através de um procedimento de avaliação presente na linha 20. A condicional na linha 25 garante o armazenamento do melhor cromossomo intermediário. A Figura 24 ilustra um exemplo deste tipo de cruzamento. Neste exemplo, o cromossomo definido como filho seria o cromossomo intermediário 5.

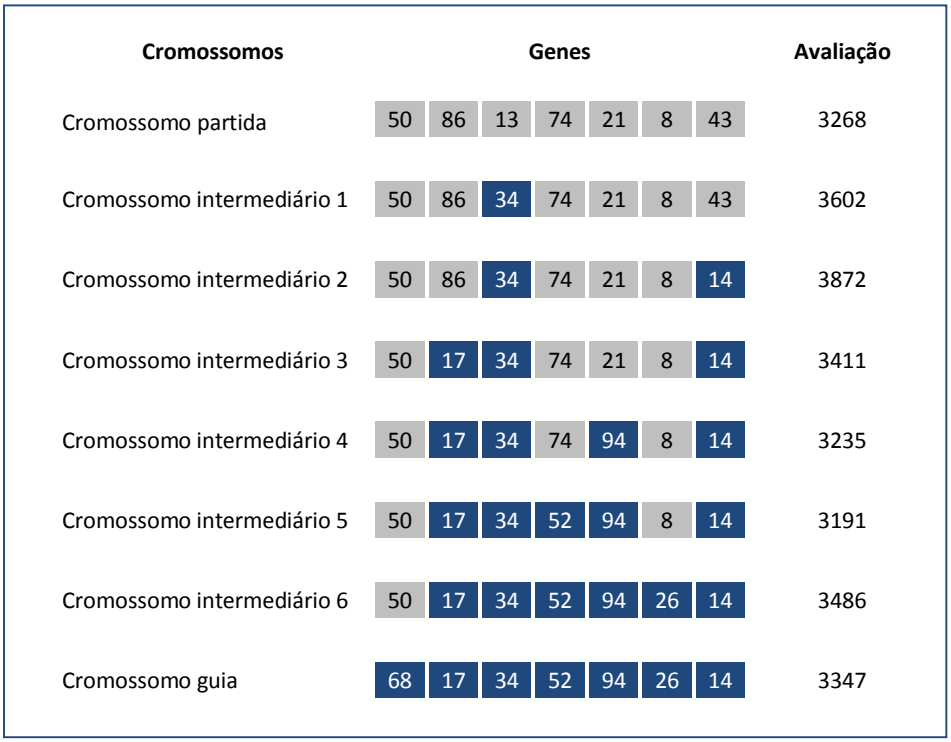

Figura 24: *Path relinking* aleatório.

4.5.4 *Path relinking* melhor

Funciona praticamente igual ao anterior. A única diferença é que ao invés de escolher aleatoriamente o gene no cromossomo corrente a ser substituído, este operador busca entre todos os genes do cromossomo corrente e do cromossomo guia aquele par que produzirá um cromossomo de menor custo. A seguir é ilustrado na Figura 25 o pseudocódigo do cruzamento por *Path relinking* melhor.

```
1.  CrossoverPRm(pai1[],pai2[],numMed,numVert,matDist[][],fitnessFilho)
2.  Entrada
3.      pai1[] – 1° cromossomo pai escolhido
4.      pai2[] – 2° cromossomo pai escolhido
5.      numMed - número de medianas
6.      numVert – número de vértices do grafo
7.      matDist[][] - matriz de distância
8.      fitnessFilho – Fitness do filho gerado
9.  Saída
10.     filho[] – filho gerado no cruzamento
11. Início
12.     fitnessPai1 <- AcharFitness(matDist[][],numMed,numVert,pai1[]);
13.     filho[] <- pai1[];
14.     guia[] <- filho[];
15.     fitnessfilho <- fitnessPai1;
16.     fitnessGuia <- fitnessfilho;
17.     Para i<-0 até numMed faça
18.         Para j<-0 até numMed faça
19.             guia[j] <- pai2[j];
20.             fitnessGuia <- AcharFitness(matDist[][],numMed,numVert,
    guia[]);
21.             Se j = 1 então
22.                 filho[j] <- guia[j];
23.                 fitnessfilho <- fitnessGuia;
24.             Senão
25.                 Se fitnessfilho > fitnesGuia então
26.                     filho[j] <- guia[j];
27.                     fitnessfilho <- fitnesGuia;
28.                 Fim-se
29.             Fim-se
30.         Fim-para
31.     Fim-para
32. Retorne filho[];
33. Fim-CrossoverPRm
```

Figura 25: Pseudocódigo do Cruzamento *Path relinking* melhor.

A Figura 25 apresenta o pseudocódigo do cruzamento *Path relinking* melhor, o qual recebe como parâmetros os cromossomos pais definidos pelo operador de seleção, a matriz distância necessária para calcular o *fitness* de cromossomos intermediários entre os cromossomos de partida e guia e algumas variáveis e estruturas necessárias para realizar este método cruzamento. Como saída, este operador retorna o cromossomo filho gerado. Entre as linhas 13 e 16 são atribuídos valores de variáveis globais a variáveis locais para não alterar o conteúdo global dos ponteiros. O duplo laço nas linhas 17-31 realiza a busca local nos genes com melhor desempenho, sendo avaliadas todas as combinações possíveis através da chamada do procedimento de avaliação na linha 20. A Figura 26 ilustra os possíveis passos gerados durante este método de cruzamento.

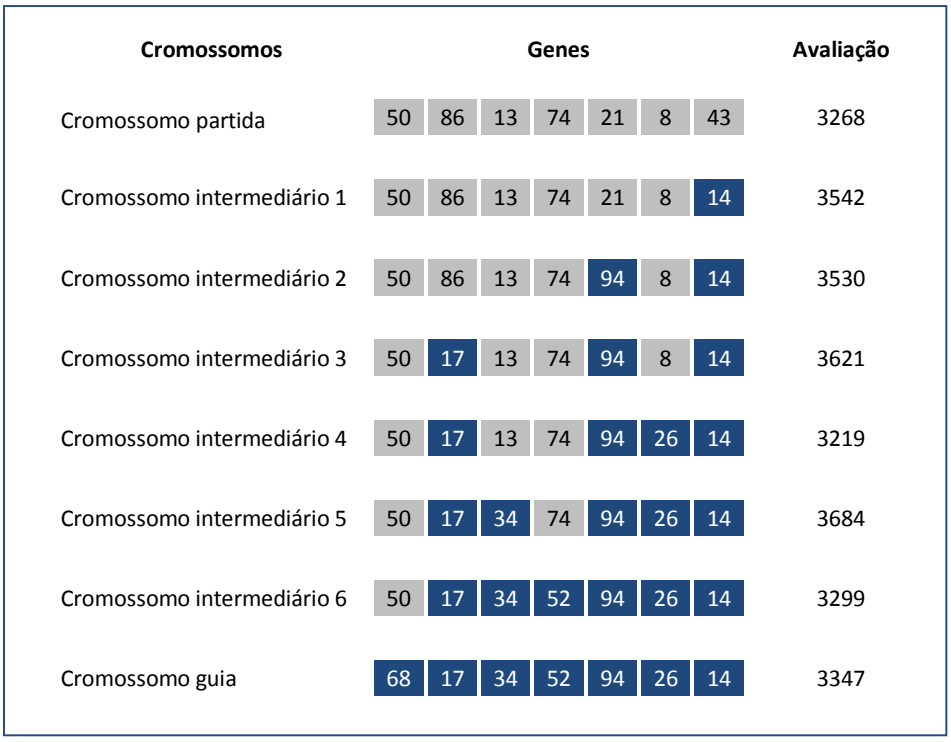

Figura 26: Path relinking melhor.

A cada criação de um cromossomo intermediário é realizada uma busca local para determinar o melhor gene a ser escolhido na criação do cromossomo intermediário, conforme mostra a Figura 27.

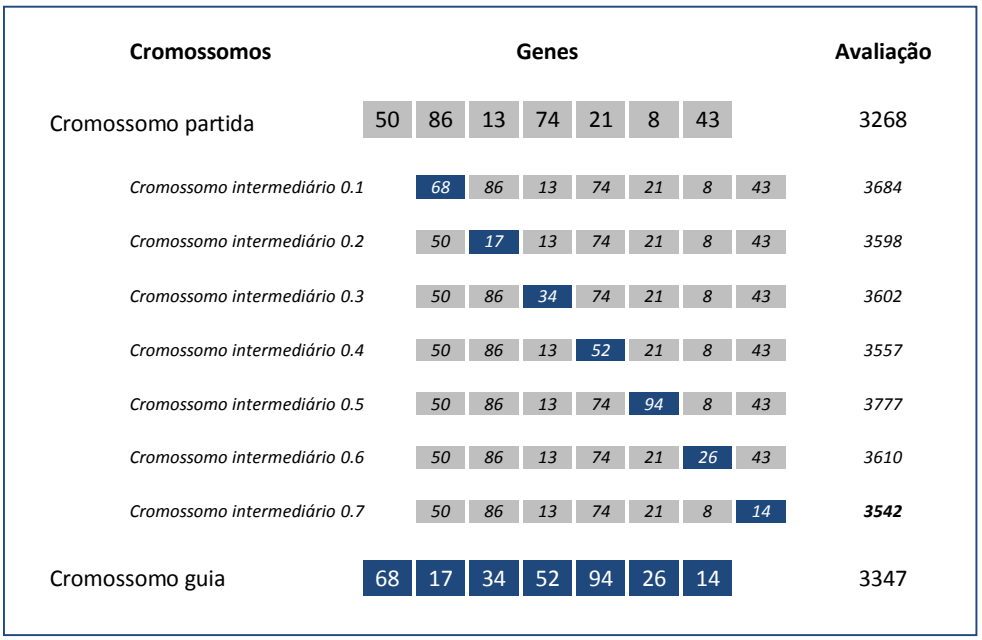

Figura 27: 1º Iteração do cruzamento *path relinking* melhor.

A cada iteração é escolhido o gene de melhor avaliação do cromossomo. As iterações dos cromossomos intermediários seguintes são feitas já com os melhores genes escolhidos nas iterações passadas conforme mostra a Figura 28.

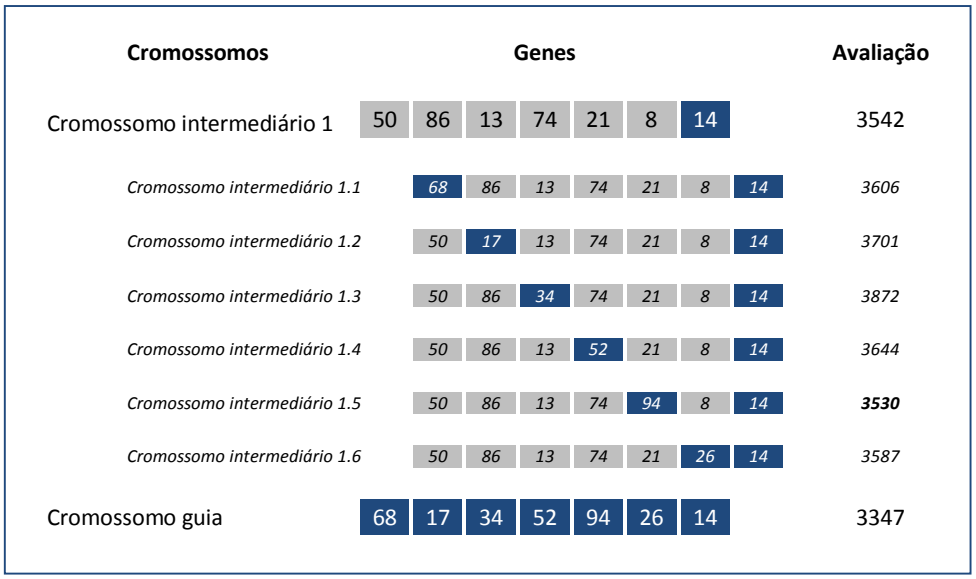

Figura 28: 2º Iteração do cruzamento *path relinking* melhor.

As iterações continuam sucessivamente até o cromossomo de partida se transformar no cromossomo guia. Neste tipo de cruzamento, o filho gerado é o cromossomo intermediário de melhor avaliação. No caso do exemplo mostrado na Figura 26, o cromossomo intermediário 4.

4.5.5 Reativo

O tipo de cruzamento reativo abrange todos os cruzamentos anteriormente citados, onde a cada iteração do AG um método de cruzamento diferente pode ser escolhido.

Cada tipo de cruzamento tem uma determinada probabilidade de ser escolhido, onde o valor desta chance se da conforme um valor de desempenho ditado pela expressão abaixo.

$$p_i = (q_i \times 100) / \left(\sum_{i=1}^{4} q_i \right)$$

Onde:

- p_i é a probabilidade do método de cruzamento *i* ser escolhido.
- q_i é o coeficiente do método de cruzamento *i* definido pela expressão abaixo.

$$q_i = (me/a_i)^{\propto}$$

Onde:

- *me* é o melhor *fitness* alcançado durante todas iterações anteriores.
- $\propto$ é uma constante definida para aumentar a probabilidade de escolha dos métodos que tem melhor desempenho no AG.
- a_i é a média dos *fitness* gerados pelo método de cruzamento *i* dada pela expressão abaixo.

$$a_i = \left(\sum_{i=1}^{n} f_i \right) / n$$

Onde:

- f_i é o valor do *fitness* referente a *i*.
- *N* é o número de vezes que o método de cruzamento *i* foi escolhido para o cruzamento.

$$x = \frac{numPop}{2}$$

Onde:

- *numPop é o numero de cromossomos na população.*

- *A cada x iterações, estes coeficientes são recalculados, alterando probabilidades de escolha de cada método de cruzamento. A Figura 29 ilustra o pseudocódigo do cruzamento pelo método* reativo.

```
1.  CrossoverReativo(pai1[],pai2[],numMed,matDist[][],numVert,cromossomo
    [],fitnessFilho,populacao[][],matDist[][],filho[],vetFitness[],
    numPop)
2.  Entrada
3.      populacao[][] – matriz de cromossomos "soluções"
4.      matDist[][] - matriz de distância
5.      numMed - número de medianas
6.      filho[] – Cromossomo gerado pelo ultimo cruzamento feito
7.      fitnessFilho – Fitness do ultimo filho gerado
8.      vetFitness[] – Vetor com os fitness da população
9.      pai1[] – 1° cromossomo pai escolhido
10.     pai2[] – 2° cromossomo pai escolhido
11.     numPop - Número de cromossomos da população
12.     numVert – número de vértices do grafo
13.     cromossomo[] – uma solução construída
14. Saída
15.     filho[] – Cromossomo gerado pelo ultimo cruzamento feito
16. Início
17.     sorteio <- Numero randômico compreendido entre 1 e 100;
18.     Alpha <- 10;
19.     Se sorteio <= p[1] então
20.         filho[] <- CrossoverPC(pai1[],pai2[],numMed,numVert,
    matDist,filho[], fitnessFilho);
21.         n[1] <- n[1] + 1;
22.         f[1] <- f[1] + fitnessFilho;
23.         a[1] <- f[1] / n[1];
24.         q[1] <- (me/a[1])^alpha;
25.     Senão
26.         Se sorteio <= p[2] então
27.             filho[] <- CrossoverPMP(pai1[],pai2[],numMed,
    matDist[][],numVert,cromossomo[],fitnessFilho);
28.             n[2] <- n[2] + 2;
29.             f[2] <- f[2] + fitnessFilho;
30.             a[2] <- f[2] / n[2];
31.             q[2] <- (me/a[2])^alpha;
32.           Fim-se
33.     Senão
34.         Se sorteio <= p[3] então
35.             filho[] <- CrossoverPRa(pai1[],pai2[],numMed,numVert,
    matDist[][],fitnessFilho);
36.             n[3] <- n[3] + 3;
37.             f[3] <- f[3] + fitnessFilho;
38.             a[3] <- f[3] / n[3];
39.             q[3] <- (me/a[3])^alpha;
40.           Fim-se
41.     Senão
42.         filho[] <- CrossoverPRm(pai1[],pai2[],numMed,numVert,
    matDist[][], fitnessFilho);
43.         n[4] <- n[4] + 4;
44.         f[4] <- f[4] + fitnessFilho;
45.         a[4] <- f[4] / n[4];
46.         q[4] <- (me/a[4])^alpha;
47.     Fim-se
48.     qTotal = q[1] + q[2] + q[3] + q[4];
49.     contador = contador + 1;
50.     Se contador > numPop/2 então
51.         contador <- 0;
52.         p[1] <- q[1]*100/ qTotal;
53.         p[2] <- q[2]*100/ qTotal + p[1];
54.         p[3] <- q[3]*100/ qTotal + p[2];
55.         p[4] <- q[4]*100/ qTotal + p[3];
56.     Fim-se
57.  Fim-FuncaoPrincipal
```

Figura 29: Pseudocódigo do Cruzamento Reativo.

A Figura 29 apresenta o algoritmo reativo, o qual recebe como parâmetros todas variáveis e estruturas necessárias para chamar qualquer operador de cruzamento proposto, e retorna como saída o cromossomo gerado pelo método de cruzamento escolhido. As condicionais entre as linha 19 a 47 fazem a seleção dos métodos de cruzamentos. A cada método de cruzamento escolhido são atualizadas as variáveis que definem as probabilidades de escolha de cada cruzamento calculadas entre as linhas 48 a 56.

4.6 MUTAÇÃO

A mutação ocorre em 1% das iterações. Quando isso ocorre 10% dos genes são trocados. A probabilidade de ocorrer a mutação é dada pela taxa de mutação *chanceMut*. Além da taxa de mutação outro fator importante é o nível de mutação *nivelMutacao*; o nível de mutação diz a quantidade de genes do cromossomo que serão afetados pela mutação. A Figura 30 ilustra o pseudocódigo do operador de mutação do AG.

```
1.  Mutacao(matDist[][],numMed,numVert,filho[],nivelMutacao,
    fitnessMelhor)
2.  Entrada
3.      matDist[][] - matriz de distância
4.      numMed - número de medianas
5.      numVert – número de vértices do grafo
6.      filho[] – Cromossomo gerado pelo ultimo cruzamento feito
7.      nivelMutacao – Porcentagem de alteração feita pela mutação
8.  Saída
9.      filho[] – Cromossomo gerado pelo ultimo cruzamento feito
10. Início
11.       genesMut <- inteiro(numMed * nivelMutacao);
12.       Para i<-1 até genesMut faça
13.           posicao <- numero aleatório compreendido entre 1 e numMed;
14.           filho[posicao] <- número aleatório entre 1 e numVert;
15.     Fim-para
16.     filho[] <- TeitzAndBart(matDist[][],numMed,numVert,filho[],
    fitnessFilho);
17. Retorne filho[];
18. Fim-Mutacao
```

Figura 30: Pseudocódigo do Algoritmo de mutação.

A Figura 30 apresenta o pseudocódigo do algoritmo de mutação, o qual recebe como parâmetros a matriz de distâncias, o cromossomo filho a ser mutado, o nível de mutação e outras variáveis necessárias para sua execução. Como saída, retorna o cromossomo filho mutado por seu processo. Na linha 11 é definido quantos genes do cromossomo serão mutados através do nível de mutação. O laço nas linhas 12-15 define uma posição aleatória no vetor para se sofrer a mutação onde este gene tem seu valor substituído por um valor aleatório. Na linha 16 é chamado o algoritmo de Teitz and Bart, otimizando o cromossomo mutado.

Na estratégia de mutação proposta é gerada uma máscara na qual onde houver 1, o gene correspondente será substituído por um vértice aleatório e onde houver 0 não há modificação, conforme ilustra a Figura 31. O percentual de 1 presente na máscara é definido por *nivelMutacao*.

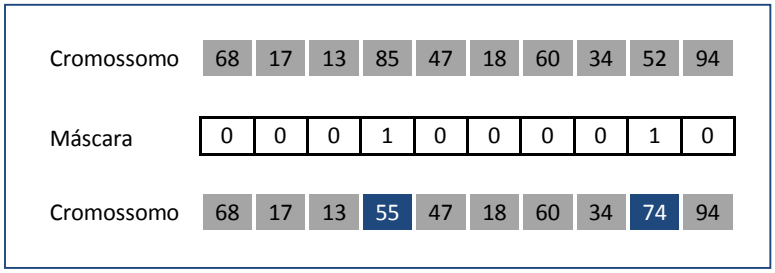

Figura 31: Mutação.

4.7 DETECÇÃO DE CLONES

É comum em AGs a população convergir ao longo das gerações para cromossomos idênticos (clones). Isso prejudica a diversidade da população tornando o AG pouco eficaz.

Este procedimento foi desenvolvido pele primeira vez neste trabalho para ser um operador do AG; sua função é fazer a detecção de cromossomos com a mesma aptidão e alterá-los. A Figura 32 ilustra o pseudocódigo do operador de detecção de clones do AG.

```
1.  DetectaClone(populacao[][],matDist[][],numMed, filho[],fitnessFilho,
    vetFitness[])
2.  Entrada
3.      populacao[][] – matriz de cromossomos "soluções"
4.      matDist[][] - matriz de distância
5.      numMed - número de medianas
6.      filho[] – Cromossomo gerado pelo ultimo cruzamento feito
7.      fitnessFilho – Fitness do ultimo filho gerado
8.      vetFitness[] – Vetor com os fitness da população
9.  Saída
10.     filho[] – Cromossomo gerado pelo ultimo cruzamento feito
11. Início
12.     Para i<-1 até numPop faça
13.         Se fitnessFilho = vetFitness[i] então
14.             nivelMutacao <- 0.3;
15.             filho[] <- Mutacao(matDist[][],numMed,numVert,filho[],
    nivelMutacao,fitnessFilho);
16.         Fim-se
17.     Fim-para
18. Fim-DetectaClone
```

Figura 32: Pseudocódigo do Algoritmo de detecção de clones.

A Figura 32 apresenta o pseudocódigo do algoritmo de detecção de clones, o qual recebe como parâmetros o *fitness* do filho gerado, o vetor de *fitness* da população, e outras variáveis necessárias para sua execução. Como saída, retorna o cromossomo filho alterado ou não. O laço nas linhas 12-17 percorre o vetor de *fitness* procurando por *fitness* idênticos ao do filho como se pode observar na condicional da linha 13. Caso seja detectado um cromossomo clone, o nível de mutação é setado em 30% e é chamado o procedimento de mutação.

Através de experimentos computacionais, em mais de 99,5% dos testes foi constatado que cromossomos que possuíam a mesma aptidão eram idênticos. Assim através deste operador, quando um clone é detectado, este é mutado em 30% dos seus genes e otimizado a partir do algoritmo de Teitz and Bart.

4.8 SOBREVIVÊNCIA

Depois de exaustivos testes com alguns modelos para este operador de AG encontrados na literatura, foi definida para este trabalho uma variação do modelo *Steady State Genetic Algorithm* (SSGA) (CEDENO; VEMURI; SLEZAK, 1995) (SYSWERDA, 1989).

Neste modelo, somente um ou dois descendentes são produzidos em cada geração. Após o cruzamento são definidos os indivíduos da população que serão substituídos pelos novos descendentes. Segundo esta base, foram definidas as seguintes regras para a evolução da população:

- Apenas o filho com melhor aptidão poderá concorrer a ser inserido na população;
- Caso o filho tenha uma aptidão melhor que um dos pais, ele poderá ser inserido na população; e
- Caso o filho seja inserido, deverá entrar no lugar do pai com a pior aptidão, excluindo-o da população.

A Figura 33 ilustra o pseudocódigo do operador de sobrevivência definido.

```
1.  Eliminacao(fitnessFilho,populacao[][],vetFitness[],fitnessPai1,
    fitnessPai2,numPop,filho[])
2.  Entrada
3.      fitnessPai1 – Fitness do pai 1
4.      fitnessPai2 – Fitness do pai 2
5.      fitnessFilho – Fitness do ultimo filho gerado
6.      populacao[][] – matriz de cromossomos "soluções"
7.      vetFitness[] – Vetor com os fitness da população
8.      numPop – número de cromossomos da população
9.      filho[] – Cromossomo gerado pelo ultimo cruzamento feito
10. Saída
11.     populacao[][] – matriz de cromossomos "soluções"
12. Início
13.     Se fitnessPai1 < fitnessPai2 então
14.         elimina <- fitnessPai2;
15.     Senão
16.         elimina <- fitnessPai1;
17.     Fim-se
18.     Se fitnessFilho < elimina então
19.         Para i<-1 até numPop faça
20.             Se elimina = vetFitness[i] então
21.                 pos <- i;
22.             Fim-se
23.         Fim-para
24.         vetFitness[pos] <- fitnessFilho;
25.         populacao[pos][] <- filho[];
26.     Fim-se
27.     Retorne populacao[][];
28. Fim- Eliminacao
```

Figura 33: Pseudocódigo do operador de sobrevivência.

A Figura 33 apresenta o pseudocódigo do algoritmo de sobrevivência, o qual recebe como parâmetros o *fitness* do filho gerado, o *fitness* dos pais do último cruzamento, o vetor de *fitness* da população, a matriz da população e outras variáveis necessárias para sua execução. Como saída, retorna a matriz da

população alterada ou não. As condicionais entre as linhas 13 e 18 definem se o cromossomo filho tem o fitness melhor que um dos seus pais. Caso tenha, o laço entre as linhas 19 e 23 buscam a localização deste pai no vetor de *fitness*, e nas linhas 24 e 25 este pai é substituído pelo cromossomo filho de melhor desempenho.

5 TESTES E RESULTADOS COMPUTACIONAIS

Nesse capítulo são apresentados os resultados para os testes realizados com as diferentes variações de algoritmos genéticos propostas neste trabalho.

Todos os experimentos computacionais deste trabalho foram realizados em um microcomputador equipado com processador Intel Core 2 Duo CPU E4500 com *clock* de 2.20GHz e 2.0 Gb de memória RAM sob a plataforma Windows ™ XP.

Todos os algoritmos genéticos foram desenvolvidos utilizando a linguagem de programação C, compilados no ambiente Dev-C++ versão 4.9.9.2 – IDE para programação de executáveis Win32, console ou GUI na linguagem C/C++. A fim de melhorar o desempenho do software, todas variáveis tratadas pelo algoritmo genético foram criadas dinamicamente com o uso de ponteiros.

A Tabela 1 mostra as instâncias utilizadas durante os testes, as quais foram as mesmas instâncias utilizadas por Lorena *et al*.(1999), para as quais já foram definidos os limites inferiores.

Tabela 1: instâncias e medianas

(n)	(p)
324	5
324	10
324	20
324	50
324	108

(n)	(p)
818	5
818	10
818	20
818	50
818	100
818	150
818	272

5.1 EXPERIMENTOS REALIZADOS

Foram desenvolvidos neste trabalho cinco algoritmos genéticos (AGs), os quais se diferenciam pelo operador genético de cruzamento utilizado. A seguir são apresentadas as nomenclaturas de cada AG, assim como o operador genético que este utiliza. É importante lembrar que todos os AGs desenvolvidos possuem a estrutura descrita no Capítulo 3.

- AGPC - Algoritmo genético com cruzamento “ponto de corte”.
- AGPMP - Algoritmo genético com cruzamento “ponto mais próximo”.
- AGPRa - Algoritmo genético com cruzamento “*path relinking* aleatório”.
- AGPRm - Algoritmo genético com cruzamento “*path relinking* melhor”.
- AGR - Algoritmo genético com cruzamento “Reativo”.

Foram realizados 3 tipos de experimentos, os quais são apresentados nas Subseções 4.2, 4.3 e 4.4.

- **Desempenho por execução:** Este teste foi aplicado dez vezes para cada uma das instâncias e cada um dos métodos de cruzamentos propostos neste trabalho. Sua finalidade é fazer um comparativo de desempenho entre os métodos propostos neste trabalho com o limite inferior e o trabalho de Lorena *et al*. (1999);

- **Nuvem de pontos:** este teste foi aplicado em uma única instância para os cinco métodos de cruzamentos propostos. Sua finalidade é observar o comportamento da população durante as gerações percorridas, onde se pode analisar o grau de heterogeneidade e convergência da população ao decorrer das gerações;

- **Aptidão alvo:** assim como o teste “nuvem de pontos”, este teste foi aplicado em uma única instância para os cinco métodos de cruzamentos propostos. Sua finalidade é medir o desempenho de cada AG proposto em decorrer do seu tempo de execução.

5.2 DESEMPENHO POR EXECUÇÃO

No experimento realizado, cada Algoritmo Genético realizou 10 (dez) execuções para cada uma das 12 (doze) instâncias descritas na Tabela 1. As Tabelas 2 e 3 apresentam os melhores resultados obtidos para as instâncias com *n* = 324 e *n* = 818, respectivamente, onde *n* é o numero de vértices de cada instância. Na coluna 1 é apresentado o número *p* de medianas. O limite inferior definido em (LORENA *et al*., 1999) para cada instância é descrito na coluna 2. Nas colunas 3 e 4 são apresentados, respectivamente, os melhores resultados obtidos por Lorena *et al*. (1999) e o *gap* (diferença percentual entre o resultado obtido e o limite inferior). Nas colunas seguintes são apresentados o custo médio e o *gap* obtidos por cada AG desenvolvido. Foram destacados em negrito os melhores resultados (custos) obtidos para cada instância.

O critério de parada para este teste é por tempo, descrito na Seção 4.1

	Gap	AGR	Gap
02	0,000	**122518,02**	0,000
34	0,007	**79256,34**	0,007
34	0,070	54505,34	0,070
35	1,325	32188,67	0,295
9	1,050	18926,70	1,106

	Gap	AGR	Gap
81	0,161	**605855,81**	0,161
25	0,540	384489,11	0,541
8	0,124	251874,14	0,133
50	0,487	147096,22	0,542
31	1,931	99314,01	1,586
55	2,361	77699,45	2,960
24	3,466	49521,66	4,297

Analisando os resultados apresentados nas Tabelas 2 e 3, percebe-se que o algoritmo AGPRa, que utiliza o operador de cruzamento “*path relinking* aleatório”, foi o que alcançou os melhores resultados, obtendo o melhor custo médio para todas as instâncias analisadas. O algoritmo AGPMP, que utiliza o operador de cruzamento “ponto mais próximo”, também atingiu bons resultados, obtendo o melhor custo médio para 7 das 12 instâncias. O algoritmo AGPRm, que utiliza o operador de cruzamento “*path relinking* melhor”, não obteve os resultados esperados. Isso ocorreu pois o operador de cruzamento “*path relinking* melhor” exige maior esforço computacional, o que acarreta em um menor número de gerações executadas pelo algoritmo AGPRm dentro do tempo estabelecido (critério de parada). O método AGR, que utiliza o operador de cruzamento “reativo”, assim como o AGPRm, não obteve os resultados esperados. Isso ocorreu devido a grande alternância de métodos de cruzamento, onde percebeu-se que no AGR é necessário um certo número de iterações em um único método de cruzamento para haver evolução da população. O algoritmo AGPC, que utiliza o operador de cruzamento “ponto de corte”, foi o que obteve os piores resultados entre os AGs desenvolvidos, o que já era esperado pela simplicidade do operador.

As Tabelas 4 e 5 apresentam os resultados médios obtidos para $n = 324$ e $n = 818$.

Tabela 4: Instância de n = 324 vértices.

P	LInf	Lorena	Gap	AGPC	Gap	AGPMP	Gap	AGPRa	Gap	AGPRm	Gap	AGR	Gap
5	**122518,02**	**122518,02**	0,000	**122518,02**	0,000	**122518,02**	0,000	**122518,02**	0,000	**122518,02**	0,000	**122518,02**	0,000
10	**79250,84**	**79256,35**	0,007	**79256,34**	0,007	**79256,34**	0,007	**79256,34**	0,007	**79256,34**	0,007	**79256,34**	0,007
20	**54467,23**	54533,11	0,121	54998,97	0,976	**54505,34**	0,070	**54505,34**	0,070	54644,25	0,325	54747,21	0,514
50	**32094,13**	**32101,52**	0,023	32679,49	1,824	32230,44	0,425	**32101,52**	0,023	32740,52	2,014	32565,08	1,467
108	**18719,7**	19683,61	5,149	18950,09	1,231	18812,35	0,495	**18732,59**	0,069	18970,56	1,340	18973,42	1,355

Tabela 5: Instância de n = 818 vértices.

P	LInf	Lorena	Gap	AGPC	Gap	AGPMP	Gap	AGPRa	Gap	AGPRm	Gap	AGR	Gap
5	**604883,69**	**605855,81**	0,161	605864,22	0,162	**605855,81**	0,161	**605855,81**	0,161	**605855,81**	0,161	**605855,81**	0,161
10	**382420,75**	385371,44	0,772	384513,75	0,547	**384341,81**	0,502	**384341,81**	0,502	384514,12	0,547	384546,19	0,556
20	**251540,45**	251717,77	0,070	251956,15	0,165	**251720,59**	0,072	**251719,59**	0,071	251897,07	0,142	251925,35	0,153
50	**146303,64**	149251,13	2,015	147588,59	0,878	147178,78	0,598	**147019,06**	0,489	147192,92	0,608	147273,76	0,663
100	**97763,44**	98992,31	1,257	99797,66	2,081	99067,54	1,334	**98353,40**	0,603	99776,48	2,059	99470,91	1,747
150	**75465,67**	77440,57	2,617	77799,58	3,093	76701,52	1,638	**76641,30**	1,558	77538,81	2,747	77779,46	3,066
272	**47481,36**	50086,61	5,487	49710,41	4,695	48584,74	2,324	**48529,57**	2,208	49390,81	4,021	49666,93	4,603

As Tabelas 6 e 7 apresentam os tempos médios obtidos por cada AG desenvolvido para as instâncias com *n* = 324 e *n* = 818, respectivamente. Os valores mostrados na Tabela é o tempo gasto em segundos até se encontrar a melhor solução daquela execução. Percebe-se que o algoritmo AGPRa, além de obter os melhores custos médios, obteve também, em geral, os melhores tempos médios.

Tabela 6: Instância de *n* = 324 vértices.

(p)	AGPC	AGPMP	AGPRa	AGPRm	AGR
5	1	**0**	**0**	**0**	**0**
10	58	3	**1**	24	20
20	132	22	**18**	93	108
50	321	53	**42**	224	274
108	674	111	**89**	475	572

Tabela 7: Instância de *n* = 818 vértices.

(p)	AGPC	AGPMP	AGPRa	AGPRm	AGR
5	42	2	**0**	37	27
10	157	28	**19**	168	204
20	342	267	**210**	295	319
50	859	671	**516**	738	790
100	1699	1325	**1002**	1425	1585
150	2541	1922	**1486**	2137	2348
272	4918	3547	**2910**	4022	4488

5.3 NUVEM DE PONTOS

Este teste faz uma representação gráfica da evolução da população em um algoritmo genético, evidenciando o teste de heterogeneidade entre os cromossomos da população durante o decorrer das gerações. Um AG tem um melhor desempenho quando há convergência da população preservando também certo grau de heterogeneidade da população ao decorrer das gerações.

O eixo X mostra as iterações decorridas durante o teste e o eixo Y mostra o *fitness* alcançado em decorrência a estas iterações.

São apresentados nesta seção os resultados computacionais dos cinco AGs propostos neste trabalho para o tratamento do problema das p-medianas.

No experimento realizado, cada algoritmo genético (AG) realizou 10.000 (dez mil) iterações. Para tornar mais evidente as diferenças de desempenho entre métodos de cruzamentos distintos, foi escolhida a instância com 818 vértices e 50 medianas, por ser uma instância com alto grau de dificuldade, porém com um tempo computacional viável para o teste de nuvem de pontos proposto. Lembrando que a população é composta por 50 indivíduos “cromossomos” como descrito na Seção 3.4.1

- **AGPC:** dentre os cinco métodos propostos, o AGPC foi o que obteve o pior desempenho de otimização não conseguindo atingir o *fitness* de 147.000. Porém foi o método que manteve a população mais heterogênea durante as gerações. Neste caso não há muita soluções convergindo para os melhores resultados, conforme ilustra a Figura 34.

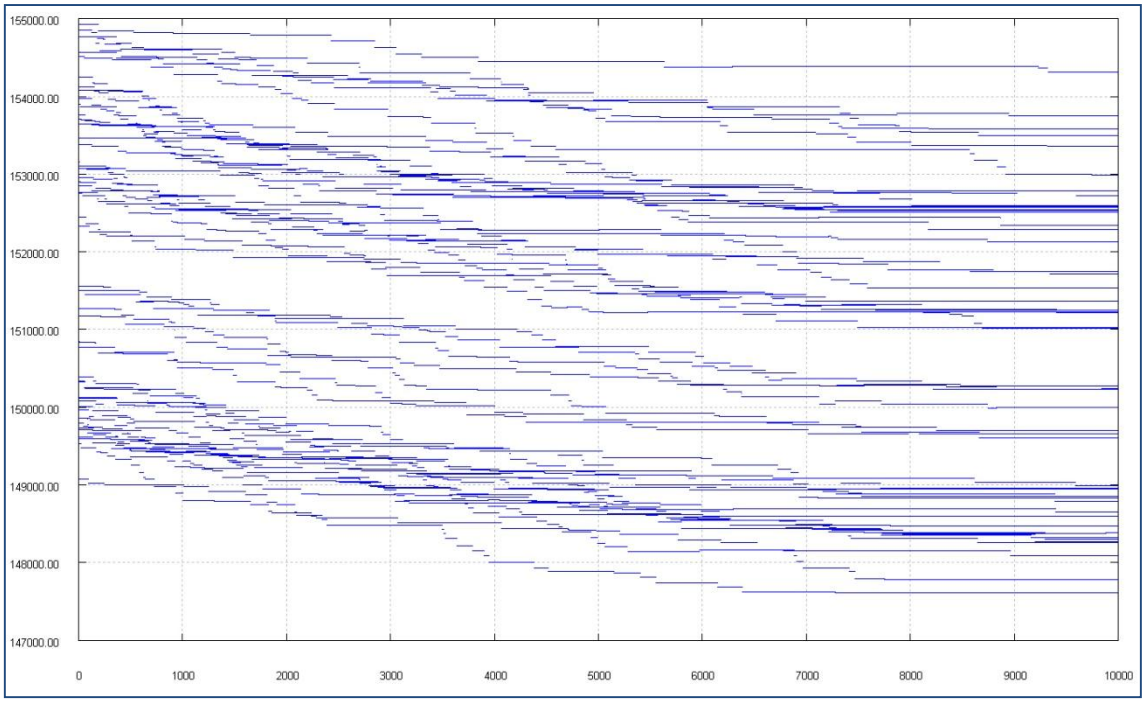

Figura 34: Nuvem de pontos para o AGPC.

- **AGPMP:** obteve o segundo melhor desempenho dentre os cinco métodos testados, conseguindo baixar o *fitness* a menos que 147.000. Observa-se que apesar de atingir um *fitness* muito baixo, o AGPMP consegue manter a população bastante heterogênea durante as gerações, conforme ilustra a Figura 35.

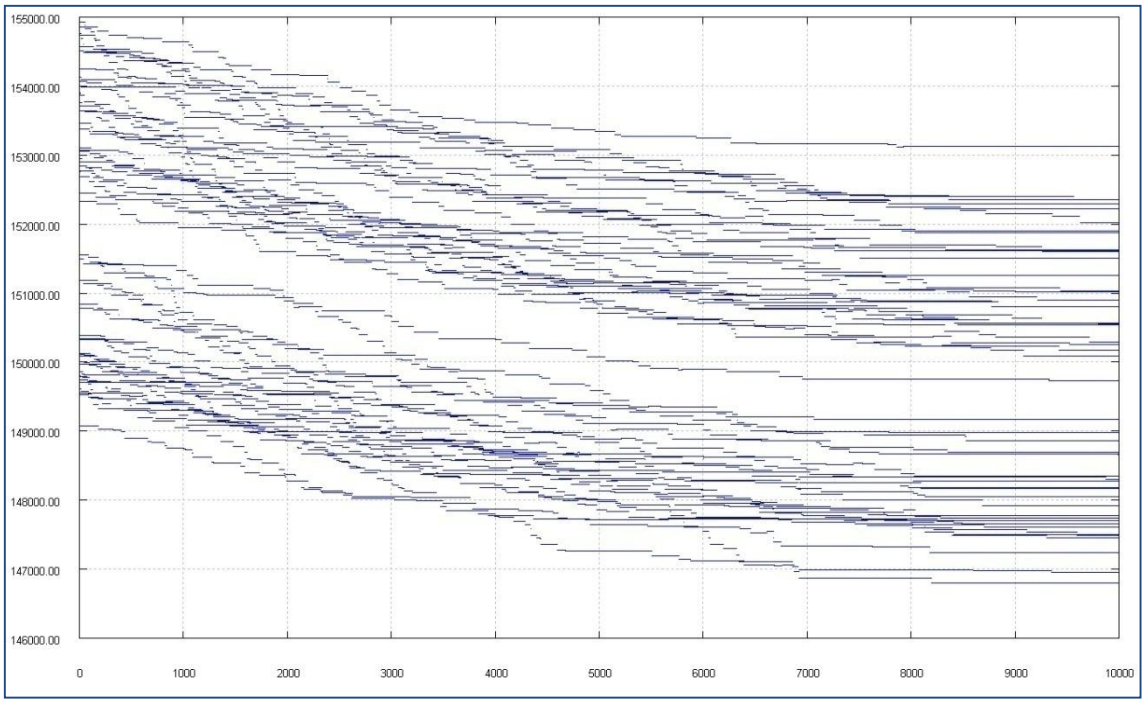

Figura 35: Nuvem de pontos para o AGPMP.

- **AGPRa:** Dentre os cinco tipos de cruzamentos testados, o AGPRa foi o método que obteve o melhor desempenho, conseguindo baixar o *fitness* a menos que 147.000 em vários cromossomos. Pode-se reparar na Figura 36 que no AGPRa existe certo grau de convergência depois de quinhentos segundos de execução, porém, este AG mantém uma boa heterogeneidade na população durante as gerações.

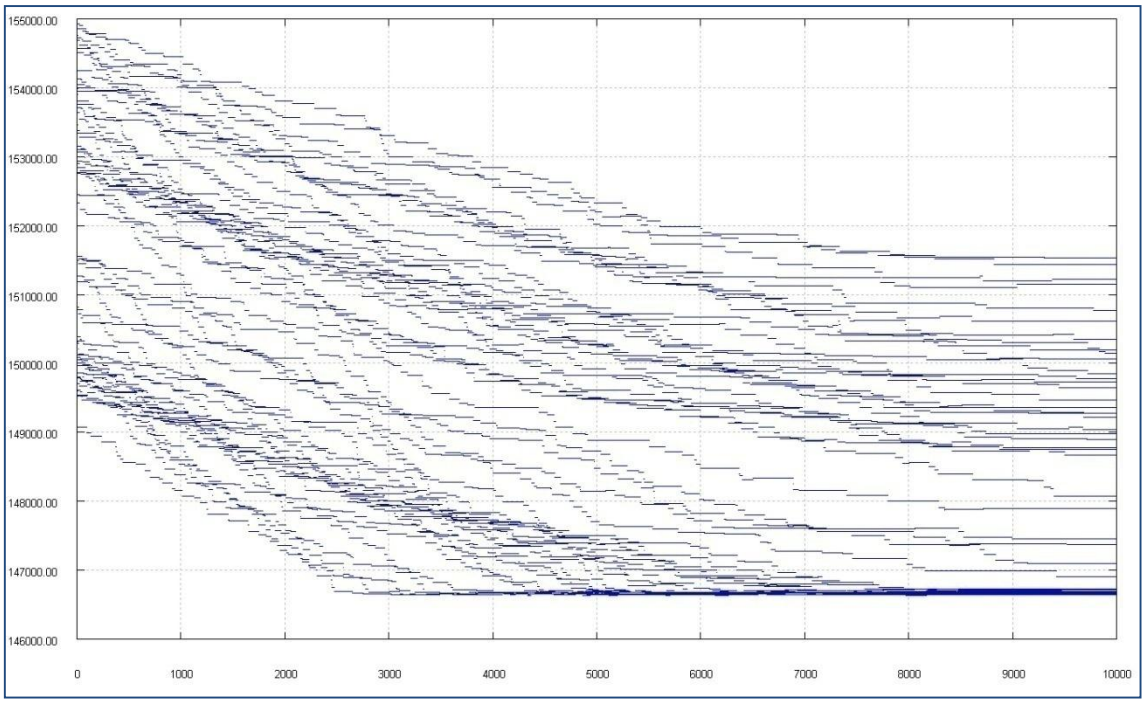

Figura 36: Nuvem de pontos para o AGPRa.

- **Nuvem de pontos do AGPRm:** o AGPRm apresenta o maior nível de complexidade dentre os 5 métodos propostos, logo é o método que consumiu a maior quantidade de tempo para este teste, como pode-se observar na ilustração da Figura 37. Assim como o AGPC, o AGPRm não consegue baixar o *fitness* a menos que 147.000, porem deixa a população muito heterogênea, apresentando um desempenho levemente superior que o AGPC. Entretanto, dependendo da aplicação, esta pequena diferença de desempenho, pode ser pouco significativa devido ao tempo de processamento consumido por este método.

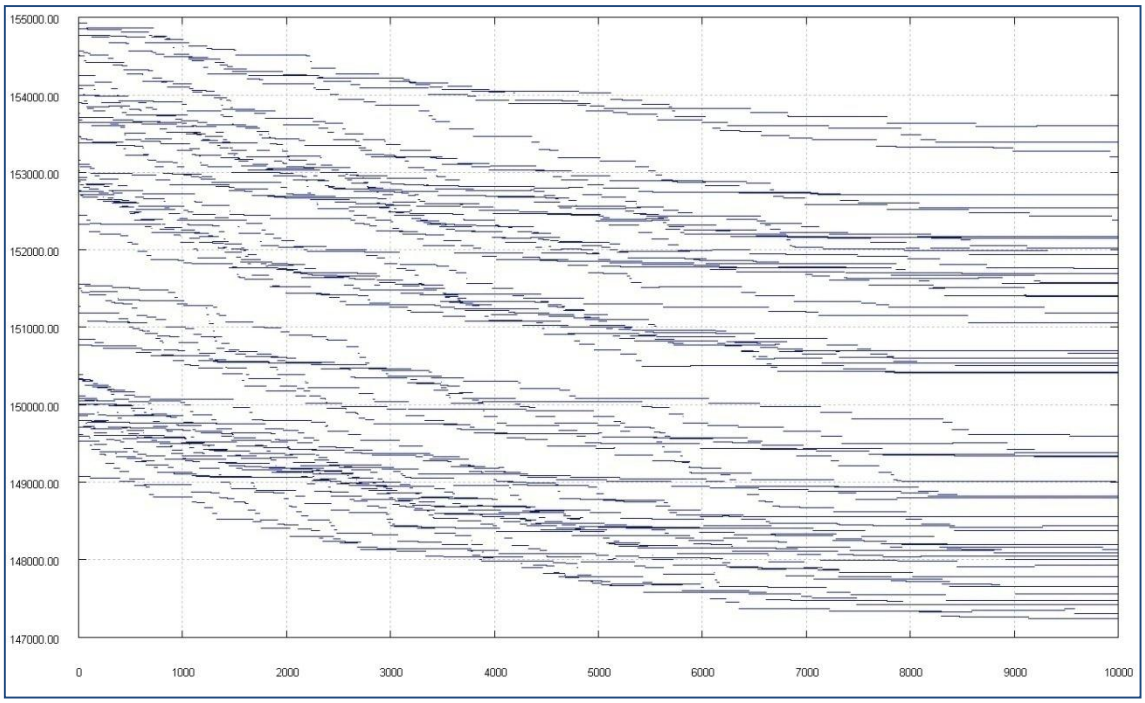

Figura 37: Nuvem de pontos para o AGPRm.

- **Nuvem de pontos do AGR:** este método foi desenvolvido no intuito de proporcionar maior desempenho para o algoritmo genético. Sua lógica permite que quando um dos métodos anteriormente propostos parasse de evoluir a população, ele trocasse os métodos de cruzamento oferecendo assim uma nova lógica para a troca de genes entre os cromossomos, o que teoricamente proporcionaria a continuação da evolução da população. Porém na prática, este método se mostrou pouco eficaz, onde manteve uma boa heterogeneidade da população durante as gerações e um baixo desempenho não conseguindo melhorar o *fitness* a menos que 147.000 conforme ilustra a Figura 38, assim como os métodos AGPC e AGPRm.

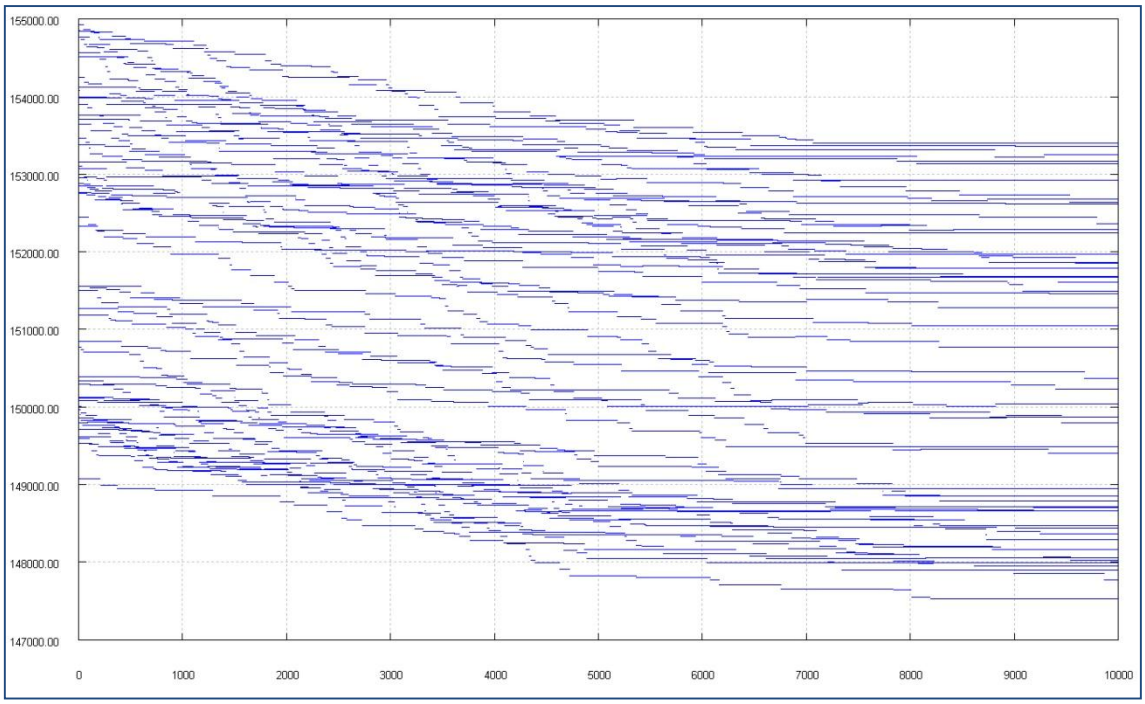

Figura 38: Nuvem de pontos para o AGR.

5.4 APTIDÃO ALVO

Neste teste pode-se observar o desempenho de cada método de cruzamento proposto para este trabalho em decorrência do tempo de execução dos mesmos. Assim como no teste anterior, foi definida a instância com 818 vértices e 50 mediadas, por ser uma instância com alto grau de dificuldade, porém com um tempo computacional viável. Foram realizadas 100 independentes execuções de cada AG para esta instância.

Cada execução terminava quando uma solução de valor menor ou igual a um certo valor alvo era encontrado. Este valor "*fitness*" foi escolhido de tal forma que o AG pudesse terminar depois de um tempo computacional considerável.

São definidos pelo usuário 3 parâmetros de entrada: *fitness*, tempo e número de execuções.

- **Fitness:** é o valor alvo "aptidão" a ser alcançado; caso o programa alcance ou ultrapasse o *fitness* alvo, o programa é finalizado, registrando em um arquivo o tempo gasto para alcançar o alvo. **Para este teste foi definido o fitness alvo de 147.500.**

- **Tempo:** é o tempo limite que o programa ficara em execução para que atinja o *fitness* alvo. Caso o tempo limite se esgote, o programa é finalizado e nada é registrado já que não foi alcançado o *fitness* alvo. **Para este teste foi definido o tempo de 1000 segundos**.

- **Número de execuções:** é a quantidade de vezes que o programa será executado para a amostragem deste teste. **Para este teste foi definido que o programa terá 100 amostragens da sua execução.**

O gráfico na Figura 39 mostra o desempenho da cada um dos métodos de cruzamentos proposto neste trabalho em para este teste. O eixo X representa o tempo de execução em segundos para alcançar o alvo, o eixo Y mostra a probabilidade do *fitness* alvo ser alcançado em decorrência ao tempo.

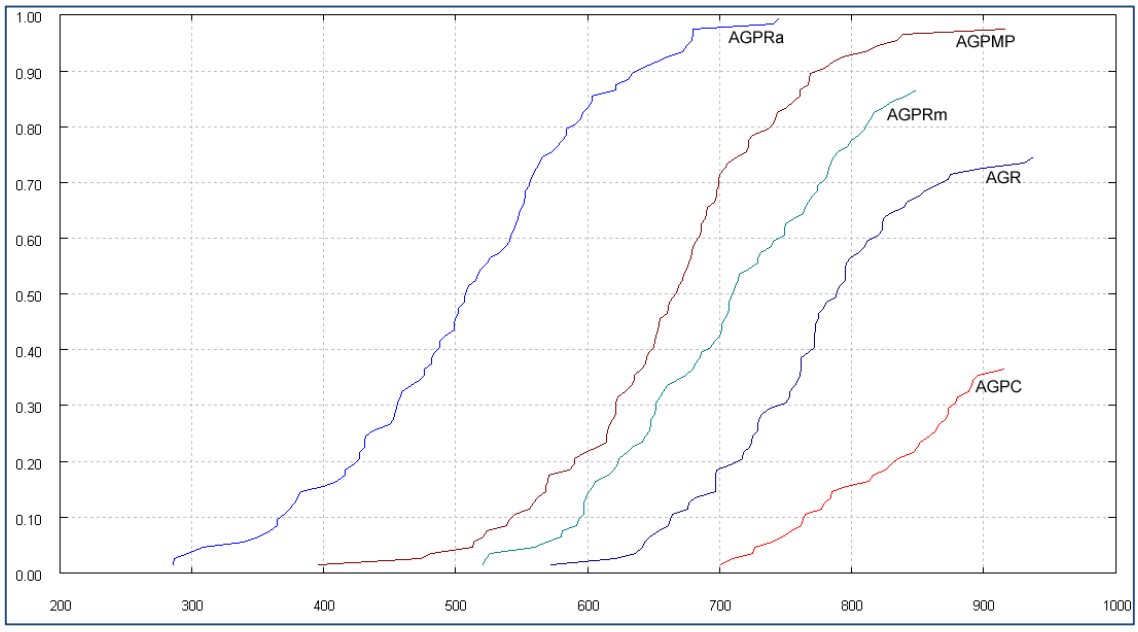

Figura 39: Aptidão alvo.

A seguir é apresentada uma análise do gráfico do teste aptidão alvo realizado, comparando os cinco AGs propostos neste trabalho.

- **AGPC:** consegue alcançar o alvo apenas em menos de 40% das execuções realizadas, observem que apenas depois de 700 segundos em execução o método consegue alcançar o *fitness* alvo.

- **AGPMP:** método de cruzamento que apresenta o segundo melhor desempenho. Consegue atingir o alvo em quase 100% das execuções.

- **AGPRa:** é o método que mostra o melhor desempenho dentre os cinco métodos propostos, conseguindo atingir o alvo em 100% das execuções. Observa-se que em de 50% das execuções o AGPRa consegue atingir o alvo em pouco mais de 500 segundos, antes mesmo que a maioria dos outros métodos propostos consiga atingir o alvo pela primeira vez.

- **AGPRm**: conforme discutido na Subseção 3.4.5.4, este método teoricamente seria uma evolução do AGPRa. Porém, na prática, seu desempenho se mostrou inferior ao AGPRa, conseguindo atingir o alvo em menos de 90% das execuções, além de consumir um tempo consideravelmente maior para alcançar o *fitness* alvo.

- **AGR**: assim como discutido no teste de nuvem de pontos na Subseção 4.3.5, no teste de *fitness* alvo este método também se mostrou pouco eficaz em comparação com os outros métodos propostos nesse trabalho, conseguindo atingir o alvo em menos de 80% das execuções e com alto consumo de tempo para alcançar o alvo.

5.5 APLICATIVO *VISUAL MEDIAN*

Por fim, foi desenvolvido o aplicativo "*Visual Median*" para representar graficamente os vértices, medianas e suas respectivas ligações conforme ilustra a Figura 40. Os algoritmos genéticos desenvolvidos para solução do problema de p-medianas descrito neste trabalho foi integrado ao *Visual Median*. Tais integrações são descritas a seguir.

O *Visual Median* foi desenvolvido em Flash usando a linguagem Action Script que utiliza como entrada de dados de dois arquivos:

- **mediana.txt:** Gerado como saída no aplicativo do AG implementado em linguagem C como descrito na Seção 4; contém apenas a melhor solução "Cromossomo" encontrada pelo AG durante todas as gerações.

- **dados.txt**: Arquivo que contém as coordenadas (x,y) das instâncias citadas na Seção 4. A Fugira 40 Ilustra o *layout* do aplicativo desenvolvido.

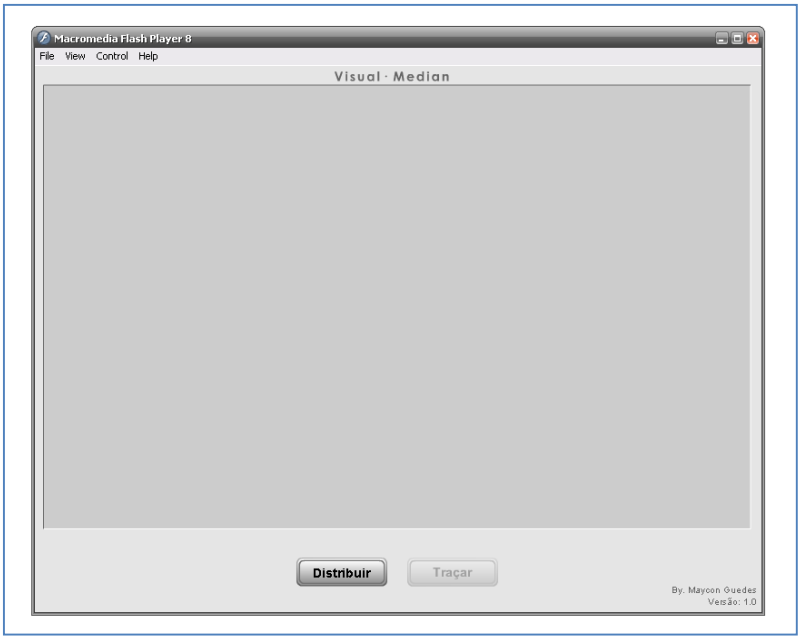

Figura 40: *Visual Median*.

A partir desses dados, o usuário pode reproduzir graficamente todos os vértices no aplicativo conforme mostra a Figura 41.

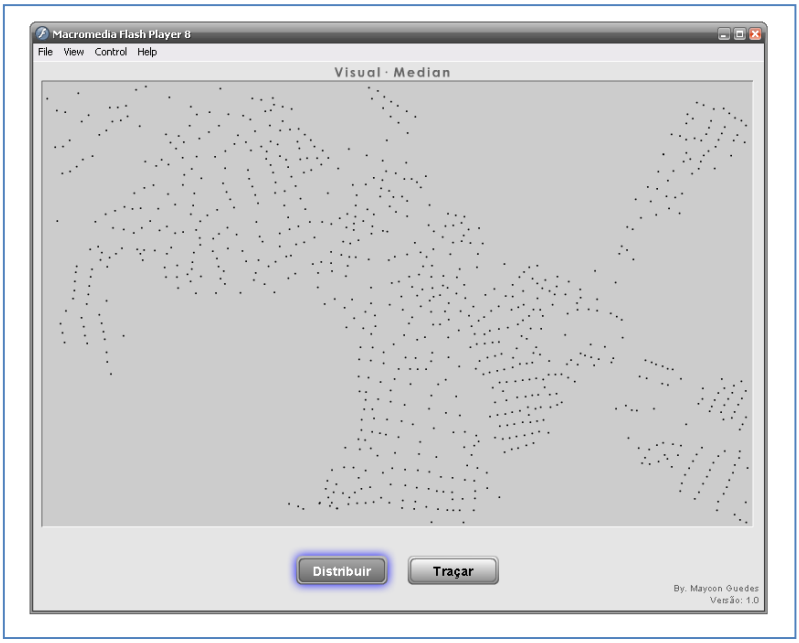

Figura 41: *Visual Median* - Desenhando Vértices.

A instância representada graficamente na Figura 42 é a de 818 vértices descrita na Seção 4. Em seguida, o usuário pode traçar graficamente as ligações com as medianas dadas como solução do algoritmo genético.

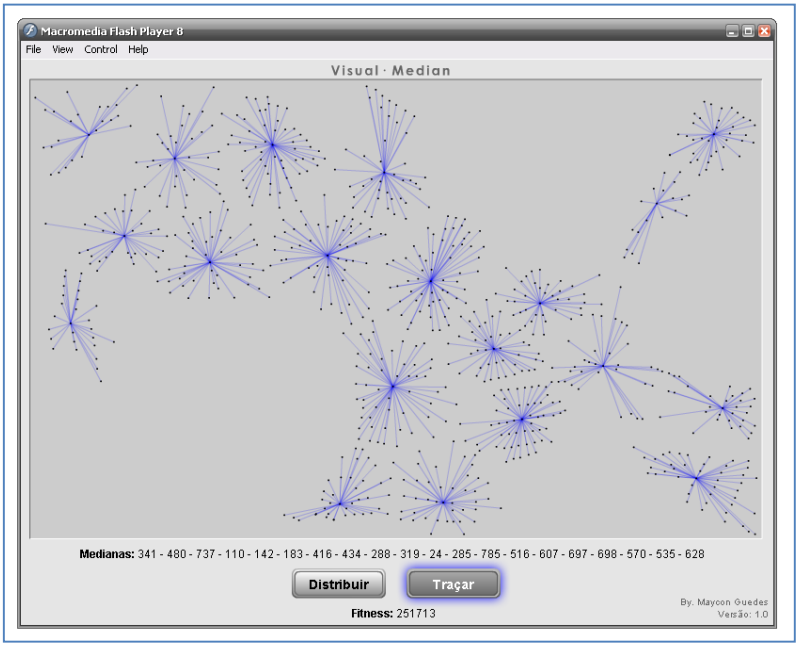

Figura 42: *Visual Median* - Desenhando ligações com as medianas.

As medianas representadas no gráfico da Figura 42 foram tratadas pelo algoritmo do AGPRa onde foi definido pelo usuário a busca de 20 medianas dentre 818 vértices.

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÕES

No problema das p-medianas pretende-se localizar facilidades (medianas) para melhor servir a clientes de forma a otimizar um certo critério (DREZNER, 1995).

Neste trabalho foram propostos cinco algoritmos genéticos para o problema das p-medianas,(AGPC, AGPMP, AGPRa, AGPRm, AGR), os quais se diferem pelo operador genético de cruzamento utilizado. Dois desses operadores, denominados "*path relinking* aleatório" e "*path relinking* melhor", foram aplicados pela primeira vez ao problema das p-medianas neste trabalho, onde se obteve bons resultados, utilizando a técnica de reconexão por caminhos, originalmente proposta como uma estratégia de intensificação para explorar trajetórias conectando soluções elites obtidas por heurísticas busca tabu e busca dispersa (*scatter search*) (GLOVER, 1996, 2000; GLOVER & LAGUNA, 1997; GLOVER *et al*., 2000). Outro operador de destaque é o operador denominado neste trabalho como "ponto mais próximo" desenvolvido pela primeira vez neste trabalho que se baseia em uma ideia já conhecida de escolha aleatória de informações dos pais para preenchimento dos filhos, mas utiliza uma eficiente etapa de recombinação de genes durante o cruzamento.

Destaca-se também neste trabalho o desenvolvimento de um novo operador em AG, denominado "Detecção de Clone" responsável por manter a heterogeneidade na população durante as gerações.

Os experimentos realizados mostraram que o algoritmo AGPRa, que utiliza o operador de cruzamento "*path relinking* aleatório", foi o que obteve os melhores resultados quando os custos médios foram comparados e também quando os tempos médios foram comparados. O algoritmo AGPMP, que utiliza

o operador de cruzamento “ponto mais próximo”, também obteve bons resultados. Já o algoritmo AGPRm, que utiliza o operador de cruzamento “*path relinking* melhor”, e o operador AGR, que utiliza o operador de cruzamento “reativo” não obtiveram os resultados esperados.

6.2 TRABALHOS FUTUROS

Para Trabalhos futuros sugerem-se as seguintes opções:

- Aprimoramento de algoritmo genético para o problema de p-medianas capacitada;
- Inclusão de novos métodos heurísticos para criar a população inicial do AG e substituição de clones; e
- Desenvolver um algoritmo genético reativo para outros operadores de um AG, tais como seleção, mutação e sobrevivência.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ALP, O.; ERKUT, E.; DREZNER, D. An efficient genetic algorithm for the *p*-median problem. *Annals of Operations Research,* v.122, p.21-42, 2003.

CHAUDHRY, S. S.; HE, S.; CHAUDHRY, P.E. Solving a class of facility location problems using genetic algorithm. *Expert Systems,* v.20, p.86-91, 2003.

CHOU, H.; PREMKUMAR, G.; CHU, C. H. Genetic algorithms and network design: Ananalysis of factors influencing ga's performance. *From the e-Business Research Center Working Paper*, p. 1–38, 1999.

COTTA, C. Scatter search with *path relinking* for phylogenetic inference*. European Journal of Operational Research*, v.169, p.520-532, 2006.

CRAINIC, T.; GENDREAU, M.; HANSEN, P.; MLADENOVIĆ, N. Cooperative parallel variable neighborhood search for the *p*-median. *Journal of Heuristics,* v.10, p.293-314, 2004.

DOMINGUEZ, M. E.; MUNOZ, P. J. A neural model for the p-median problem. *Computers & Operations Research,* v.35, p.404-416, 2008.

DOMINGUEZ M. E.; MUNOZ P. J.; JEREZ A. J. Neural Network Algorithms for the *p*-median problem. Em: *ESANN'2003 Proceedings – European Symposium on Artificial Neural Networks,*Bruges, Belgium, p.385-391, 2003.

DREZNER, Z. *A Survey of Applications and Methods*, NY, USA: Springer-Verlag, 1995.

FESTA, P.; RESENDE, M. G. C. GRASP An annotated bibliography. Em: Ribeiro, C. C., Hansen, P. (editores). *Essays and Surveys in Metaheuristics*. Kluwer, Dordrecht, p.325-367, 2002.

FLESZAR, K.; HINDI, K. S. An effective VNS for the capacitated p-median problem. *European Journal of Operational Research*, v.191(3), p.612-622, 2008.

GARCÍA-LÓPEZ, F.; MELIÁN BATISTA, B.; MORENO-PÉREZ, MORENO-VEJA, J. A. The parallel variable neighborhood search for the *p*-median problem. *Journal of Heuristics,* v.8, p.375-388, 2002.

GAREY, M. R.; JOHNSON, D. S. *Computers and intractability: a guide to the theory of NPcompleteness*, San Francisco, USA: W. H. Freeman and Co, 1979.

GLOVER, F. Multi-start and strategic oscillation methods – principles to exploit adaptive memory. Em: Laguna, M., Gonzáles-Velarde, J.L. (editores). *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research,* Kluwer, Dordrecht, p.1-24, 2000.

GLOVER, F. Tabu search and adaptive memory programing – advances, applications and challenges. Em: Barr, R.S., Helgason, R.V., Kennington, J.L. *Interfaces in Computer Science and Operations Research*, Kluwer, Dordrecht, p.1-75, 1996.

GLOVER, F.; KOCHENBERGER, G. *Handbook of Metaheuristics*. Dordrecht: Kluwer, 2003.

GLOVER, F.; LAGUNA, M. *Tabu Search*. Dordrecht: Kluwer, 1997.

GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning.* Massachusetts, USA: Addison Wesley. 1989.

LEVANOVA, T.; LORESH, M. A., Algorithms of ant system and simulated annealing for the *p*-median problem. *Automation and Remote Control,* v.65, p.431-438, 2004.

LORENA, L. A. N.; SENNE, E. L. F.; PAIVA, J. A. M.; MARCONDES, S. S. P. B. Integração de um modelo de p-medianas a sistemas de informações geográficas. Em: *Anais do XXXI Simpósio Brasileiro de Pesquisa Operacional*, Juiz de Fora, MG, p.635-647, 1999.

M. PRAIS e C.C. RIBEIRO, “Variação de parâmetros em procedimentos GRASP”, Investigación Operativa 9 2000,p. 1-20.

MLADENOVIĆ, N.; MORENO-PÉREZ, J. A.; MORENO-VEJA, J. M. A chain-interchange heuristic method. *Yugoslav Journal of Operations Research,* v.6, p.41-54, 1996.

PALAZZO, L. A. M. *Algoritmos para Computação Evolutiva*. Relatório Técnico, Grupo de Pesquisa em Inteligência Artificial - Universidade Católica de Pelotas, Pelotas, RS, 1997.

RESENDE, M. G. C.; RIBEIRO, C. C. GRASP with path-relinking: Recent advances and applications. Em: Ibaraki, T., Nonobe, K., Yagiura, M. (editores). *Metaheuristics: Progress as Real Problem Solvers.* Springer, Berlin, p.29-63, 2005.

RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures. Em: Glover, F., Kochenberger, G. (editores). *Handbook of Metaheuristics*. Kluwer, Dordrecht, p.219-249. 2003.

RESENDE, M.; WERNECK, R. F. On the implementation of a swap-based local search procedure for the *p*-median problem. Em: Ladner, R. E. (editor), *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*, SIAM, Philadelphia, p.119-127, 2003.

RIBEIRO, C. C.; VIANNA, D. S. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. Em: *Anais do II Workshop Brasileiro de Bioinformática. Universo*, Macaé, p.97-102, 2003.

RIBEIRO, C. C.; VIANNA, D. S. A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. *International Transactions in Operational Research*, v.16, p.641-657, 2009.

SOBRINHO, A. C. *Uma análise dos algoritmos genéticos e suas aplicações em sistemas de acesso à informação*. Monografia de conclusão de curso de graduação, CGCC, Universidade Federal do Maranhão, Maranhão, 2003.

SOBRINHO; A. C., GIRARDI, R. Uma Análise das Aplicações dos Algoritmos Genéticos em Sistemas de Acesso à Informação Personalizada. *REIC. Revista Eletrônica de Iniciação Científica*, v.3(4), p.1, 2003. Disponível em:<http//www.sbc.org.br>. acessado em:

SOUZA, F. Togai – Uma ferramenta para implementação de Algoritmos Genéticos. 2008. 118 f. Dissertação (Mestrado) – Programa de pós-graduação em pesquisa operacional e inteligência artificial, Universidade Candido Mendes – UCAM, Campos dos Goytacazes, RJ. 2008.

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

VIANNA, D. S.; RIBEIRO, C. C. *Heurísticas híbridas para o problema da filogenia*. 101p. Tese (Doutorado) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, RJ, 2004.