Monografia de Graduação

Algoritmos Genéticos: Uma Aplicação à Robótica Submarina

Caio Júlio César do Vale Fernandes da Silva

Natal, fevereiro de 2012

UFRN - CT - NUPEG - Campus Universitário - CEP: 59070-970 - Natal-RN - Brasil
Fone-Fax: (84)3215 3773 - www.nupeg.ufrn.br - prhanp14@nupeg.ufrn.br

Caio Júlio César do Vale Fernandes da Silva

Algoritmos Genéticos: Uma Aplicação à Robótica Submarina

Monografia apresentada à Coordenação do Curso de Graduação em Engenharia Mecânica da Universidade Federal do Rio Grande do Norte para a obtenção da nota da Atividade DEM0551 – Trabalho de Conclusão de Curso II.

Orientador:

Prof. Dr. Wallace Moreira Bessa

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE – UFRN
PRÓ-REITORIA DE GRADUACÃO – PROGRAD
CENTRO DE TECNOLOGIA – CT
DEPARTAMENTO DE ENGENHARIA MECÂNICA – DEM

Natal – RN
Fevereiro 2012

RESUMO

Existem muitos problemas em que a descrição matemática é impossível ou simplesmente se constitui de características difíceis e custosas (não impossíveis) demais para sua implementação. Matemáticos e biólogos, inspirados no processo de seleção natural como um mecanismo de evolução proposto por Charles Darwin para explicar a adaptação e especialização dos seres vivos, desenvolveram técnicas biologicamente inspiradas por volta da segunda metade do século XX, para validar modelos biológicos e resolver problemas de busca e otimização.

Os algoritmos genéticos possuem capacidade de lidar com problemas não lineares e ambientes não estacionários, adaptabilidade, gerar boas soluções em tempo suficientemente rápido para problemas de elevada complexidade com custo computacional aceitável, enquanto técnicas convencionais de obtenção de soluções ótimas requerem um custo computacional impossível (para os dias de hoje). Essa última característica é uma das mais relevantes para o uso dos algoritmos genéticos para a resolução do problema do caixeiro viajante, pois o esforço computacional necessário para a sua resolução cresce exponencialmente com o número de cidades.

Neste trabalho, os algoritmos genéticos são aplicados a um problema frequente na área de robótica submarina. Veículos robóticos submarinos são comumente utilizados para operações de instalação e manutenção de dutos na indústria petrolífera, percorrendo distâncias entre os pontos de operação. Reduzir a distância percorrida pelo robô se traduz como diminuição de horas de trabalho, redução de consumo de combustível, portanto reduzindo os custos operacionais. Devido a semelhança com o problema do caixeiro viajante, podemos tratar o problema acima citado como tal, utilizando uma outra conotação.

SUMÁRIO

1 INTRODUÇÃO

O homem, na busca pelo progresso e melhoria das condições de vida, tem utilizado os recursos existentes no ambiente como uma forma de otimizar sua vida. Para isso, usualmente como já foi visto na história da humanidade, a inspiração para a resolução de problemas que antes pareciam impossíveis de serem resolvidos vieram da natureza, alguns exemplos típicos disso são o avião e o submarino, invenções humanas, ambas criadas aproveitando características das aves e dos peixes, respectivamente.

Com o avanço da matemática, muitos problemas mundanos puderam ser descritos, mas isso não significou que as respostas foram encontradas. Com o advento dos computadores, muitos problemas tiveram suas respostas encontradas, e a área da otimização cresceu consideravelmente. Nesse contexto, entende-se como problema de otimização aquele em que se procura maximizar ou minimizar uma função numérica com certo número de variáveis, sujeitas a certo conjunto de condições que restringem o espaço das soluções do problema.

Mesmo com o grande avanço na área, existem muitos problemas onde a descrição matemática é impossível ou simplesmente se constitui de características difíceis e custosas (não impossíveis) demais para sua implementação. Desta forma, técnicas inspiradas na teoria da evolução das espécies, de Charles Darwin, deram origem a computação evolutiva. Segundo von Zuben (2007) uma grande vantagem é resolver os problemas com uma descrição matemática simples do objetivo da resposta, não necessitando explicitar cada passo até o resultado.

Em termos históricos, três algoritmos para computação evolutiva foram desenvolvidos independentemente, citados por von Zuben (2007):

Algoritmos genéticos: HOLLAND (1962), BREMERMANN (1962) e FRASER (1957);

Programação evolutiva: FOGEL (1962);

Estratégias evolutivas: RECHENBERG (1965) e SCHWEFEL (1965).

Os algoritmos genéticos se assemelham muito aos processos evolutivos naturais, pois são estruturados de forma que as informações de determinado sistema possam ser codificadas de forma análoga aos sistemas naturais. O algoritmo genético básico envolve passos como

codificação das variáveis, criação da população inicial, avaliação da resposta, cruzamento (crossover), mutação e seleção dos mais aptos.

Algumas aplicações dos algoritmos genéticos citadas por Sivanandam e Deepa (2008) são para o uso em problemas de otimização, aprendizagem de máquinas, sistemas dinâmicos não lineares, análise de arquivos, caixeiro viajante (TSP – Traveling Salesman Problem), processamento de sinais, controle de vazão de gás em gasodutos, distribuição de água, problemas de logística de transporte (Feitosa e Sena, 2009), esquematização de horários (Matos, 2007) dentre outros.

Neste trabalho, os algoritmos genéticos são aplicados a um problema frequente na área de robótica submarina. Veículos robóticos submarinos são comumente utilizados para operações de instalação e manutenção de dutos na indústria petrolífera, percorrendo distâncias entre os pontos de operação. Reduzir a distância percorrida pelo robô se traduz como diminuição de horas de trabalho, redução de consumo de combustível, portanto reduzindo os custos operacionais. Devido à semelhança com o problema do caixeiro viajante, podemos tratar o problema acima citado como tal, utilizando uma outra conotação.

2 CONCEITOS FUNDAMENTAIS

Devido a grande inter-relação existente entre os algoritmos genéticos e a biologia, faz-se necessário explicitar os conceitos da biologia que deram origem a computação evolutiva e aos algorítmos genéticos.

2.1 Hereditariedade

Hereditariedade diz respeito aos processos biológicos que garantem a relação de cada ser vivo para receber e transmitir informações genéticas através da reprodução. Grandes estudiosos do passado tentaram elucidar o mecanismo de reprodução humana e assim a transferência de material genético dos pais para os filhos. Segundo von Zuben (2007), as primeiras idéias vinculadas a hereditariedade datam de 6000 anos atrás. Várias teorias foram criadas, dentre elas, por exemplo, acreditava-se que a mulher servia apenas como incubadora para o processo, e o sexo da criança era definido pelo testículo em que o líquido seminal era liberado, se fosse do testículo direito a criança seria do sexo masculino, se do testículo esquerdo a criança seria do sexo feminino, sendo essa teoria criada por filósofos gregos por volta de 500 a.C. Aproximadamente dois mil anos depois, em 1672, Graaf descobriu o óvulo e a evidência do papel da mulher na reprodução. Em 1675 Von Leeuwenhoeck descobriu o espermatozoide, e assim criou-se a teoria do homúnculo, onde acreditava-se que o feto em estágio final já se encontrava dentro ou do espermatozoide ou do óvulo em pequena dimensão, e que durante a gestação ele apenas adquiria maiores proporções. Em 1775, Spallanzani provou que a reprodução só aconteceria se houvesse um espermatozoide e um óvulo, ocorrendo então a fecundação. Em 1866, Gregor Mendel, conhecido como o pai da genética, realizou estudos na área e elucidou os primeiros fatos sobre os genes, propondo que a existência de algumas características das flores se dava devido à existência de um par de unidades elementares de hereditariedade (genes).

2.2 Seleção Natural

O processo de seleção natural foi um mecanismo de evolução proposto por Charles Darwin para explicar a adaptação e especialização dos seres vivos. Este processo implica que

aqueles mais adaptados ao ambiente em que vivem, têm mais chances de sobreviver e assim gerar descendentes, desta forma as características genéticas dos mais adaptados seriam perpetuadas para a nova geração, enquanto aqueles indivíduos que possuem características desfavoráveis ao ambiente têm menos chance de passar seu material genético para a nova geração. Ainda, Segundo Chediak (2005), o processo evolutivo deve ser considerado como um mecanismo que envolve basicamente a sobrevivência diferenciada de entidades a partir da existência de variação na aptidão e da hereditariedade.

2.3 Genótipo e Fenótipo

Os termos genótipo e fenótipo foram criados pelo pesquisador dinamarquês Wilhelm L. Johannsen (1857-1912). O genótipo são as informações hereditárias de um organismo contidas em seu genoma. O fenótipo resulta da expressão dos genes do organismo, da influência de fatores ambientais e da possível interação entre os dois. Sendo assim, organismos que possuem o mesmo genótipo podem apresentar características diferentes, dependendo do ambiente em que vivem. Por exemplo, no caso de gêmeos idênticos, que morem em lugares diferentes, um perto da linha do equador e outro próximo de um dos polos, mesmo com genótipos idênticos, eles vão apresentar fenótipos diferentes, pois o modo de se sobreviver em cada um desses lugares é diferente em relação ao outro, evidenciando características necessárias a cada um desses ambientes, como por exemplo a pigmentação da pele. De qualquer forma, vale salientar que organismos que sejam praticamente idênticos podem apresentar genótipos diferentes.

2.4 Outros Conceitos Importantes

Alguns outros conceitos genéticos também são utilizados por diversos algoritmos evolutivos (alguns conceitos foram adaptados de WIKIPEDIA):

DNA: sigla que vem do inglês para o Ácido Desoxirribonucléico, a principal função do DNA é passar o material genético para os descendentes, além de conter genes fundamentais para a produção de proteínas fundamentais à vida.

Célula: unidade estrutural básica dos seres vivos, que se compõe de numerosas partes, sendo as fundamentais a parede ou membrana, o protoplasma e o núcleo. A célula é a menor unidade de matéria viva que pode existir de maneira independente, e ser capaz de se

reproduzir. Toda célula de um mesmo organismo contém o mesmo conjunto de um ou mais cromossomos.

Cromossomo: é uma estrutura formada por uma cadeia de DNA sendo a base física dos genes. Cada organismo tem um número diferente de cromossomos. O ser humano, por exemplo, tem 46 (recebemos 23 da mãe e outros 23 do pai).

Genes: cada gene representa uma ou um grupo de características relacionada ao ser vivo.

Genoma: é toda a informação hereditária de um organismo que está codificada em seu DNA (ou, em alguns vírus, no RNA).

Crossover: Operação biológica que promove a troca de material genético entre dois indivíduos.

Mutação: mutações são mudanças na sequência dos nucleotídeos do material genético de um organismo.

Locus: (do latim "lugar", no plural **loci**) é o local fixo num cromossomo onde está localizado determinado gene.

Alelo: é uma sequência de uma molécula de DNA (gene) situada no mesmo locus e que corresponde a diferentes versões do mesmo gene.

3 ALGORITMOS GENÉTICOS

Conforme Sivanandam e Deepa (2008), a computação evolutiva foi introduzida por volta de 1960 por Rechenberg no trabalho intitulado “Evolution strategies” (Estratégias de evolução). Os Algoritmos Genéticos foram inventados por John Holland e mais desenvolvidos em seu livro “Adaptation in natural and artificial systems” no ano de 1975. Neste livro ele descreve como aplicar os princípios da evolução natural para problemas de otimização e construiu o primeiro algoritmo genético, inspiração essa que veio da teoria da evolução natural na origem das espécies de Charles Darwin. Holland propôs os Algoritmos Genéticos como um método heurístico baseado na “sobrevivência do mais apto”. Assim, os algoritmos genéticos foram descobertos como uma ferramenta útil para problemas de procura e otimização.

Algoritmos genéticos são implementados como uma simulação de computador em que uma população de representações abstratas de solução é selecionada em busca de soluções melhores. A evolução geralmente se inicia a partir de um conjunto de soluções criado aleatoriamente (população inicial) e é realizada por meio de gerações. A cada geração, a adaptação de cada solução na população é avaliada, alguns indivíduos são selecionados para a próxima geração, e recombinados ou mutados para formar uma nova população que é avaliada de acordo com os critérios de parada, se a solução for aceitável, o algoritmo retorna o conjunto de respostas encontrado, se não, nova população então é utilizada como entrada para a próxima iteração do algoritmo.

Termos originados na biologia serviram de inspiração para a terminologia dos algoritmos genéticos. De acordo com von Zuben (2007), cromossomos são usualmente implementados na forma de listas de atributos ou vetores, onde cada atributo é conhecido como gene. Os possíveis valores que um determinado gene pode assumir são denominados alelos. Na figura abaixo, retirada de Oliveira (2007), podemos ver mais alguns exemplos da comparação terminológica entre as duas áreas.

Linguagem natural	GA
cromossomo	individuo, *string*, cromossomo, árvore
gen	características
alelo	valor
locus	posição
genótipo	estrutura
fenótipo	conjunto de parâmetros
população	conjunto de pontos (indivíduos) no Espaço de Busca
geração	iteração completa do GA que gera uma nova população
aptidão bruta	saída gerada pela função objetivo para um individuo da população
aptidão normalizada	Aptidão bruta normalizada, entrada para o algoritmo de seleção
aptidão máxima	melhor individuo da população corrente
aptidão media	aptidão media da população corrente

Figura 1 – Terminologia dos algoritmos genéticos (Oliveira, 2007 p. 30)

Algoritmos genéticos diferem dos algoritmos tradicionais de otimização em basicamente quatro aspectos, baseiam-se em uma codificação do conjunto das soluções possíveis, e não nos parâmetros da otimização em si; os resultados são apresentados como uma população de soluções e não como uma solução única; não necessitam de nenhum conhecimento derivado do problema, apenas de uma forma de avaliação do resultado; usam transições probabilísticas e não regras determinísticas.

3.1 Indivíduos

Um indivíduo é uma solução simples. Nos algoritmos genéticos, indivíduos podem receber codificação binária, inteira, de ponto flutuante ou qualquer outro tipo que possa melhor representar o problema. A escolha adequada da codificação pode trazer melhorias para a otimização como menor custo computacional e aumento da precisão, por exemplo.

3.2 População

Segundo Cordeiro (2008, p. 13), "[...] uma população representa o conjunto atual de indivíduos encontrados em uma determinada iteração do algoritmo". A idéia é que de acordo

com as iterações, soluções mais adequadas devam ser encontradas. Dois importantes aspectos que devem ser levados em consideração: a geração da população inicial e o tamanho da população. De acordo com Sivanandam e Deepa (2008) idealmente a população inicial deve conter uma grande variedade de material genético de forma que todo o espaço de soluções possíveis seja explorado. Para obter isso, geralmente inicia-se a população de forma aleatória. Porém pode-se usar algum tipo de heurística para iniciar a população já com uma alta média dos valores da função de aptidão, mas vale salientar que tal atitude pode restringir o espaço de busca, diminuindo a variedade do material genético presente na população e assim fazendo com que a busca por uma solução ótima global não seja possível. Em relação ao tamanho da população, Goldberg mostrou que a eficiência de um algoritmo genético atingir o ótimo global em vez de ótimos locais é altamente determinado pelo tamanho da população, pois com grandes população torna-se mais fácil explorar o espaço de busca, entretanto isso requer muito mais custo computacional, memória e tempo.

3.3 Aptidão (Fitness)

A aptidão corresponde a nota associada a um indivíduo ou cromossomo que permite a avaliação de quão boa é a solução por ele representada. A função aptidão deve ser planejada para cada problema a ser resolvido. Dado um cromossomo em particular, a função de aptidão irá retornar um simples dado numérico, que indica a utilidade ou habilidade do indivíduo representado pelo cromossomo. A aptidão é utilizada como parâmetro para operações como a de seleção; utilizando o valor de aptidão também é possível indicar que indivíduos devem ser substituídos quando se usa o conceito de elitismo.

3.4 Seleção

A seleção é um dos mais importantes elementos de todos os algoritmos genéticos. A seleção determina quais indivíduos da população irão ter todo ou parte de seu material genético transferido para a próxima geração de indivíduos. O objetivo do método de seleção aplicado ao algoritmo é fazer com que o material genético de boa qualidade aumente de geração a geração, enquanto que o material genético ruim venha a desaparecer ou ficar em número reduzido. Inspirado no processo de seleção natural dos seres vivos, o algoritmo seleciona os melhores indivíduos (maior aptidão) para gerar cromossomos filhos por meio de

crossover e mutação, sempre com o objetivo de levar o algoritmo para as melhores regiões do espaço de busca. Os alguns mecanismos utilizados são: seleção roda de roleta (Roulette Wheel Selection), seleção por ranqueamento (Rank Selection) e a seleção por estado estacionário (Steady State Selection).

3.4.1 Seleção Roda de Roleta (Roulette Wheel Selection)

É um dos métodos de seleção mais tradicionais nos Algoritmos Genéticos, sendo o principio de funcionamento simples de ser explicado e implementado. Para cada indivíduo é atribuído um espaço da roleta sendo o tamanho proporcional ao valor da aptidão do indivíduo. A roda girará N vezes, onde N é o número de indivíduos da população, assim, em cada volta, um indivíduo é selecionado para fazer parte dos pais para a próxima geração.

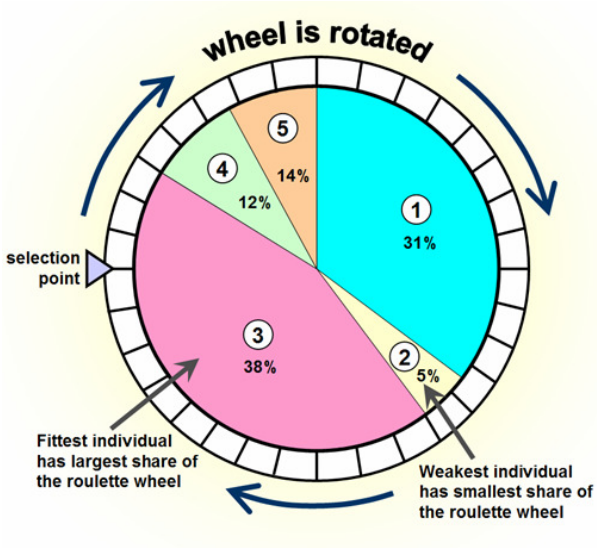

Figura 2 – Roda de Roleta (ENGINEERING Design Centre, New Castle University, Roulette Wheel Selection)

Podemos ver na figura 2 (acima) que o indivíduo de número 3 foi selecionado, pois, claramente, quanto maior a aptidão do indivíduo maior a chance dele ser selecionado.

3.4.2 Seleção por Ranqueamento (Rank Selection)

O método de seleção roda de roleta terá problemas quando os valores das aptidões variarem muito. A seleção por ranqueamento primeiro ranqueia a população e então cada cromossomo recebe uma nova aptidão de acordo com esse ranqueamento. O pior terá aptidão 1, por exemplo, o segundo pior terá aptidão 2 e assim por diante. O melhor indivíduo terá aptidão N, onde N é o número de cromossomos da população.

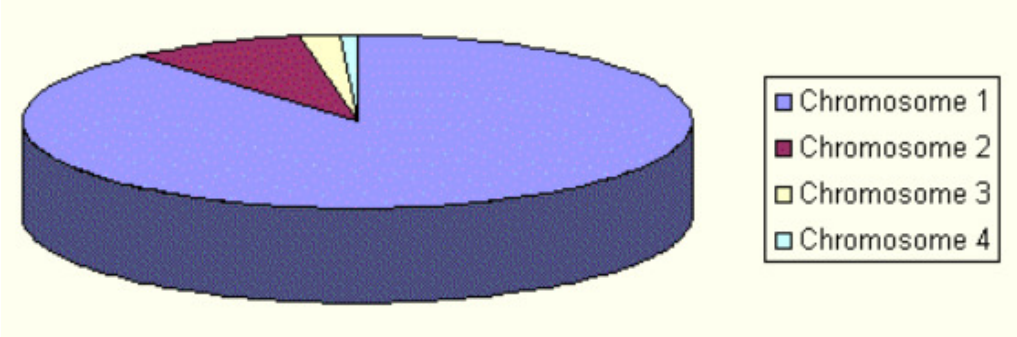

Figura 3 – Situação antes do ranqueamento. (OBTIKO)

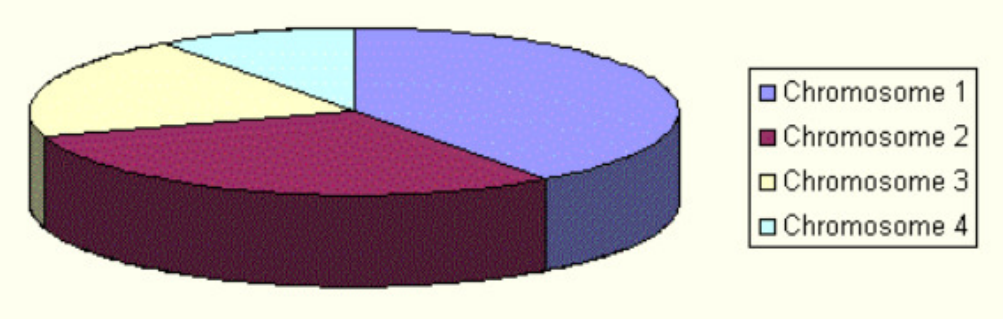

Figura 4 – Situação após o ranqueamento. (OBTIKO)

Nas figuras acima, está representada como a situação muda após trocar a aptidão por números em ordem, aumentando assim as chances de indivíduos com baixa aptidão, trazendo para a população a ser selecionada uma maior quantidade de material genético, portanto aumentando o espaço de busca das possíveis respostas e assim também diminuindo as chances de se ficar preso em ótimos locais. Porém, a convergência pode ficar mais lenta.

3.4.3 Seleção por Estado Estacionário (Steady State Selection)

Em cada nova geração um número reduzido de melhores cromossomos (com alta aptidão) são selecionados para as operações de troca de material e mudança de genético. A seguir poucos cromossomos com baixa aptidão são substituídos e os novos descendentes gerados são colocados em seus lugares (OBTIKO). Todo o resto da população que não é selecionado ou é reposto, sobrevive para a próxima geração. Portanto, este tipo de seleção visa manter um grande número de indivíduos da geração anterior em sua população.

3.5 Cruzamento (Crossover)

Segundo Von Zuben (2007, p. 13) "O operador de crossover ou recombinação cria novos indivíduos através da combinação de dois ou mais indivíduos. A idéia intuitiva por trás do operador de crossover é a troca de informação entre diferentes soluções candidatas". Para a aplicação do crossover, os pais são selecionados e, através de uma probabilidade de ocorrência o operador age formulando a troca de informações formando os novos indivíduos. Geralmente a probabilidade de ocorrência da operação é alta, acima de 50%. Quando o operador atua, cada novo indivíduo (filho) recebe características de ambos (ou todos) os pais envolvidos, quando não, os filhos são exatamente iguais aos pais. Existem vários tipos de operadores, dentre eles destacamos o crossover de um ponto, que é o mais popular. De acordo com Lima (2008, p. 13) "Trata-se de uma recombinação entre dois cromossomos pais que trocam partes de sua cadeia binária a partir de um ponto aleatório de corte".

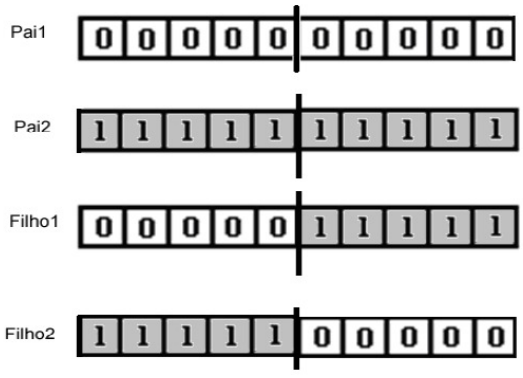

Figura 5 - Crossover de um ponto (Lima, 2008, p. 14).

3.5.1 Cruzamento PMX (Partially Matched Crossover)

De acordo com Saraiva e Oliveira (2010) o operador PMX foi proposto por Goldberg e Lingle para o Problema do Caixeiro Viajante. Dados dois cromossomos pais p1 e p2, dois pontos de corte são escolhidos aleatoriamente em ambos uniformemente. Estas subcadeias geradas serão o material genético de troca, sendo herdadas pelos filhos f1 e f2. Para evitar rotas inviáveis, um mapeamento é feito para respeitar a restrição. Geralmente a restrição imposta para o problema do caixeiro viajante é a de que não se deve visitar uma cidade mais de uma vez, ou seja, em uma rota um número não pode se repetir. Portanto, se na troca de informação uma rota inválida for criada, os números fora da subcadeia gerada pelos pontos de corte devem ser arrumados de forma a gerar uma rota válida.

$$\left.\begin{array}{l} p_1 = (1, 2, \mid 3, 4, 5, \mid 6) \\ p_2 = (6, 3, \mid 1, 4, 5, \mid 2) \end{array}\right\} \rightarrow \left\{\begin{array}{l} f_1 = (3, 2, \mid 1, 4, 5, \mid 6) \\ f_2 = (6, 1, \mid 3, 4, 5, \mid 2) \end{array}\right.$$

Figura 6 – Crossover PMX (Silva e Oliveira, 2006, p. 4)

Como pode ser visto na figura acima, as cidades 3, 4 e 5 do pai 1 e 1, 4, e 5 do pai 2, formam o material genético de troca. Porém, verifica-se que nesta troca, uma rota inválida seria gerada, pois o pai 1 já tem a cidade 1 em sua rota e o pai 2 também já tem a cidade 3. Para resolver esse problema, o mapeamento 1-3, 4-4, 5-5 é realizado, e, no pai 1, onde havia 1 fora da região de troca foi-se substituído por 3, finalmente dando origem ao filho 1, e, no pai 2, onde havia 3 fora da região de troca foi-se substituído por 1, dando origem ao filho 2.

3.6 Mutação

Segundo Cordeiro (2008, p. 14), mutação "[...] consiste em mutar aleatoriamente um indivíduo existente de forma que um novo indivíduo seja criado". O objetivo da mutação é promover a variabilidade do material genético, ou seja, é um mecanismo que também serve para impedir que o algoritmo fique preso em um máximo ou mínimo local. A operação de mutação ocorre com probabilidades bem menores do que a de crossover, ficando geralmente abaixo de 1% de chance de sua ocorrência.

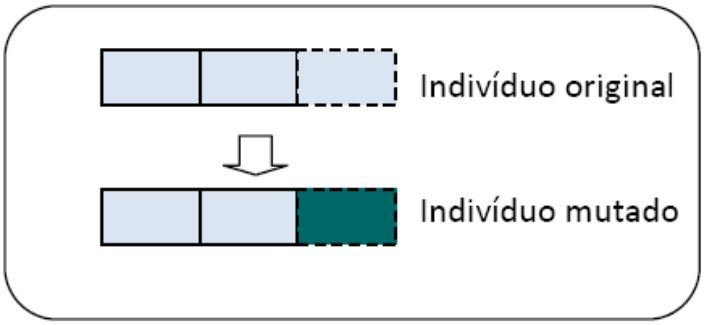

Figura 7 - Operação de mutação (Cordeiro, 2008 p. 14).

3.7 Geração

Segundo Cordeiro (2008, p. 13) "uma geração corresponde a uma iteração realizada sobre a população, de forma que realize alguma mudança sobre os indivíduos da população". Em muitos algoritmos o critério de parada pode ser definido pelo número de gerações, pois a idéia é que a cada nova geração a população evolua e possa a estabelecer respostas mais adequadas ao problema.

3.8 Elitismo

Elitismo é o nome do método que primeiro copia os melhores cromossomos da geração anterior para a geração atual. Isso garante que as melhores respostas já encontradas não sejam perdidas em operações como crossover e mutação, ou seja, prevenindo a perda da melhor solução já encontrada. De acordo com Linden (2008), elitismo é uma pequena modificação no módulo de população que quase não altera o tempo de processamento, mas que garante que o desempenho do algoritmo genético sempre cresça com o decorrer das gerações. O elitismo serve melhor para grandes populações, pois para populações pequenas tal prática pode levar a uma convergência rápida demais, aumentando as chances de se ficar preso em máximos ou mínimos locais.

3.9 Estrutura de um Algoritmo Genético

Na maioria dos casos um algoritmo genético é estruturado da seguinte forma (adaptado de Sivanandam e Deepa, 2008):

Início – Gera-se uma população aleatória de n cromossomos. Quando já se tem alguma informação sobre o problema, pode-se introduzir soluções adequadas para já iniciar a população com um valor de aptidão alto ou adequado.

Aptidão – é associado a cada cromossomo o seu valor de aptidão na população.

Seleção – os pais são selecionados de acordo com o seu valor de aptidão, quanto melhor o valor de aptidão, maior a chance desses pais serem selecionados.

Cruzamento (crossover) – de acordo com a probabilidade de ocorrer o cruzamento, as informações entre os pais são trocadas, formando-se os filhos. Se a operação de cruzamento não acontecer, os filhos serão exatamente iguais aos pais (dependendo ainda se a mutação ocorrerá ou não).

Mutação – com certa probabilidade, geralmente bem menor do que a probabilidade de ocorrer o cruzamento, a mutação pode ocorrer em cada locus dos descendentes.

Aceitação – os filhos são colocados na nova população.

Troca – substitui-se a antiga população pela nova população.

Teste – Se a condição de parada é satisfeita, o programa deve retornar a melhor solução encontrada na população atual.

Laço – Se não, volte para o passo em que a aptidão é calculada.

4 ESTUDO DE CASO: UMA APLICAÇÃO A ROBÓTICA SUBMARINA

4.1 Introdução

Dados o contexto histórico, biológico e os conceitos fundamentais sobre os algoritmos genéticos, falaremos agora sobre um problema de otimização que pode ser resolvido com a ajuda dos algoritmos genéticos.

A robótica submarina surgiu como uma ferramenta para se conseguir trabalhar em maiores profundidades com segurança e agilidade. Neste contexto destacamos os veículos robóticos não tripulados para a exploração submarina, também conhecidos como ROV (Remotely Operated underwater Vehicle). Como informação relevante para o uso destes veículos, é sabido que a maior parte do petróleo brasileiro se encontra em águas profundas e ultra profundas o que, portanto, traria risco a vida dos mergulhadores envolvidos em operações de manutenção e instalação de plataformas *offshore,* por exemplo. Segundo Bessa (Bessa et al., 2004) os ROVs "[...] têm substituído os mergulhadores na realização de tarefas que ofereçam risco à vida humana". Tais informações reforçam a importância do uso deste veículo. Entretanto, esses veículos são caros e seu custo de utilização durante a operação também, devido a alta tecnologia envolvida nos sistemas de controle, navegação e equipamentos envolvidos. Visto esses fatores, se faz necessário encontrar formas de se diminuir o tempo de operação no fundo do mar, reduzindo custos, otimizando assim as operações por eles realizadas.

Figura 8 – ROV (Google imagens).

Sabendo que o ROV deve visitar um certo número de pontos de operação, podemos encontrar a menor distância provável que este deve percorrer através de uma analogia com o problema do caixeiro viajante, onde o caixeiro deve percorrer a menor distância possível entre as cidades a serem visitadas.

Os algoritmos genéticos podem ser utilizados para solucionar o problema do caixeiro viajante aplicado à robótica submarina, onde deve-se achar a melhor rota dentre os locais de visitação, ou seja, deve-se minimizar a distância percorrida pelo ROV, com a restrição de que ele só pode passar uma vez por cada ponto de operação, com exceção do ponto de partida pois este também é o ponto de retorno e portanto é visitado duas vezes.

O problema do caixeiro viajante pertence a uma categoria de problemas (NP-completo) em que o esforço computacional necessário para a sua resolução cresce exponencialmente com o tamanho do problema. Assim, dado que é difícil, senão impossível, determinar a solução ótima, os métodos de resolução passam pelas heurísticas e afins que, do ponto de vista matemático, não asseguram a obtenção de uma solução ótima (WIKIPEDIA).

Considerando um número de 100 pontos a serem visitados pelo ROV, teríamos 99! = 9,33 x 10^{155} de rotas possíveis, tornando inviável de se analisar cada rota e assim encontrar o ótimo global (menor percurso possível). Métodos de otimização clássica aproveitam boas soluções mas ignoram o resto do espaço de busca, já métodos de busca aleatórios conseguem explorar bem o espaço de busca, mas ignoram regiões promissoras para a busca de uma boa resposta. Os algoritmos genéticos possuem um grande balanço entre aproveitar as melhores soluções e explorar o espaço de busca, tornando-se viável achar uma boa resposta para tal problema, porém, não há garantia de que o ótimo global seja encontrado. Segundo von Zuben (2007), "[...] os algoritmos genéticos não são métodos de busca puramente aleatórios, pois combinam variações aleatórias com seleção, polarizada pelos valores de adequação (fitness) atribuído a cada indivíduo". Lembrando também que tais algoritmos lidam com populações, sendo que cada indivíduo pode representar uma resposta em potencial para o problema, realizando assim um processo de busca multidirecional.

4.1 Metodologia

A linguagem de programação utilizada foi o C++ e o compilador o DevC++ versão 4.9.9.2. Primeiramente o tamanho da população e o número de pontos de visitação foi determinado. A codificação da população foi decidida como inteira, seguindo o exemplo de

autores como Sivanandam e Deepa (2008) para o problema do caixeiro viajante. Portanto, cada cromossomo será considerado como uma rota, sendo cada ponto representada por um número inteiro diferente, com a restrição de que para ser considerado como uma resposta de fato, os pontos de visitação não se repitam dentro de uma mesma rota (excetuando-se o ponto de retorno). A primeira população foi gerada de forma aleatória, sem nenhuma informação prévia das distâncias ou coordenadas envolvidas, com a restrição de que os locais de visitação dentro de uma rota não podem se repetir. Em seguida, as coordenadas em duas dimensões, consideradas como x e y, são entradas no programa e então o cálculo das distâncias entre os pontos é realizado, sendo seguido do cálculo da rota para cada cromossomo (possível resposta). Vale salientar que o cálculo da rota individual entra no cálculo da função avaliação do problema, introduzindo a condição de que quanto menor for a rota, maior a aptidão do indivíduo.

A partir disso, o método de seleção escolhido foi o de estado estacionário (steady state), conforme proposto por Sivanandam e Deepa (2008). São selecionados dois pais, considerados os melhores indivíduos da geração, ou seja com a maior aptidão atual. Depois de selecionados os pais, dá-se início a operação de crossover. Como indicado por vários autores (Sivanandam e Deepa (2008), von Zuben (2007), Linden (2008)), o operador de crossover selecionado foi o PMX (partially matched crossover), pois esse operador evita que as crianças (novas rotas) geradas possam ter problemas como pontos de operação repetidos (visitados mais de uma vez em uma mesma rota) e não visitados, o que invalidaria a rota. A probabilidade de crossover adotada foi de 100%, ou seja, em todas as iterações a operação de crossover ocorrerá com certeza. Este valor foi adotado com o propósito de aumentar a troca material genético entre os pais.

Após a operação de crossover, a mutação foi implementada com uma probabilidade de 0.1%. Este número baixo foi adotado com o propósito de não interferir artificialmente nos resultados. A mutação foi implementada de modo que aleatoriamente um ponto (alelo) trocasse de posição (locus) com outro dentro do mesmo indivíduo (cromossomo). Após a mutação, os novos indivíduos gerados foram repostos na população no lugar dos dois piores indivíduos da população inicial à iteração.

Mesmo tomando os cuidados para que os pontos não se repetissem e que todos fossem visitados, algumas rotas inválidas foram criadas no processo. Como forma de resolver tal problema, pensou-se em utilizar o conceito de elitismo, onde os melhores indivíduos de uma geração anterior são perpetuados para a próxima geração. Portanto, sempre que uma rota

inválida era criada, tal rota era substituída pelo melhor indivíduo disponível e válido da geração anterior. Entretanto, isso foi resolvido com o aprimoramento do método de seleção.

O critério de parada adotado foi o de número máximo de iterações, visto que não há informações sobre o ótimo global para se fazer comparações. Outro critério pensado foi o de diminuição da distância em relação à distância inicial, por exemplo, se o melhor indivíduo da geração atual tiver sua rota reduzida em 50% em relação a inicial então essa será a resposta. Porém, se o chute inicial for muito bom, provavelmente esse valor nunca será alcançado. Portanto, por questões de simplicidade de implementação e inviabilidade de outros critérios de parada, o número máximo de iterações foi selecionado.

4.2 Resultados obtidos

No primeiro caso consideramos uma população de tamanho igual a 20 e o número de pontos de visitação igual a 7. As coordenadas foram decididas de forma aleatória e podem facilmente serem modificadas, pois sempre que o programa é executado é pedido a entrada das coordenadas, que são decididas pelo usuário. As coordenadas dos pontos de operação são mostradas na tabela a seguir:

Tabela 1 – Coordenadas.

Cidade	Coordenadas	
Nomes	x	y
1	1	25
2	5	30
3	20	7
4	17	15
5	12	20
6	25	3
7	8	10

Em seguida é mostrado o gráfico que representa a melhor resposta encontrada pelo algoritmo. A rota final escolhida foi 1-2-5-6-3-4-7-1. O programa utilizado para gerar os gráficos das rotas foi o Rt-plot, versão 2.8 shareware.

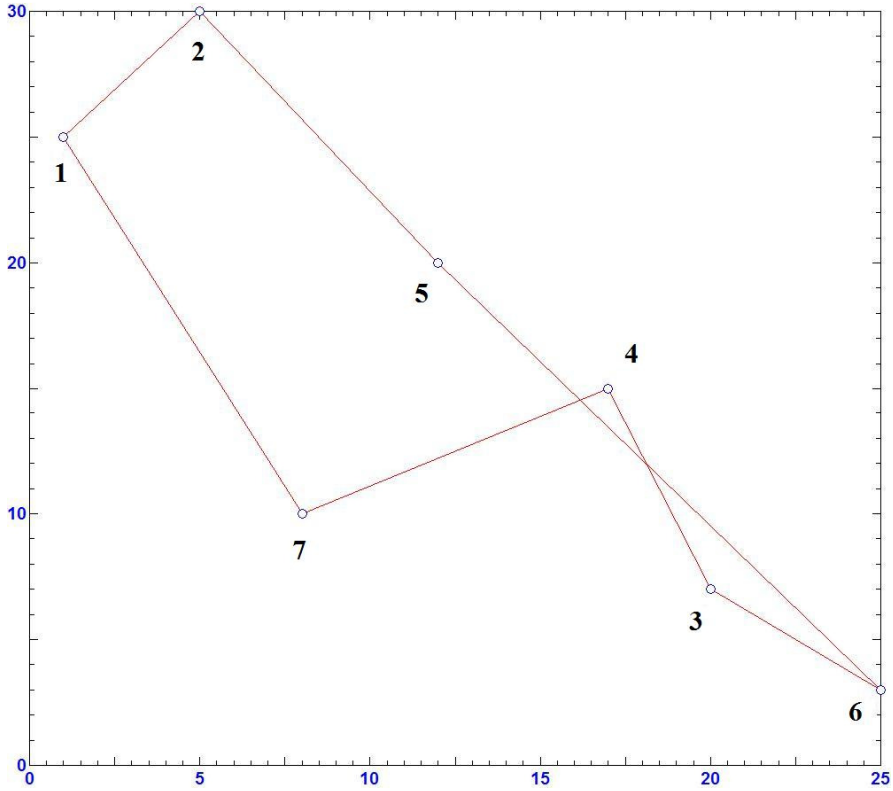

Figura 9 – Rota final do primeiro caso.

Como em Sivanandam e Deepa (2008), é apresentada a média das distâncias das rotas iniciais, em seguida a média das distâncias da última rota, e depois a comparação relativa entre as rotas iniciais e finais, e por fim o número de iterações.

Tabela 2 – Resultados.

Distancia inicial	Distancia final	Decréscimo	Iterações
118	79	33,05%	20

Pode-se observar uma grande diferença relativa entre a média das distâncias iniciais e a média das distâncias finais, evidenciando o poder do algoritmo em achar uma rota mais curta. Outro fato a ser notado é a rapidez da convergência. Em apenas poucas iterações o algoritmo foi capaz de convergir, achando uma boa resposta (não necessariamente a ideal, pois não há garantias sobre isso, como já foi explicado anteriormente). Uma desvantagem encontrada nesse algoritmo foi que ele é altamente dependente da população inicial, pois como foi constatado durante vários testes, a resposta final, na maioria das vezes, já se encontrava na população inicial.

Comentado por Sivanandam e Deepa (2008), Goldberg mostrou que a eficiência de um algoritmo genético atingir o ótimo global em vez de ótimos locais é altamente determinado pelo tamanho da população, pois com grandes população torna-se mais fácil explorar o espaço de busca, entretanto isso requer muito mais custo computacional, memória e tempo. Portanto, uma saída para melhorar a resposta pode ser o aumento da população, criando novas rotas inicialmente, o que aumenta o custo computacional, pois também é necessário um maior número de iterações para a convergência, além de um maior número de cálculos realizados.

Iremos aumentar a população de 20 para 100, e as iterações de 20 para 70, conservando as mesmas coordenadas. O aumento das iterações é necessário, visto que para um maior número de locais de visitação é necessário um maior número de iterações para analisar as possíveis respostas disponíveis. A rota final é mostrada na figura a seguir, apresentando a rota 1-7-6-3-4-5-2-1.

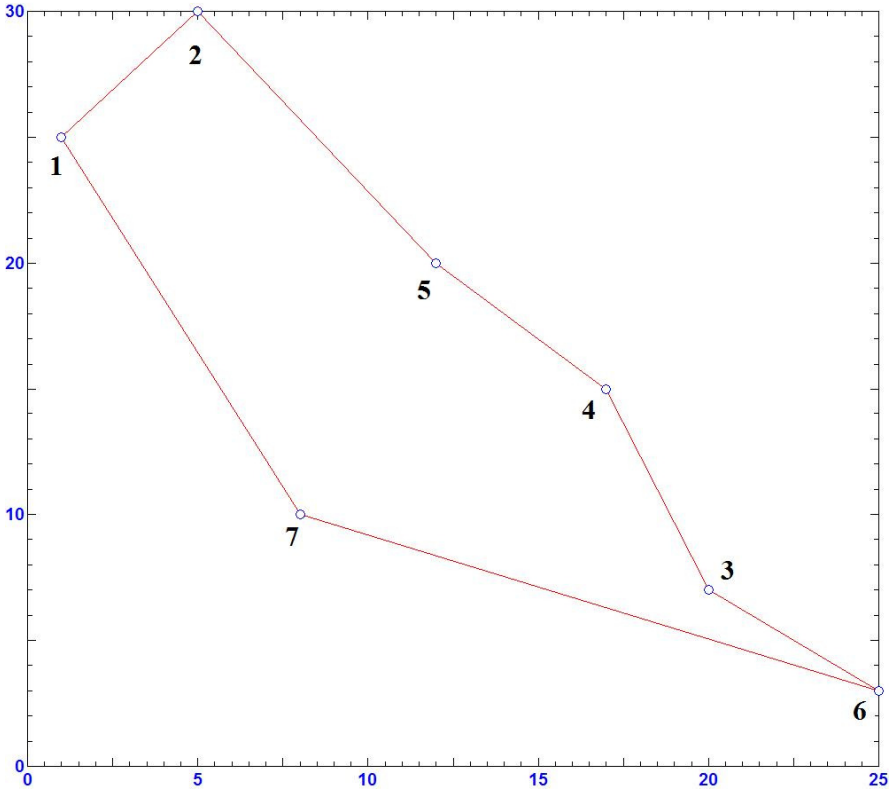

Figura 10 – Rota final do segundo caso.

Tabela 3 – Resultados.

Distancia inicial	Distancia final	Decréscimo	Iterações
113	73	35,39%	70

Comparando as médias das distancias finais (79 no primeiro caso e 73 no segundo), podemos ver uma melhora 7,59%. Vale salientar que o aumento da população aumenta a eficiência em se encontrar o ótimo global em vez de o local, não se garante que aumentando a população o ótimo global será encontrado.

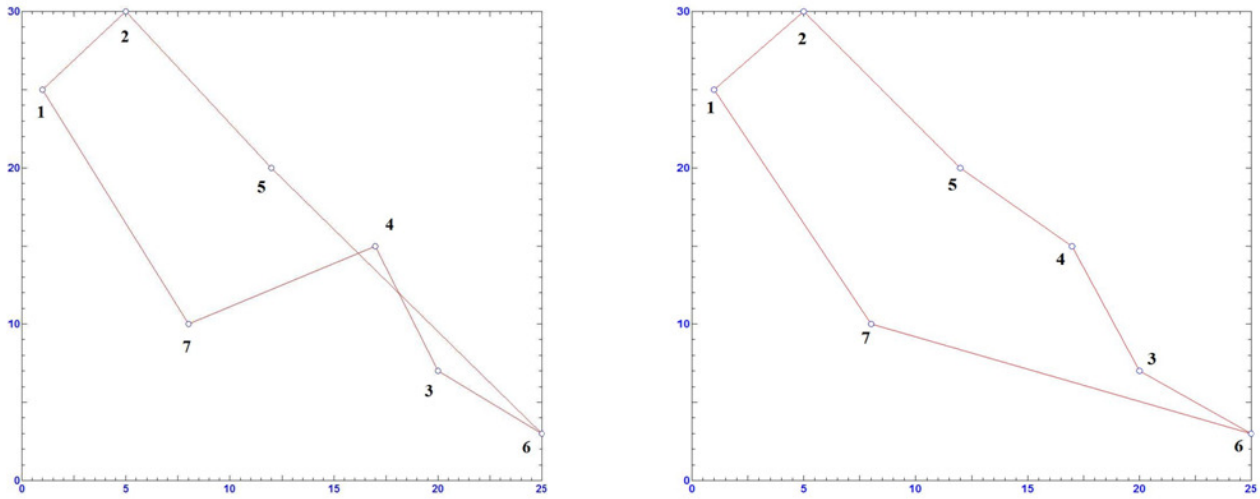

Figura 11 – Comparação das rotas.

5 CONSIDERAÇÕES FINAIS

Como sempre ocorreu na história do homem, a natureza foi a inspiração de várias inovações tecnológicas. A computação evolutiva foi baseada nas teorias de evolução das espécies propostas por Charles Darwin, que, em resumo, defendia que os indivíduos mais aptos ao ambiente em que viviam tinham mais chances de transmitir sua informação genética para a próxima geração. Baseado nesses preceitos, em 1975, John Holland, em seu livro "Adaptation in natural and artificial systems" propôs os Algoritmos Genéticos como um método heurístico baseado na "sobrevivência do mais apto".

A utilização dos algoritmos genéticos para a otimização se mostrou como uma poderosa ferramenta. Os algoritmos genéticos possuem um grande balanço entre direcionar a pesquisa das respostas para espaços de busca interessantes do ponto de vista da aptidão, e vasculhar todo o espaço de busca de resposta. Isso faz com que esses algoritmos se tornem versáteis, e possam ser utilizados para resolver problemas em que a descrição matemática seja impossível, pois os algoritmos genéticos se baseiam no que se quer ver na resposta e não na descrição do problema em si. Mesmo sendo reconhecidamente uma ferramenta poderosa, não há nenhuma garantia de que os algoritmos genéticos funcionem igualmente sempre, pois estes, na verdade, são altamente dependentes dos parâmetros impostos pelo programador, como foi visto no caso do aumento da população mostrado na seção de resultados obtidos, onde a simples mudança no número da população representou um maior custo computacional e uma relativa melhora no resultado final. Outro fato a ser considerado é que também não há garantia de que o ótimo global seja encontrado, apenas pode-se aumentar a chance de se encontrá-lo.

Como vantagens, os algoritmos genéticos são fáceis de serem implementados e adaptáveis para outros problemas, pois geralmente a estrutura geral é bem definida; não é necessário um conhecimento matemático aprofundado sobre o problema, apresentam várias possíveis soluções (cada indivíduo na população é considerado como uma resposta em potencial), não necessariamente encontra a solução ideal, mas mostra uma lista de boas soluções para o problema, apresenta um bom desempenho em problemas de otimização em larga escala, dentre outros. As desvantagens dos algoritmos genéticos são: a dificuldade em se definir a função objetivo, a escolha de vários parâmetros que podem influenciar na resposta (tamanho da população, taxa de mutação, taxa de crossover, método de seleção etc.), tornado

o trabalho custoso, dificuldade em se determinar o critério de parada, a não garantia de se encontrar o ótimo global etc.

O problema do caixeiro viajante encontra aplicações em várias áreas, como por exemplo na logística de entregas, na indústria do petróleo, robótica submarina e engenharia em geral. A otimização da rota tomada implica redução do caminho percorrido que pode trazer vantagens como economia de combustível, redução do desgaste de peças, e portanto economia de dinheiro. Logicamente, no mundo real, vários outros fatores devem ser analisados, como por exemplo a prioridade de execução e o tempo. Muitas vezes a rota mais curta não significa necessariamente a rota percorrida em um menor tempo, pois no caso para a robótica submarina, podem haver contra correntes no percurso, prioridade de se operar uma estação primeiro que as outras devido a gravidade da situação, barreiras etc. Portanto, tais dificuldades impostas podem provocar redução da velocidade, aumento do tempo gasto, desgaste elevado dos elementos de máquinas dentre outros, mesmo em uma rota mais curta. Consequentemente, tais fatores deveriam ser levado em consideração no momento de implementação da função objetivo, dependendo do que for requerido.

Para os fins do nosso estudo, em que a função objetivo levou em conta apenas a redução do caminho percorrido, com a restrição de que nenhum ponto de operação fosse visitado mais de uma vez, excetuando-se o ponto de retorno, o algoritmo apresentou resultados satisfatórios, visto que a redução do caminho percorrido foi realmente significativa nos casos estudados. Como vantagens do algoritmo implementado, podemos citar a rápida convergência, poder de redução do caminho percorrido, versatilidade de se poder entrar com as coordenadas em duas dimensões, mudar facilmente o número da população, número de pontos e o número de iterações, tornando-o adaptável para qualquer problema do tipo em que se deseje encontrar a menor rota possível. Como desvantagens podemos citar a dependência da população inicial, a fraca influência da mutação no processo de busca de melhores soluções. Em alguns casos onde o número de iterações era muito grande em relação a população (o triplo, por exemplo) e a taxa de mutação alta, foi-se constatado a criação de rotas inválidas.

6 REFERÊNCIAS

1 BESSA, Wallace Moreira. DUTRA, Max Suell. KREUZER, Edwin. REIS, Ney Robinson Salvi. **Projeto e Construcão de um Veículo Robótico Submarino Teleoperado Via Internet.** Congresso Nacional de Engenharia Mecânica (CONEM), Belém, 2004.

2 CHEDIAK, Karla. **O problema da individuação na biologia à luz da determinação da unidade de seleção natural**. *Sci. stud.* [online]. 2005, vol.3, n.1, pp. 65-78. ISSN 1678-3166.

3 CORDEIRO, Filipe Rolim. **Uma Ferramenta de Simulação Para Otimização Multi-Objetiva Evolucionária**. Trabalho de Conclusão de Curso, Universidade de Pernambuco, Recife, 2008.

4 ENGINEERING Design Centre, New Castle University. **Roulette Wheel Selection.** Disponível em: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/. Acesso em: 28 de Outubro de 2011.

5 FEITOSA, Rubens Gonçalves. SENA, Elda. **Algoritmos Genéticos e Problemas de Transporte.** Disponível em: http://www.ebah.com.br/content/ABAAABfGMAH/artigo-algoritmos-geneticos-problemas-transporte. Acesso em: 13 de julho de 2011.

6 GOOGLE imagens. Disponível em: http://www.google.com.br/imghp?hl=pt-BR&tab=wi. Acessado em: 15 de dezembro de 2011.

7 LIMA, Ednaldo Oliveira. **Algoritmo Genético Híbrido Aplicado a Otimização de Funções.** Monografia, UFLA, Lavras, 2008. Disponível em: http://www.bcc.ufla.br/monografias/2008/Algoritmo_genetico_hibrido_aplicado_a_otimizacao_de_funcoes.pdf. Acesso em: 14 de julho de 2011.

8 LINDEN, Ricardo. **Algoritmos Genéticos, Uma Importante Ferramenta de Inteligência Computacional**, segunda edição, Brasport, Rio de Janeiro, 2008.

9 MATOS, Renan Dupas. **Utilização de Algoritmo Genético Para Resolução do Problema de Geraçao de Horários.** Trabalho de Conclusão de Curso, Universidade Estadual de Londrina, Londrina, 2007.

10 OBTIKO. **Introduction to Genetic Algorithm**. Disponível em: http://www.obitko.com. Acesso em: 02 de Novembro de 2011.

11 OLIVEIRA, Matheus Sousa. **Um Estudo Sobre Algoritmos Genéticos**. Monografia, UNICEUMA, São Luís, 2007. Disponível em:

http://pt.scribd.com/doc/50116219/Monografia-Algoritmos-Geneticos-Matheus-Final. Acesso em: 13 de julho de 2011.

12 SARAIVA, Felipe de Oliveira. OLIVEIRA, Antônio Costa. **Uma Comparação Empírica de Operadores de Crossover Para o Problema de Job Shop Com Datas de Entregas.** Disponível em: http://www.abepro.org.br/biblioteca/enegep2010_TN_STO_118_772_15277.pdf. Acesso em: 03 de Novembro de 2011.

13 SILVA, Anderson Freitas. OLIVEIRA, Antônio Costa. **Algoritmos Genéticos: alguns experimentos com os operadores de cruzamento ("crossover") para o problema do caixeiro viajante assimétrico.** Disponível em: http://www.abepro.org.br/biblioteca/ENEGEP2006_TR460314_7093.pdf. Acesso em: 11 de dezembro de 2011.

14 SIVANANDAM, S. N. DEEPA, S. N. **Introduction to Genetic Algorithms**, Springer, New York, 2008.

15 VON ZUBEN, Fernando J. **Computação Evolutiva: Uma Abordagem Pragmática.** Disponível em: ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/tutorial/tutorialEC.pdf. Acesso em: 13 de julho de 2011.

16 WIKIPEDIA: a enciclopédia livre. **Problema do caixeiro viajante**. Disponível em: http://pt.wikipedia.org/wiki/Problema_do_caixeiro_viajante. Acesso em: 15 de Julho de 2011.

APÊNDICE A - Código do programa

```
//UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE - UFRN
//PRÓ-REITORIA DE GRADUACÃO - PROGRAD
//CENTRO DE TECNOLOGIA - CT
//DEPARTAMENTO DE ENGENHARIA MECÂNICA - DEM
//TRABALHO DE CONCLUSÃO DE CURSO
//ALUNO: CAIO JÚLIO CÉSAR DO VALE FERNANDES DA SILVA
//ORIENTADOR: DR. WALLACE MOREIRA BESSA
//ALGORITMOS GENÉTICOS: UMA APLICAÇÃO A ROBÓTICA SUBMARINA

//Bibliotecas
#include<iostream>
#include<math.h>
#include <time.h>
#include <fstream.h>
#include <stdio.h>

using namespace std;
// Variáveis Globais
int population=20;
int cromolength=8;
float medium[20];

// Programa Principal
int main()
{   cout<<"TRABALHO DE CONCLUSAO DE CURSO"<<endl<<endl;
   cout<<"ALGORITMOS GENETICOS: UMA APLICACAO A ROBOTICA
SUBMARINA"<<endl<<endl;;
   cout<<"ALUNO: CAIO JULIO CESAR DO VALE FERNANDES DA
SILVA"<<endl<<endl;
   cout<<"ORIENTADOR: WALLACE MOREIRA BESSA"<<endl<<endl;
```

```
    srand((int) time(NULL));
    unsigned int rota[population][cromolength];
    unsigned int parent[population];
    int pop, flag;

//Iniciar população aleatoriamente
   for(int k=0;k<population;k++)
    {
    rota[k][0]=1;    //Representa que sai de 1
    rota[k][cromolength-1]=1;//Representa que volta para 1
    for(int i=1;i<cromolength-1;i++)
    {
        pop=rand()%cromolength; //Gera os numeros randomicos
        flag=0;        //Artifício

       if(pop==0)      //Para não confundir os cromossomos com cidade 0
       {pop=pop+1;}
        for(int j=0;j<i;j++)
        {
         if(rota[k][j]==pop)      //Condição para não repetir numeros
         {
           flag=1;
           break;          //Para o loop e vai pro próximo código
         }
        }

        if(flag==1)
        {i=i-1; //Para garantir a saída do número certo de alelos
                continue;  //Começa um novo loop
                }
          rota[k][i]=pop;
       }
}
cout<<endl;
```

```
//Distâncias
 float P[cromolength-1][2];
 cout<<"Entre com as coordenadas x "<<endl; //Entrada das coordenadas X
  for(int i=1;i<cromolength;i++)
 {
        cin>>P[i][1];

 }
  cout<<"Entre com as coordenadas y "<<endl; //Entrada das coordenadas Y
   for(int i=1;i<cromolength;i++)
 {
        cin>>P[i][2];
 }
 cout<<endl<<endl;
for(int super=1; super<=20; super++) //Número de iterações
{
int d[population][cromolength-1]; //Distâncias de cidade para cidade
int somatorio_rota[population]; //Distâncias de cada rota (cromossomo)
   for(int i=0; i<population; i++)
   {
         somatorio_rota[i]=0;
   }
int somatorio_total=0; //Somatorio de todos as aptidões da população
int media=0;
 for(int k=0; k<population; k++)  //Cálculo das distâncias
{
        for(int j=0; j<cromolength-1; j++)
        {
              d[k][j]=sqrt(pow(P[rota[k][j]][1]-P[rota[k][j+1]][1],2)+pow(P[rota[k][j]][2]-
P[rota[k][j+1]][2],2)); //calculo da distâncias entre os pontos
              somatorio_rota[k]=somatorio_rota[k]+d[k][j];
              somatorio_total=somatorio_total+d[k][j];
              media=somatorio_total/population;
```

```
        }
}

//Atribuição da media
       medium[super]=media;
//Probabilidade de seleção
float probi[population]; //Probabilidade
float exp_count[population]; //Contagem esperada
int percent[population]; //Porcentagem
         for(int i=0;i<population;i++)
        {
             probi[i]=1-((somatorio_rota[i])/(somatorio_total)); //"1-..."Adaptação para a
minimização.
             percent[i]=probi[i]*100;
             exp_count[i]=1-((somatorio_rota[i])/(somatorio_total/population)); //Da mesma
forma.
        }

   //Declaração de mais variáveis auxiliares
Int      min1=somatorio_rota[0],     min2=somatorio_rota[0],     i,max1=somatorio_rota[0],
max2=somatorio_rota[0];
           int pos[2]; //posições de mínimo
           pos[0]=0;
           pos[1]=0;
           int posmax[2]; //posições de máximo
           posmax[0]=0;
           posmax[1]=0;

   //Calculando a posição do mínimo
           for(i=1;i<population;i++)
           {
                    if(somatorio_rota[i]<min1)
                    {
                             min1=somatorio_rota[i];
```

```
                pos[0]=i;
            }
        }

    //Calculando a posição do segundo mínimo
        for(i=1;i<population;i++)
        {
            if(somatorio_rota[i]<min2&&i!=pos[0])
            {
                min2=somatorio_rota[i];
                pos[1]=i;
            }
        }

        //Calculando a posição de máximo
        for(i=1;i<population;i++)
        {
            if(somatorio_rota[i]>max1)
            {
                max1=somatorio_rota[i];
                posmax[0]=i;
            }
        }

    //Calculando o segundo máximo
        for(i=1;i<population;i++)
        {
            if(somatorio_rota[i]>max2&&i!=posmax[0])
            {
                max2=somatorio_rota[i];
                posmax[1]=i;
            }
        }
```

```
//Seleção do pontos de crossover
 int crosspt1, crosspt2; //Pontos de crossover
 int pais[2][cromolength]; //Pais
 int tempo1[2][cromolength], tempo2, tempo, cnt,j; //variáveis auxiliares
 cnt=0;
      do
      {
            crosspt1=rand()%cromolength; //Seleção dos pontos de crossover de forma
aleatória
      }while(crosspt1>2) ;
      do
      {
            crosspt2=rand()%cromolength; //Seleção dos pontos de crossover de forma
aleatória
      }while(crosspt2<=3);

      for(j=0;j<cromolength;j++)
      {
            pais[0][j]=rota[pos[0]][j]; //Pai 1, melhor indivíduo da população
      }
      for(j=0;j<cromolength;j++)
      {
            pais[1][j]=rota[pos[1]][j]; //Pai 2, segundo melhor indivíduo da população
      }
      for(int j=crosspt1+1;j<=crosspt2;j++)
      {  cnt++;
            tempo1[1][cnt]=pais[0][j];
            tempo1[0][cnt]=pais[1][j];
            tempo=pais[0][j];
            pais[0][j]=pais[1][j];
            pais[1][j]=tempo;
  }
```

```
//Crossover PMX
  int k,m; //Variáveis auxiliares
       for(m=0;m<2;m++)
       {
            for(i=0;i<crosspt1+1;i++)   //Analisando a rota antes do crosspt 1
            {
                 for(j=0;j<cnt;j++)   //Comparando a rota dentro dos pontos de crossover
                 {
                      if(pais[m][i]==tempo1[m][j])
                      {
                           if(m==0)   //Para criança 1
                           {
                                tempo2=tempo1[1][j];  //Pegar a rota da criança 2
                                for(k=0;k<cromolength;k++)
                                {
                                     if(pais[m][k]==tempo2) //Se houver erro, repita o processo
                                     { tempo2=pais[1][k];
                                       k=0;
                                     }
                                }
                                pais[m][i]=tempo2;     //Atribuindo os valores à criança
                           }
                           else  //Para a criança 2
                           {
                                tempo2=tempo1[0][j];
                                for(k=0;k<cromolength;k++)
                                {
                                     if(pais[m][k]==tempo2) //Se houver erro, repita o processo
                                     {tempo2=pais[0][k];
                                      k=0;
```

```
                                        }
                                    }
                                    pais[m][i]=tempo2; //Atribuindo os valores à
criança
                                }
                            }
                        }
                    }
                }
                for(m=0;m<2;m++)
                {
                    for(i=crosspt2+1;i<cromolength;i++)  //Checando a rota depois do crosspt 2
                    {
                        for(j=0;j<cnt;j++)  //Comparando a rota dentro dos pontos de crossover
                        {
                            if(pais[m][i]==tempo1[m][j])
                            {
                                if(m==0)  //para criança 1
                                {
                                    tempo2=tempo1[1][j];  //Pegar a rota da criança 2
                                    for(k=0;k<cromolength;k++)
                                    {
                                        if(pais[m][k]==tempo2) //Se houver erro,
repita o processo
                                        {tempo2=pais[1][k];
                                         k=0;
                                         }
                                    }
                                    pais[m][i]=tempo2;    //Atribuindo os valores à
criança
                                }
                                else  //para criança 2
                                {
```

```
                                                    tempo2=tempo1[0][j];
                                                    for(k=0;k<cnt;k++)
                                                    {
                                                            if(pais[m][k]==tempo2) //Se houver erro,
repita o processo
                                                            {tempo2=pais[0][k];
                                                             k=0;
                                                             }
                                                    }
                                                    pais[m][i]=tempo2; //Atribuindo os valores à
criança
                                            }
                                    }
                            }
                    }
            }

//Aprimoramento das restrições
int conta=0;
int a[6],b,c,v;
b=0;
//Primeira parte, identificação dos locus
for(i=1; i<cromolength-1;i++)
{conta=0;
        for(j=1; j<cromolength-1;j++)
        {
                if(pais[0][i]==pais[0][j])
                { a[i]=j;
                 conta++;
                 if(conta==2)
                 {
                 b=j;
                 }
                 }
```

```
        }
}
for(i=1; i<cromolength-1;i++)
{
        if(pais[0][a[i]]==pais[0][b])
        {
          c=i;
          v=i;
        }
}
int conta1=0;
int g[6],e,f,z;
e=0;
for(i=1; i<cromolength-1;i++)
{conta1=0;
        for(j=1; j<cromolength-1;j++)
        {
                if(pais[1][j]==pais[1][i])
                {g[i]=j;
                 conta1++;
                 if(conta1==2)
                 {
                 e=j;
                 }
                 }
        }
}
if( (b!=0) && (e!=0) )
{
for(i=1; i<cromolength-1;i++)
{
        if(pais[1][g[i]]==pais[1][e])
        {
          f=i;
```

```
        }
}
v=pais[0][c];
z=pais[1][f];
pais[1][f]=v;
pais[0][c]=z;
}

//Segunda parte, troca dos alelos
int conta2=0;
int ab[6],bc,cd,ef;
bc=0;
for(i=1; i<cromolength-1;i++)
{conta2=0;
        for(j=1; j<cromolength-1;j++)
        {
                if(pais[0][i]==pais[0][j])
                { ab[i]=j;
                 conta2++;
                 if(conta2==2)
                 {
                 bc=j;
                 }
                 }

        }
}
for(i=1; i<cromolength-1;i++)
{
        if(pais[0][ab[i]]==pais[0][bc])
        {
          cd=i;
          ef=i;
        }
```

```
}
int conta3=0;
int gh[6],ij,kl,mn;
ij=0;
for(i=1; i<cromolength-1;i++)
{conta3=0;
        for(j=1; j<cromolength-1;j++)
        {
                if(pais[1][j]==pais[1][i])
                {gh[i]=j;
                 conta3++;
                 if(conta3==2)
                 {
                 ij=j;
                 }
                 }
        }
}
if( (bc!=0) && (ij!=0) )
{
for(i=1; i<cromolength-1;i++)
{
        if(pais[1][gh[i]]==pais[1][ij])
        {
          kl=i;
        }
}
v=pais[0][bc];
z=pais[1][ij];
pais[1][ij]=v;
pais[0][bc]=z;
}
```

```
 //Mutação
   int numero, numero1, numero2;
   for(int i=0; i<=1; i++)
  {
   numero=rand()%1000;
   if(numero==10) //Probabilidade de ocorrencia de 0,1%
   {
       numero1=rand()%cromolength;
       numero2=rand()%cromolength;
       if(numero1==0||numero1==1)
       {numero1==2;}
       if(numero2==0||numero2==1)
       {numero2==3;}
       pais[i][numero1]=pais[i][numero2];  //Indivíduo mutado
   }
   }

  //Substituição: Inclusão das crianças no lugar dos piores indivíduos
   for(int j=0;j<cromolength;j++)
           {
                   rota[posmax[0]][j]=pais[0][j];
                   rota[posmax[1]][j]=pais[1][j];
           }
 }

//Apresentação da última população
cout<<"Ultima populacao"<<endl<<endl;
for(int i=0; i<population; i++)
{
for(int j=0;j<cromolength;j++)
           {
           cout<<rota[i][j]<<"-"; //Nova população
           }
cout<<endl;
```

```
}
cout<<endl;

//Apresentação dos parâmetros para comparação
cout<<" Media das distancias "<<endl<<endl;
for(int i=1; i<=20; i++)
{
      cout<<" Populacao "<<i<<"  "<<medium[i]<<endl;
}
cout<<endl<<" Decrescimo relativo "<<((medium[1]-medium[20])/medium[1])*100<<endl;
//decréscimo relativo
   cin.get();
   cin.get();
   return 0;
} //fim do programa
```

APÊNDICE B – Tela de Saída

Exemplo de tela de saída do algoritmo implementado. Caso população igual a 100, número de pontos de visitação igual a 7 e 70 iterações.

TRABALHO DE CONCLUSAO DE CURSO
ALGORITMOS GENETICOS: UMA APLICACAO A ROBOTICA SUBMARINA
ALUNO: CAIO JULIO CESAR DO VALE FERNANDES DA SILVA
ORIENTADOR: WALLACE MOREIRA BESSA

Entre com as coordenadas x
1
5
20
17
12
25
8
Entre com as coordenadas y
25
30
7
15
20
3
10
Ultima populacao
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-
1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

1-7-6-3-4-5-2-1-

Media das distancias

Populacao 1 110

Populacao 2 109

Populacao 3 107

Populacao 4 106

Populacao 5 105

Populacao 6 104
Populacao 7 102
Populacao 8 101
Populacao 9 100
Populacao 10 99
Populacao 11 98
Populacao 12 97
Populacao 13 96
Populacao 14 95
Populacao 15 94
Populacao 16 93
Populacao 17 92
Populacao 18 91
Populacao 19 90
Populacao 20 89
Populacao 21 88
Populacao 22 88
Populacao 23 87
Populacao 24 86
Populacao 25 85
Populacao 26 85
Populacao 27 84
Populacao 28 83
Populacao 29 83
Populacao 30 82
Populacao 31 81
Populacao 32 81
Populacao 33 80
Populacao 34 79
Populacao 35 79
Populacao 36 78
Populacao 37 78
Populacao 38 77
Populacao 39 77

Populacao 40 76
Populacao 41 76
Populacao 42 75
Populacao 43 75
Populacao 44 74
Populacao 45 74
Populacao 46 74
Populacao 47 73
Populacao 48 73
Populacao 49 73
Populacao 50 73
Populacao 51 73
Populacao 52 73
Populacao 53 73
Populacao 54 73
Populacao 55 73
Populacao 56 73
Populacao 57 73
Populacao 58 73
Populacao 59 73
Populacao 60 73
Populacao 61 73
Populacao 62 73
Populacao 63 73
Populacao 64 73
Populacao 65 73
Populacao 66 73
Populacao 67 73
Populacao 68 73
Populacao 69 73
Populacao 70 73

Decrescimo relativo 33.6364

APÊNDICE C - Comportamento das Distâncias Médias ao Longo das Iterações.

Para o primeiro caso, com 7 pontos de visitação, população de tamanho igual a 20 e 20 iterações. Não necessariamente e mesma resposta será encontrada.

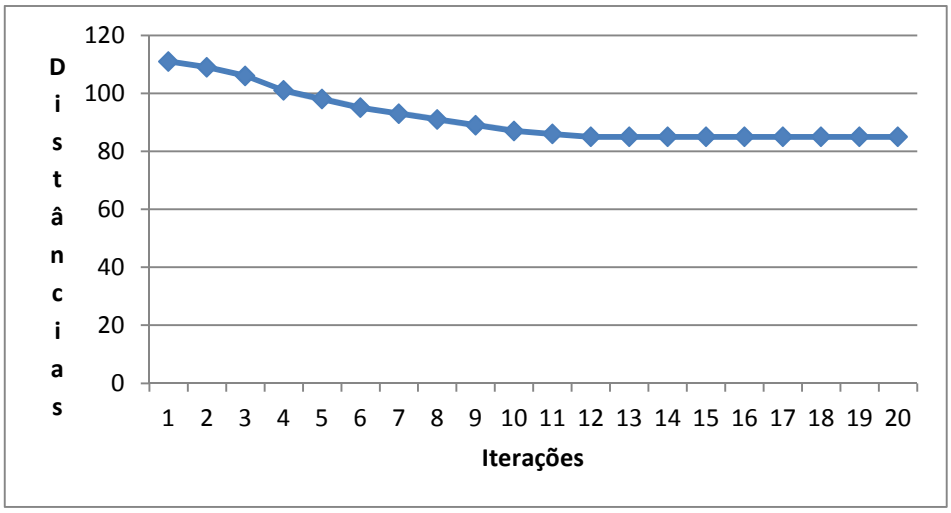

Figura 12 – Distância x Iterações

Percebe-se que após 12 iterações o algoritmo converge para uma distância média de 85. Apenas para reforçar a idéia do comportamento, o número de iterações foi aumentado para 100, como pode ser visto no gráfico abaixo.

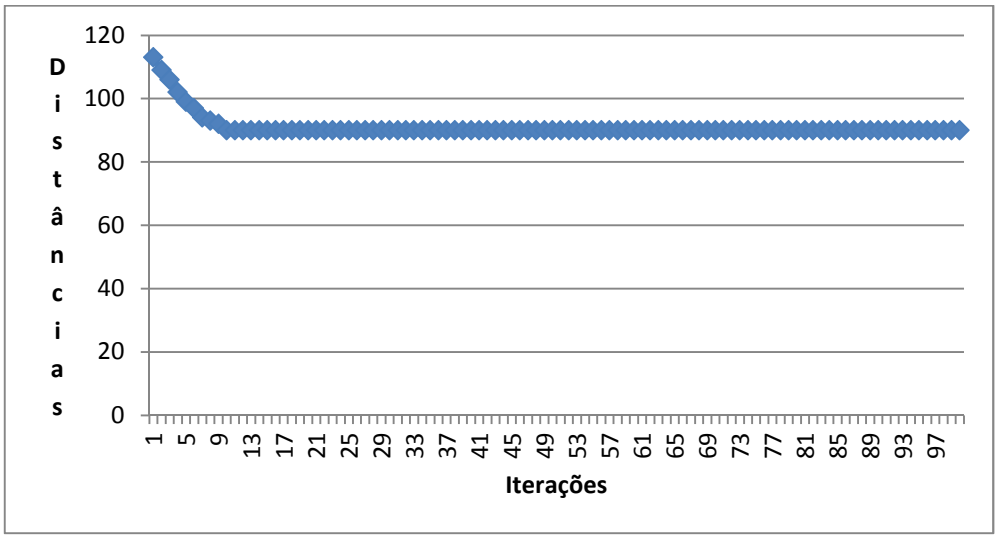

Figura 13 – Distância x Iterações

Após 10 iterações o algoritmo convergiu para a distância média de 90.

Para o segundo caso, com 7 pontos de visitação, população de tamanho igual a 100 e número de iterações igual a 70.

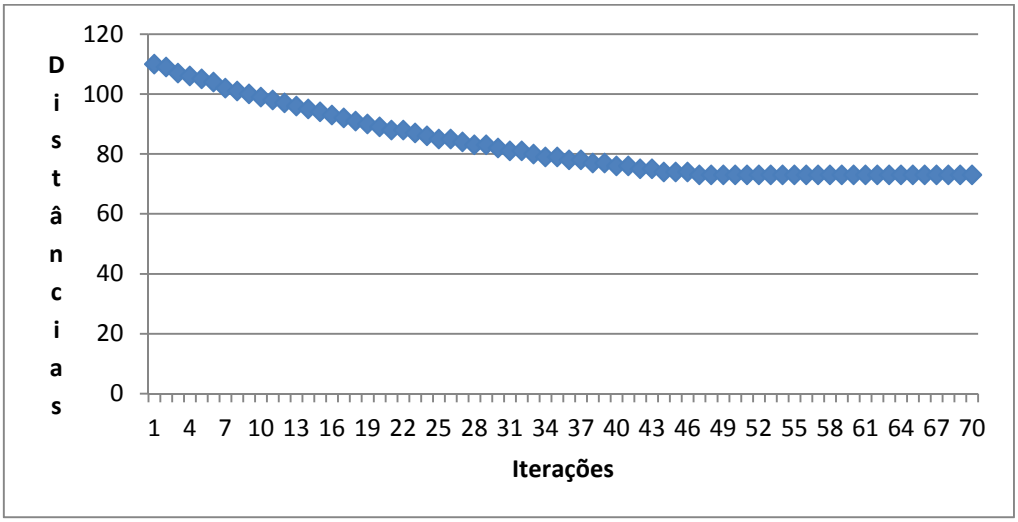

Figura 14 – Distância x Iterações

O algoritmo demora um pouco mais para convergir devido ao maior número de rotas possíveis a serem analisadas. Percebe-se que o algoritmo converge na iteração de número 47 para distância de 73. Para reforçar a idéia, aumentamos o número de iterações para 300, como pode ser visto no gráfico abaixo.

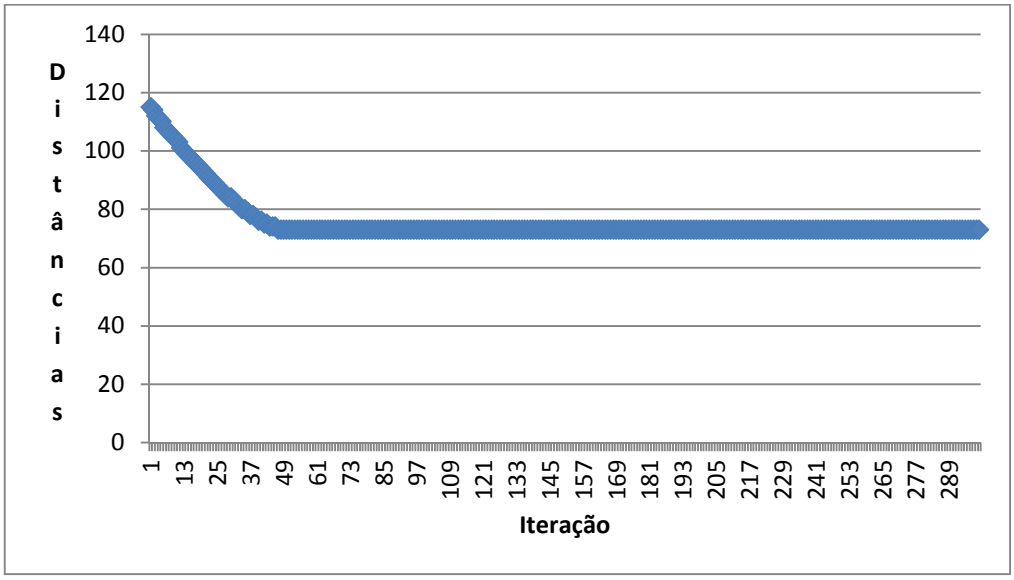

Figura 15 – Distância x Iterações

Após 47 iterações o algoritmo convergiu para a distância de 73 novamente.