Desempenho de um algoritmo genético de representação mista na solução do roteamento de caminhos em redes de sensores sem fio

Sadraque S. Viana
Departamento de Engenharia Eletrônica
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais 31270–901
Email: viana.sadraque@gmail.com

Igor Finelli
Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais
Florestal, Minas Gerais, 35690-000
Email: igorfinelli@hotmail.com

Vincent Barrault
Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais 31270–901
Email: vincent.barrault@gmail.com

***Resumo*—Neste artigo, sugerimos a utilização de um algoritmo genético de representação mista para a solução do problema de caminho ótimo em redes de sensores sem fio. Descrevemos o problema de otimização, dando sua formulação matemática. A representação dos indivíduos e os operadores de cruzamento, seleção e mutação são brevemente descritos. Executamos testes gerais dos parâmetros do algoritmo, procurando a sua melhor configuração. Apresentamos os resultados obtidos e nossas considerações finais, incluindo possíveis modificações na implementação do algoritmo.**

I. Introdução

Existem diversos casos em que é necessário realizar análises em diversos pontos de regiões isoladas. Para tal, redes de sensores sem fio tem se tornado uma ótima solução. Porém, não são todas as situações em que é possível efetuar a transferência de dados entre todos os sensores e o receptor, muitas vezes pelo próprio raio de comunicação dos sensores, que por causa dos recursos de energia limitada e distâncias grandes não possibilitam a comunicação entre todos eles.

Apesar de não ser uma solução definitiva, os sistemas que se utilizam da comunicação sem fio têm apresentado vantagens consideráveis. Assim, uma das possíveis adaptações é utilizar um receptor móvel que siga uma rota ótima, passando pelos raios de comunicação dos sensores e recebendo os respectivos dados. Essa solução, além de apresentar um menor caminho frente à uma coleta pontual dos sensores, também é uma solução para os possíveis erros de localização.

Neste trabalho, tratamos do problema de otimização da rota a ser percorrida em uma rede de sensores sem fio. Escolhemos abordar a solução deste problema utilizando algoritmos genéticos. A principal vantagem desta abordagem é o fato de ela permitir explorar tanto a sequência de visita aos sensores como a otimização de rotas simultaneamente. As principais contribuições deste trabalho são abordar os aspectos relacionados à utilização de representação mista em algoritmos genéticos para a solução de problemas que envolvem, simultaneamente, otimização real e combinatória, como é o caso do problema de roteamento de caminhos.

Este trabalho está estruturado da seguinte forma. Na seção II, descrevemos os principais aspectos relacionados ao problema de otimização, fornecendo sua formulação matemática. Na seção III, tratamos especificamente do algoritmo de otimização, descrevendo detalhadamente os seus operadores. Na seção IV, mostramos os resultados advindos dos testes executados com o algoritmo. Por fim, encerramos com nossas considerações finais na seção V.

II. O problema de roteamento de caminhos em redes de sensores sem fio

A. Breve descrição

Consideremos uma rede de sensores sem fio distantes geograficamente um dos outros, distribuídos em um espaço bidimensional. Cada nó sensor pode comunicar-se em um certo raio de ação, função da energia que ele ainda contém. Os raios de comunicação destes sensores não apresentam interseção, de modo que é inviável que eles comuniquem-se diretamente. A figura 1 esquematiza esta distribuição.

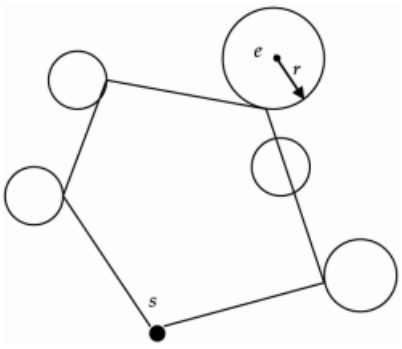

Figura 1: Distribuição dos nós sensores. Imagem de [1]

Necessitamos realizar a coleta dos dados dos sensores. Uma possível solução é utilizarmos um robô que, partindo de um ponto inicial, passe pelo raio de comunicação de todos os nós sensores e volte à sua base.

Queremos encontrar qual o caminho ótimo pelo qual o robô deve passar, de modo a coletar os dados de todos os sensores percorrendo a menor distância possível, a fim de minimizar custos, por exemplo.

O cenário descrito anteriormente ocorre diversas vezes em situações práticas, não só no caso de redes de sensores sem fio. Outra possível aplicação inclui o envio de suprimentos por via aérea, por exemplo. Surge daí a importância de se resolver esta classe de problemas .

B. Formulação matemática

Seja o espaço bidimensional onde os sensores estão alocados. Cada nó sensor é descrito pelas variáveis c e r, onde c é ponto onde o sensor está e r é o seu raio de comunicação, função de sua quantidade de energia. Consideremos que o robô parte da posição s, e que existam n sensores distribuídos no espaço.

Assim, o problema é completamente descrito por:

$$\{s, (c_1, r_1), (c_2, r_2), ..., (c_n, r_n)\} \quad (1)$$

É importante destacar aqui que a variável c é um ponto, e portanto é composta de duas variáveis reais x e y.

Depois de interceptar o raio de comunicação de um nó sensor, o robô pode proceder para coletar os dados de outro nó sensor. Assim, podemos simplificar o caminho percorrido como a conexão, através de segmentos de reta, entre o ponto inicial s e n pontos sucessivos de interceptação com os raios de comunicação. Esta situação é descrita na figura 2.

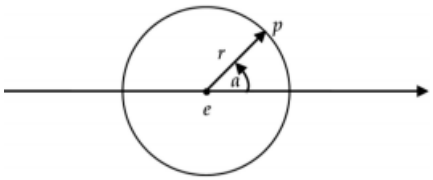

Figura 2: Ponto de interceptação do robô com um nó sensor. Imagem de [1]

Desta forma, chamando de p_i os n pontos de interceptação do robô com os raios de ação dos sensores, temos a seguinte função objetivo:

$$min\ f(x) = dis(s, p_1) + dis(s, p_n) + \sum_{i=1}^{n-1} dis(p_i, p_{i+1}) \quad (2)$$

onde dis é a função distância euclidiana.

Ressalta-se que neste problema, existem dois fatores a serem otimizados. Devemos otimizar a ordem dos pontos p_i e a melhor escolha dos pontos de interceptação, a fim de obtermos a rota com o menor percurso. Este é o motivo pelo qual escolhemos utilizar um algoritmo genético de representação mista, a fim de otimizarmos estes dois fatores simultaneamente.

Na próxima seção, descreveremos melhor o algoritmo de otimização utilizado.

III. Algoritmo de otimização

O problema descrito é da classe *TPSN - traveling salesman problem with neighborhoods*. Portanto, não existe uma solução determinística para o problema. Assim, escolhemos um método estocástico para realizar a otimização do problema descrito. Mais especificamente, escolhemos os algoritmos genéticos para a identificação heurística de uma boa solução.

A escolha deste método ocorreu por dois motivos. Como descrito anteriormente, esta classe de problemas não possui uma solução determinística ótima. Desta forma, a única maneira de solucioná-lo é utilizando heurísticas como os algoritmos evolucionários, onde se enquadram os algoritmos genéticos.

Outro fato que favorece a escolha destes algoritmos para realizarmos a otimização do problema é o seguinte. As soluções atuais para esta classe de problemas baseiam-se em inicialmente solucionar o problema das permutações para depois solucionar o problema do caminho ótimo, ou vice e versa. Isto é, o problema é dividido em partes, para depois ser solucionado. Desta maneira, perdemos a capacidade de explorar todo o espaço de busca, o que limita as soluções que podemos encontrar. Com os algoritmos genéticos, podemos solucionar os dois problemas de uma vez. Para tanto, utilizaremos uma codificação mista de nossos indivíduos.

A. Representação dos indivíduos

A primeira parte deles, inteira, representará a ordem das permutações. Já a segunda parte, de representação real, armazenará os ângulos que representam os pontos de interseção. Um exemplo da codificação a ser utilizada para a resolução do problema é mostrada na figura 3.

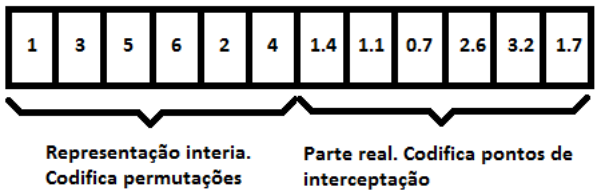

Figura 3: Representação dos indivíduos no algoritmo genético

B. Avaliação das soluções

A função objetivo do problema é aquela descrita na equação 2. A nossa função de adaptabilidade é definida como o inverso desta função, pois queremos minimizá-la.

C. O operador de cruzamento

Como estamos tratando de um problema de TSPN que, apesar de algumas diferenças na representação, já foi profundamente trabalhado na literatura, decidimos utilizar o operador *Partially Mapped Crossover*, PMX. Este operador foi desenvolvido por Goldberg e Lingle [2], e adaptado para facilitar o cruzamento da parte real, que necessita cruzar as informações referentes aos mesmos sensores. Para a parte real, implementamos o cruzamento real convexo com extrapolação.

1) Parte inteira: O cruzamento PMX é relativamente fácil de ser entendido e implementado. É capaz de se adaptar às características da representação utilizada, que conta com duas partes diferentes, real e inteira, em um mesmo indivíduo. Nesse cruzamento escolhe-se aleatoriamente um ponto de corte para dois pais e, pegando a sequência do pai-2 até o ponto de corte, busca-se colocar na mesma sequência até o ponto de corte o pai-1. Uma vez que os dois pais possuem a mesma sequência até o ponto de corte, junta-se a sequência do primeiro pai até o ponto de corte, com a nova sequência do segundo após o ponto de corte. O segundo filho é feito utilizando as mesmas operações nos pais originais só que utilizando a sequência pré-corte do pai-1 como base do filho-2.

A figura 4, retirada de [3], explica esquematicamente o cruzamento.

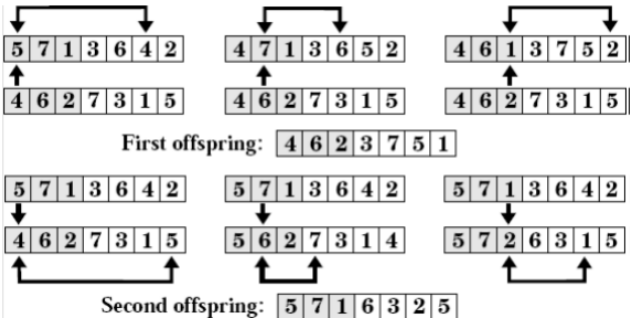

Figura 4: Esquematização do cruzamento PMX

Utilizamos a mesma lógica aplicada à parte inteira dos indivíduos. Cada troca nessa parte é acompanhada de uma respectiva troca na parte real. Observa-se que esse cruzamento realiza modificações nos pais para gerar os filhos, em especial o alinhamento das sequências até o ponto de corte, o que gerou a ideia de alinhar os pais na mesma sequência do filho antes de realizar o cruzamento da parte real.

Destaca-se o fato de que, ao final das trocas, o pai-1 já está na sequência do filho e o pai-2 possuí a mesma sequência até o ponto de corte. Dessa forma pode-se alterar somente a sequência pós-corte do pai-2 para obtermos os dois pais alinhados na sequência do filho. Nota-se também o fato de que cada filho é gerado de uma maneira separada, não se aproveitando as operações do primeiro filho para o segundo, o que implica em um custo computacional maior.

2) Parte real: Para o cruzamento da parte real aproveitamos o cruzamento da parte inteira para alinhar os pais na mesma sequência de um filho. Isso permitiu a realização do cruzamento convexo com extrapolação por variável, preservando a lógica dos ângulos referentes a cada sensor.

O cruzamento real, como descrito em [5], implementa um fator de extrapolação fixo, que busca evitar convergências e perda de diversidade na população. Assim, definimos uma extrapolação de 20%, o suficiente para aumentar o campo de soluções geradas.

Este operador de cruzamento se mostrou totalmente ineficiente quando os indivíduos pais eram idênticos, gerando filhos totalmente iguais ao pai. Esse tipo de relação pode ter sérias repercussões, dependendo do tipo de seleção utilizada. Caso o número de filhos idênticos aumente muito, as gerações serão cada vez mais saturadas e incapazes de gerar novas variações através do cruzamento, levando a uma convergência prematura do método.

D. O operador de mutação

Como já demonstrado no desenvolvimento dos algoritmos genéticos, as mutações são uma ferramenta de grande utilidade para aumentar a exploração e a diversidade da população.

Nesse trabalho foram implementados dois tipos de operadores de mutação separados: um para a parte inteira que representa a sequência a ser percorrida no cenário do problema e outro para a parte real, que representa os pontos de interceptação nos raios de cada ponto. Essa separação foi devido à própria representação dos indivíduos, que são vetores com metade do tamanho referentes a uma sequência de números inteiros e a outra metade referente a números reais, sendo necessária uma lógica para cada uma das partes.

1) Parte inteira: Este operador funciona, matematicamente, escolhendo uma posição aleatória da parte inteira do vetor que representa um indivíduo e realizando algum tipo de troca da informação dessa posição com outra. Esse operador deve garantir a particularidade da parte inteira de cada indivíduo, que não pode possuir números repetidos, pois isso significaria na prática um ponto que será visitado mais de uma vez, enquanto outro ponto não é visitado nenhuma e também garantir que para cada troca realizada a parte real também sofra a mesma troca, preservando as informações.

Foram testadas três formas diferentes de realizar as trocas de informação: a mutação subsequente, a mutação *swap mutation* e a mutação *insert mutation* descritas em [4].

A mutação subsequente foi pensada durante o trabalho, mas se mostrou muito local e pouco eficiente no funcionamento do GA. Basicamente uma posição da parte inteira é sorteada e o conteúdo nessa posição é trocado com o conteúdo da posição seguinte – a ultima posição troca com a primeira posição. Ela é descrita na figura 5.

Figura 5: Esquematização da mutação subsequente

A *swap mutation*, assim como a mutação citada anteriormente, se destaca pela facilidade de implementação, pois simplesmente sorteia duas posições e troca as informações entre elas, conforme esquematizado na figura 6. Ela não apresenta uma tendência local e, por alterar somente duas posições de forma aleatória, possuí a característica de quebrar a ordem da solução. Tais operações dificilmente geram bons resultados para as soluções que já possuem uma ordem lógica, mas garantem boa diversidade para o cruzamento.

O operador *insert mutation* é uma alternativa para a característica de quebra de ordem das demais mutações. Neste operador, duas posições são sorteadas e, ao mesmo

Figura 6: Esquematização da mutação *swap mutation*

tempo que se desloca o conteúdo das posições entre elas, move-se o conteúdo da segunda para a posição adjacente à primeira. Tal procedimento é exemplificado na figura 7

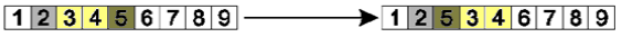

Figura 7: Esquematização da mutação *insert mutation*

2) Parte real: Da mesma forma que o operador para a parte inteira, testamos mais de uma técnica de mutação para verificar sua influência nos resultados do algoritmo genético e na busca por novas soluções. Foram implementados três operadores: mutação não-uniforme, mutação polinomial e mutação gaussiana [5].

A mutação gaussiana, que busca gerar perturbações a partir de uma distribuição gaussiana de probabilidade, mostrou forte dependência do parâmetro de desvio padrão. Devido à forma utilizada para representar os ângulos, um número real de 0 a 1, chegou-se à conclusão de que uma variação adequada deveria ser de apenas uma fração do máximo, entre 0.05 e 0.2. Esta decisão foi tomada de modo que a mutação seja capaz de gerar variações úteis para momentos de convergência das soluções, sacrificando a geração de soluções mais distantes. Apesar de muito simples, esse operador foi forneceu bons resultados, provavelmente por sempre forçar uma diversidade fixa na população, favorecendo a evolução através do cruzamento.

A mutação não-uniforme, que possuí uma dependência com o valor da geração atual em relação à geração final, foi considerada, primeiramente, como uma boa forma de tratar o problema, dado que mutações mais locais em gerações avançadas favoreceriam o refinamento da resposta. Os resultados, porém, mostraram tendências de convergência prematura do algoritmo, com poucos benefícios em gerações avançadas.

Por fim, testamos a mutação polinomial que, através de um parâmetro de controle N_m, produz variações maiores ou menores na população. Pretendíamos utilizar uma lógica adaptativa para variar o parâmetro de controle, buscando obter mutações adequadas tanto para o refinamento de soluções quanto para exploração. Entretanto, não desenvolvemos esta adaptação dos parâmetros, os resultados para parâmetros estáticos não mostraram nenhuma vantagem em cima dos já utilizados.

E. O operador de seleção

O operador de seleção é o último passo do algoritmo genético. É ele quem define as novas populações e, consequentemente, a convergência do algoritmo.

Decidimos de usar a seleção por torneio. Este operador é muito popular na área dos algoritmos genéticos, por sua eficiência comparando com a sua simplicidade de implementação. Implementamos também a seleção proporcional a aptidão. Entretanto, os resultados foram muito semelhantes, e decidimos ficar com a seleção por torneio.

Nas figuras 8 e 9, apresentamos a comparação destes dois métodos de seleção. A linha verde representa a média da função de *fitness* da população total a cada geração e a linha azul é o valor da *fitness* melhor individuo a cada geração. Utilizamos um cenário com 200 sensores e uma população 100 indivíduos. A diferença é visível na média da *fitness* da população. Podemos ver que, na seleção por roleta, a média pode aumentar ou diminuir ao longo das gerações, o que gera uma diferença entre a curva da média e a do melhor individuo. Na seleção por torneio, a média sempre diminui e rapidamente obtemos valores próximos da *fitness* do melhor individuo.

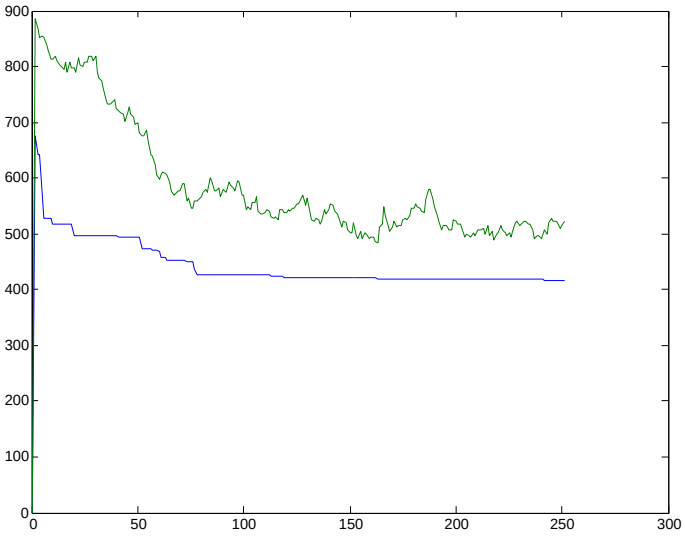

Figura 8: Evolução da *fitness* da população - seleção por roleta

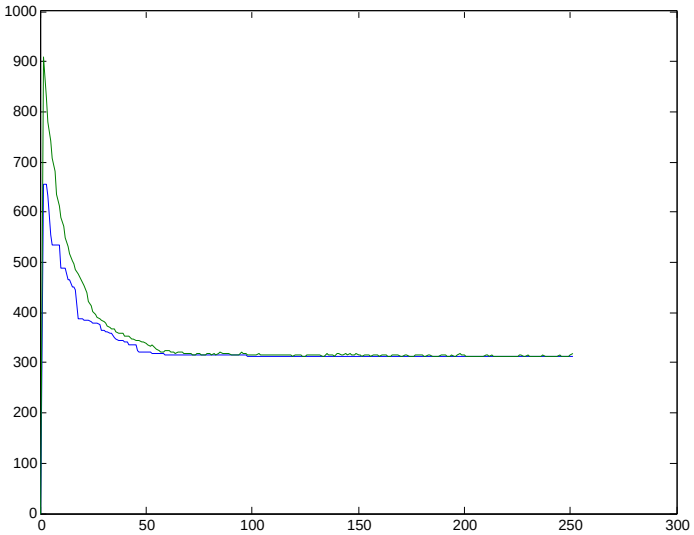

Figura 9: Evolução da *fitness* da população - seleção por torneio

A seleção por torneio possui um parâmetro P que controla a pressão seletiva do operador. Ele corresponde ao número de indivíduos que vão competir pela sobrevivência. Desta maneira, afeta diretamente o grau de aleatoriedade e determinismo na seleção. Caso existam muitos indivíduos em competição, a probabilidade indivíduos com menor grau de aptidão sobreviverem diminui, o que corresponde a um alto grau de pressão seletiva.

O caso oposto, isto é, apenas um indivíduo no torneio, gera uma seleção uniforme, sem pressão seletiva alguma. O número de indivíduos por torneio mais frequente é 2, e este foi o valor que escolhemos em nossa implementação. A grande vantagem deste operador é que existe uma probabilidade não desprezível de selecionarmos indivíduos com menor grau de aptidão, o que dificulta a perda prematura de diversidade.

IV. Análises de desempenho

A. Variação das soluções em relação à população inicial

O número de indivíduos é importante porque determina a diversidade no início do algoritmo, determinando também quão boas as soluções finais irão ser. Com um número pequeno de indivíduos, a sua diversidade inicial determinará a evolução das soluções. Por exemplo, para uma população de 100 indivíduos, depois de 100 gerações, podemos ter resultados completamente diferentes. As figuras a seguir comparam duas execuções do algoritmo, para a mesma distribuição de sensores.

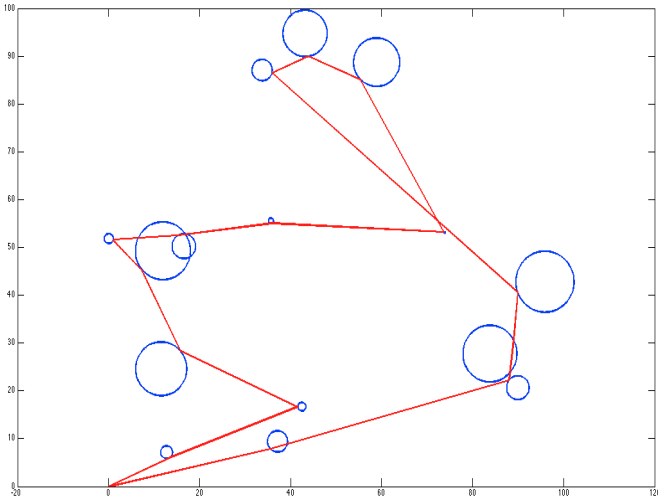

Figura 10: Solução ruim encontrada. 100 indivíduos

Utilizando a mesma distribuição anterior com 300 indivíduos, obtemos uma solução próxima da ótima em cerca de 50% das vezes de execução. Aumentando o número de indivíduos para 400, uma solução perto da ótima é encontrada em cerca de 90% das vezes.

B. Variação da fitness do melhor indivíduo e da média da população

Para uma população inicial de 50 indivíduos e um espaço de busca $[x\ y] = [10\ 100]$, observamos que, de 10 a 100 sensores,

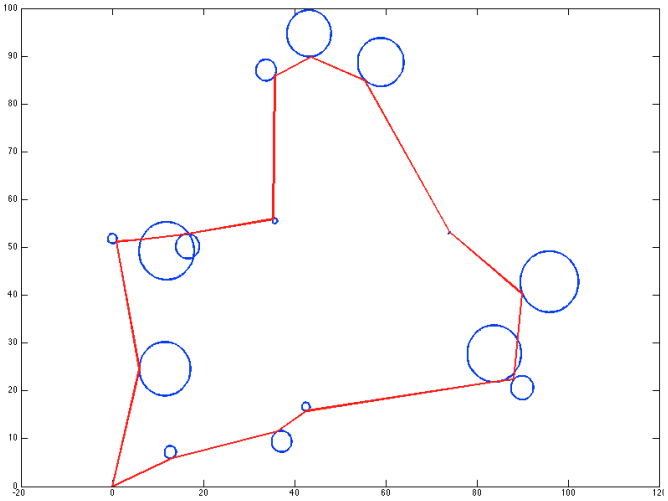

Figura 11: Solução boa encontrada. 100 indivíduos

o algoritmo começa a convergir e não evoluir muito a partir de 70 gerações. Com os mesmos parâmetros de população inicial e espaço de busca, executamos novos testes para 50, 100 e 500 sensores. Para 50 sensores, o algoritmo necessita cerca de 220 gerações para o melhor individuo e a média estabilizarem. Para 100 sensores, são necessárias 350 gerações e, para 500 sensores, cerca de 500 gerações. Estes resultados são mostrados nas figuras 12, 13 e 14. A linha verde representa a média da função de *fitness* da população total a cada geração e a linha azul é o valor da *fitness* melhor individuo a cada geração.

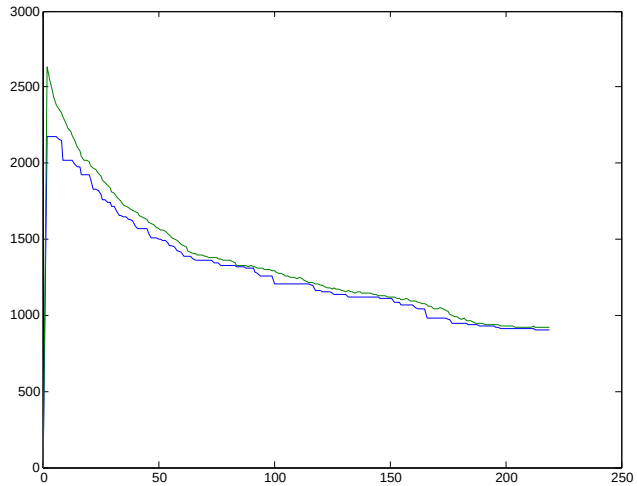

Figura 12: Estabilização do algoritmo - 50 sensores

C. Operadores e parâmetros do algoritmo

Para analisarmos a qualidade das soluções encontradas pelo algoritmo, utilizamos um cenário composto por 20 sensores gerados aleatoriamente dentro de um espaço de busca entre 0 e 100 para ambas as coordenadas x e y. Este cenário é mostrado na figura 15.

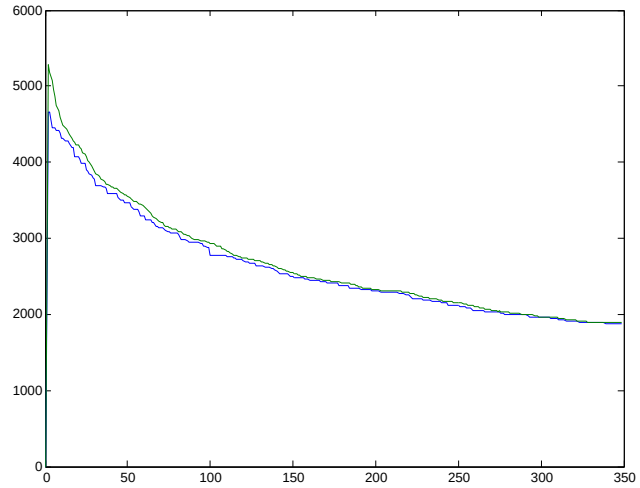

Figura 13: Estabilização do algoritmo - 100 sensores

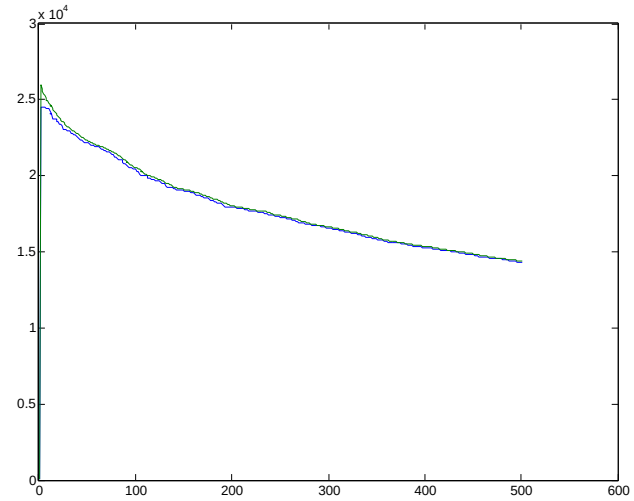

Figura 14: Estabilização do algoritmo - 500 sensores

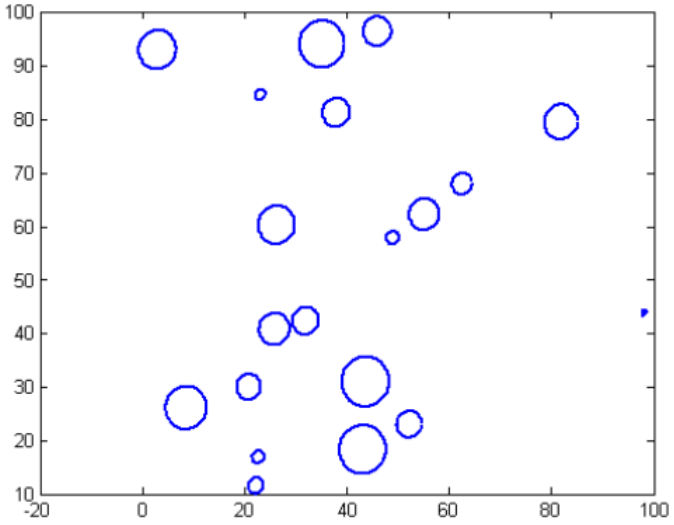

Figura 15: Distribuição dos sensores para o teste do algoritmo

Realizamos diversos testes com os parâmetros gerais do GA. Entre os parâmetros modificados podemos citar: número de indivíduos da população; número de iterações; probabilidade de mutação; tipo de mutação.

Observamos que o número de iterações deve possuir um valor mínimo para que o algoritmo chegue a convergir. Entretanto, observamos que, com a mutação gaussiana, o GA quase nunca muda a sequência de pontos após a convergência, o que mostra uma incapacidade de sair de soluções sub-ótimas.

Observamos também a ineficiência de um aumento da probabilidade de mutação, pois, no geral, acima de 50% as soluções ótimas diminuíram a capacidade de convergência. Provavelmente isto ocorreu devido a uma diversidade muito grande, incapaz de contribuir eficazmente para o refinamento das boas soluções já existentes.

Concluímos também que o operador sugerido de mutação por trocas subsequentes é ineficiente, pois os resultados sempre convergiam para soluções consideravelmente piores que os gerados pela mutação de inserção. O operador de mutação *swap mutation* se mostrou consideravelmente eficiente, produzindo soluções médias iguais ao operador de inserção. Entretanto, o operador de inserção provou-se o mais eficiente de todos.

A melhor solução encontrada é mostrada na figura 16.Ela obteve um valor de *fitness* de 393.98. Utilizamos uma população de 50 indivíduos; 200 iterações; uma probabilidade de mutação de 0.4; mutação por inversão da parte inteira e gaussiana da parte real.

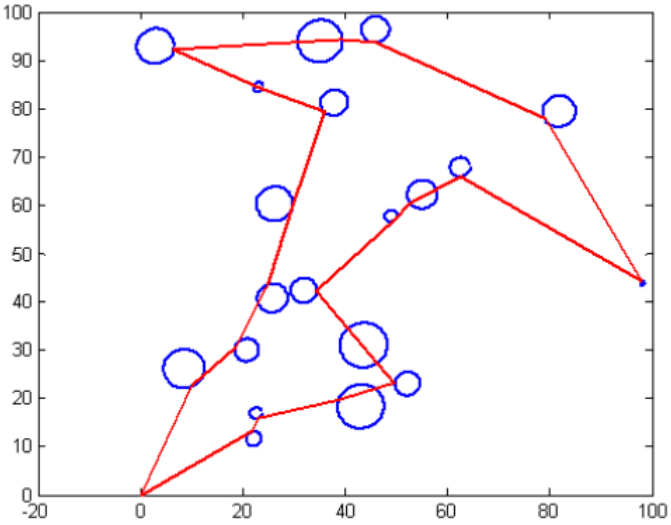

Figura 16: Melhor solução encontrada

V. Considerações finais

Considerando as simplificações realizadas para lidar com o problema, este se mostrou muito parecido com os problemas de otimização de caminhos com vizinhanças – TSPN. Os resultados obtidos foram satisfatórios para cenários de até 100 sensores. Entretanto, percebeu-se que, aumentando-se

o número de variáveis de otimização – número de sensores – o desempenho caiu consideravelmente. Para solucionar estes problemas, sugerimos como trabalhos futuros algumas alterações na implementação do GA. Listamos a seguir as principais alterações necessárias para melhorarmos os resultados do algoritmo.

Uma forma de cruzamento que seja capaz de gerar soluções diversificadas mesmo com indivíduos repetidos é um dos possíveis objetivos futuros, já que observamos a incapacidade do algoritmo de sair de bacias sub-ótimas quando as sequências de permutação dos indivíduos é muito parecida. Da mesma forma, a implementação de uma lógica de adaptação dos parâmetros do algoritmo, como a probabilidade de mutação, poderá levar a resultados consideravelmente melhores.

Outras modificações referem-se ao operador de seleção. Duas possibilidades de modificação deste operador são destacadas. Uma delas é a inclusão de uma probabilidade no torneio. No operador implementado, o melhor indivíduo entre os selecionados sempre é o ganhador. Seria possível deixar uma chance, normalmente entre 0 e 30% de que o pior indivíduo ganhe, evitando uma perda prematura de diversidade.

Outra possível melhoria seria reduzir a pressão seletiva dividindo a população em duas ou três partes, que só competiriam entre eles. Por exemplo, os dois terços dos piores indivíduos poderiam competir somente entre eles. Analogamente, o terço restante dos melhores indivíduos compete isoladamente.

Concluímos que, apesar de várias modificações possam melhorar os resultados obtidos, um algoritmo genético de representação mista é aplicável à resolução desta classe de problemas.

VI. Referências

[1] *On the Optimal Robot Routing Problem in Wireless Sensor Networks*. Bo Yuan, Maria Orlowska, and Shazia Sadiq. IEEE Transactions on Knowledge and Data Engineering. Vol. 19, No. 9, September 2007. p. 1252–1261.

[2] *Alleles, Loci, and the Traveling Salesman Problem*. Goldberg, D.E., and R. Lingle Proceedings of the First International Conference on Genetic Algorithms and Their Application, edited by Grefenstette J., Lawrence Erlbaum Associates, Hillsdale, NJ,1985, p. 154–159.

[3] *Genetic Algorithm Solution of the TSP Avoiding Special Crossover and Mutation*. G. Üçoluk. Intelligent Automation and Soft Computing,3(8), TSI Press. (2002)

[4] http://webpages.iust.ac.ir/yaghini/Courses/AOR_872/Genetic%20Algorithms_03.pdf Acesso em 27/12/12, 16:30

[5] *Notas de Aula de Otimização*. Ramírez, J.A., Batista, L.S., Escola de Engenharia, UFMG, 2012