Algoritmo Memético com Simulated Annealing, 2-Opt e Hill Climbing para otimização de rotas com pontos não exatos

José Tobias

I. INTRODUÇÃO

NESTE relatório será feita a descrição do algoritmo utilizado por mim para a resolução do problema de roteamento de drones passado em aulas anteriores. O problema se resume a achar uma rota para um drone que coletará dados de dispositivos, sendo então necessário o mesmo passar por dentro de certo raio do dispositivo para coletar seus dados.

A resolução do problema utiliza-se de algoritmo genético com intervenções utilizando heurísticas exatas que buscam otimizar suas fazes de evolução, tornando-o assim um algoritmo ”memético”.

O programa completo pode ser encontrado em: http://bit.ly/memetic_drone_reworked

II. DESCRIÇÃO DO ALGORITMO

O algoritmo utiliza um esquema de ilhas, onde diferentes populações se desenvolvem através de seus primeiros, randômicamente gerados, membros. A evolução de cada ilha é feita de forma normal, com os passos padrão de algoritmos genéticos: reprodução, mutação e seleção. O esquema de ilhas me dá a liberdade de fazer troca de membros entre ilhas, inserindo novos ”genes” em determinadas fazer da evolução de cada ilha, bem como acontece/aconteceu na história em vida real.

O algoritmo então possui a ideia de gerações e eras, que são dois loopings, um interno para a evolução da população com seus passos de algoritmos meméticos convencionais, e um externo para migração de membros que têm como objetivo ajudar a evolução da população migrante a fugir de mínimos locais, respectivamente.

Na Figura 1 um exemplo da interface a ser vista no uso do programa durante a execução, onde cada caminho mostrado é o melhor caminho alcançado dentro de cada ilha, e em uma das demonstrações temos o menor de todos os caminhos entre todas as ilhas.

Ao final da execução, uma heurística exata de algoritmos de otimização, 2-opt, e uma técnica de otimização, hill climbing, dão o toque final ao caminho, retirando seus cruzamentos e reduzindo o caminho ao limite do raio. Outra técnica de algoritmos evolutivos, simulated annealing, é usada antes deste passo nos caminhos, a mesma não parece surtir muito efeito quando o algoritmo possui iterações o bastante para evoluir o caminho até um certo ponto, porém, a mesma é sim válida para se encontrar soluções melhores quando seu tempo de execução é limitado para se ter uma resposta. Um exemplo de caminho

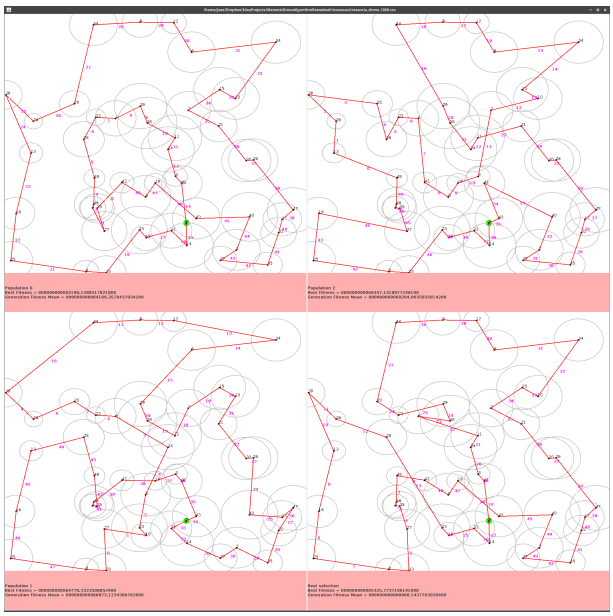

Fig. 1. Interface do programa durante sua execução e interpretação das ilhas.

reduzido ao final da execução do algoritmo pode ser visto na Figura 2. É incrível observar como os caminhos tendem a uma equidade, mesmo trocando-se tão poucos elementos poucas vezes durante a evolução do algoritmo.

III. LIMITAÇÕES

NO momento, o algoritmo não parece ter muitas limitações, tirando aquelas inerentes de sua natureza, quanto a instâncias ou o tamanho das/ mesmas. Obviamente, por sua natureza evolutiva de populações, quanto mais se cresce o tamanho da instância, mais se demora para resolvê-la de maneira satisfatória e perto da otimalidade. Por exemplo, na Figura 3 se tem resultados de uma execução sobre instância de $1,000$ elementos, utilizando-se de 50 eras com 5000 gerações cada em uma população de no máximo 200 componentes, sendo um número considerável de iterações para o tamanho da instância antes de se aplicar heurísticas exatas, as quais o algoritmo teve sua execução completada sem maiores problemas.

Limitações conhecidas de algoritmos baseados em teorias evolutiva com processamento paralelo são tempo e capacidade de processamento de unidade. Um tempo considerável é consumido para realizar até mesmo esta instância com $1,000$

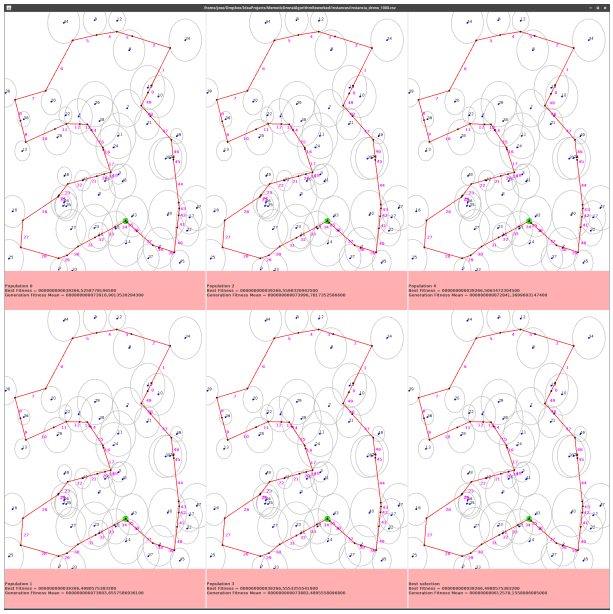

Fig. 2. Exemplo de caminhos reduzidos ao final da execução de uma instância de brinquedo de 30 pontos.

pontos (dispositivos) de maneira a se achar que a solução esteja próxima da ótima, talvez fosse um cenário impossível para minha capacidade de processamento realizar o mesmo com 10.000 pontos.

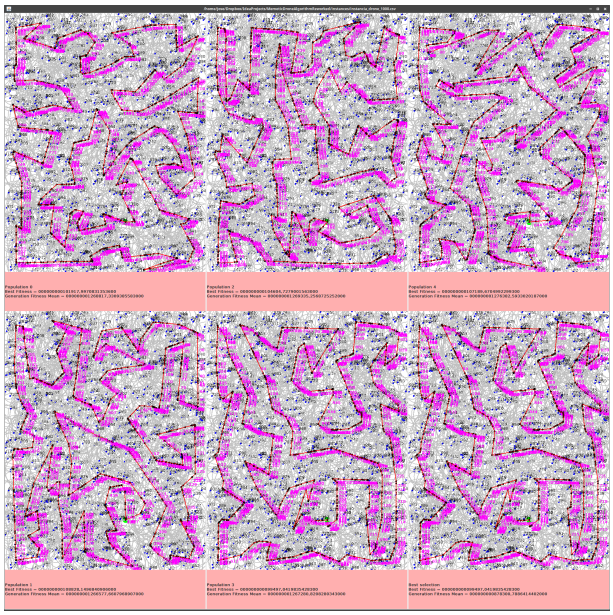

Fig. 3. Resultados para uma instância mais realística de 1,000 pontos.

Conseguindo por fim fazer a integração, como dito nos relatórios anteriores, da sequência de algoritmo genético (evolutivo), simulated annealing e 2-opt ao fim, com passos que introduzem a ”memetização” do algoritmo utilizando-se múltiplas threads, creio que o mesmo agora se encontra apto a achar soluções satisfatórias para problemas muito maiores do que o anterior, que utilizava apenas de uma thread, uma população e um algoritmo genético.

IV. DESCRIÇÃO DETALHADA ESTRUTURAL

DESCREVEREI agora, em termos mais técnicos e a fundo, o funcionamento do algoritmo. Primeiramente descreverei a estrutura do programa, passando então para uma descrição de método principais disponíveis.

A. Estrutura

O algoritmo foi totalmente refeito das suas versões anteriores. Um dos meus maiores problemas era não ter entendido a estrutura do algoritmo, e então, não saber como delegar responsabilidades de maneira correta às classes, acabando com apenas uma classe que fazia tudo sobre meu algoritmo genético e memético, o que levava a grande dificuldade de implementação e manutenção do meu código.

Após a reestruturação, dividi o código em 5 pacotes e 11 classes, sendo uma a classe de aplicação que apenas maneja as classes implementadas do algoritmo. Os pacotes (enumerados) e suas classes (precedidas de letras em ordem alfabética) são:

1) **app:**
   a) **App** - classe que tem como objetivo apenas manejar a aplicação e suas entidades.
2) **graphics:**
   a) **InfoLabel** - tem como função principal manter as informações a serem mostradas para cada população atualizadas. Implementa método de update.
   b) **MainFrame** - contêiner utilizado para armazenas todos os itens de interface gráfica, manejando seu redimensionamento e conteinerização.
   c) **MainPanel** - é o painel de amostra do melhor caminho encontrado para uma determinada ilha. Este painel mostra apenas o melhor caminho e é atualizado após toda e qualquer modificação no mesmo, mostrando então as atividades do algoritmo em tempo real.
3) **memetic:**
   a) **EntitiesManagement** - esta classe tem como função principal fazer o manejamento das populações, bem como dos grupos de acasalamento a serem montados. As fazes de acasalamento, mutação e seleção são todas implementadas nesta classe, bem como o passo de ”memetização”.
   b) **Evolutionary Algorithm** - tem como principal função fazer o manejo da população além de passar atualizações a classes necessárias, gerenciando a classe EntitiesManagement para a chamada de seus métodos implementados.
   c) **MemeticAlgorithm** - Classe que tem como função a implementação de métodos utilizando-se de heurísticas exatas para a resolução aplicada em problemas do tipo do ”caixeiro viajante”, são eles: simulated annealing, 2-opt e redução à borda de raio.

d) **PickMechanism** - Implementa métodos de torneio e roleta para seleção de indivíduos na população ou em grupos dos quais os mesmos devam ser selecionados para alguma operação.

4) **structures:**
   a) **Device** - é apenas a classe de implementação de cada dispositivo, com seu número identificador, coordenada de posição x e y e raio de transmissão associado ao dispositivo. Implementa interface Comparable, bem como um método que cria uma instância com as mesmas propriedades do objeto chamado.
   b) **Tour** - implementa um caminho bem como toda e qualquer operação a ser feita que só diga respeito ao caminho olhado no momento. Operações como troca de dispositivos, medição de distância e criação de caminho através de diversas estruturas são feitas todas nesta classe.
5) **utils:**
   a) **ManipulateData** - classe responsável por leitura e criação de objetos através de arquivo num formato explicitado .csv passado. Dispõe de métodos para leitura completa ou parcial do arquivo qualquer faixa previamente especificada.

B. Os métodos

Descreverei agora os métodos principais de cada classe que afetam apenas o desempenho do algoritmo ou performam parte dos métodos interessados ao algoritmo, deixando assim de fora métodos de interfrace gráfica ou métodos auxiliares que performam ações que levam seus métodos invocadores a certo objetivo. Assim sendo, temos:

1) **App:**
   a) **routePlan** - seleciona o plano de rota das imigrações, assim sendo, de onde para onde serão feitas as imigrações. Feito de maneira totalmente randômica, sem repetições e sem ”imigrações para a própria ilha”.
   b) **generateImigrants** - apenas uma seleção de quais imigrantes serão levados para outro país, feito de maneira totalmente randômica e sem repetição.
2) **EntitiesManagement:**
   a) **memeOver** - função que executa uma das etapas meméticas do algoritmo. Seleciona um indivíduo aleatório, bem como dos points no seu caminho, e executa o método de swap do algoritmo de 2-opt, ou seja, reverte o caminho entre estes dois pontos. Ao final, checa a condição de tamanho de caminho e caso menor, efetua a inserção do novo indivíduo mutado meméticamente na população.
   b) **selection** - função de seleção de indivíduos a serem retirados da população. Utiliza um mecanismo de roleta para a escolha, substituindo tais indivíduos por novos guardados no grupo de acasalamento.
   c) **mutate** - executa a mutação simples sobre determinados elementos da população. Primeiro faz uma seleção randômica e com repetição sobre os indivíduos, então cada indivíduo selecionado tem um dois dispositivos no seu caminho trocados de ordem.
   d) **crossover** - etapa de criação de novos indivíduos a partir de dois outros. A criação é feita através do cruzamento ordenado, e seus indivíduos criados são selecionados através do mecanismo de torneio.
3) **EvolutionaryAlgorithm:**
   a) **firstGen** - cria uma primeira geração da população de maneira totalmente randômica na seleção da ordem dos dispositivos, ou seja, dada uma lista de dispositivos, seleciona aleatoriamente aqueles que montarão a ordem para se percorrer.
   b) **run** - método que permite que o algoritmo seja executado de maneira paralela utilizando a interface Runnable do Java. Tem como principal objetivo chamar os métodos que fazem com que o algoritmo do tipo evolutivo seja executado em sua ordem natural.
4) **MemeticAlgorithm:**
   a) **twoOpt** - método que implementa o algoritmo de 2-opt na melhor solução da ilha.
   b) **annealing** - método que implementa o simulated annealing na melhor solução da ilha.
   c) **reduce** - método que implementa a redução de raio à borda na melhor solução da ilha. É executada de maneira completamente randômica utilizando um limite de operações sem melhoria para precisão.

O algoritmo é implementado de maneira que: a classe EvolutionaryAlgorithm possui uma instância de EntitiedManagement, que por sua vez possui uma instância de PickMechanism. Assim sendo, ao chamar um método apenas dentro de EvolutionaryAlgorithm, o método **run**, todo o algoritmo é executado de maneira encadeada sem intervenções necessárias, o que nos permite paralelismo imediato.

V. ANÁLISE DOS RESULTADOS

VEJAMOS agora alguns gráficos para se analisar o quão eficiente é o algoritmo genético dado o cenário acima, tendo em vista que o algoritmo de simulated annealing é aplicado após a execução do algoritmo memético, o 2-opt é aplicadoe durante e após a execução do algoritmo memético e junto a redução à borda de raio em uma segunda etapa, quero em primeiro ponto analisar apenas o quão eficiente é a saída do meu algoritmo da primeira fase.

Sendo assim, usarei uma instância de 200 dispositivos, com 5 eras e 5000 gerações separadas em 3 ilhas, com população máxima de 100 caminhos. A seleção dos parâmetros é feita para não se alcançar a solução ótima, nos permitindo fazer uma melhor análise gráfica dos dados.

Meus objetivos principais aqui são:

1) Verificar se o algoritmo genético funciona de maneira correta
2) Analisar o quão eficiente é a política(seleção) de imigrantes
3) Analisar o funcionamento dos algoritmos de seleção

Na Figura 4, 5 e 6, vemos os gráficos.

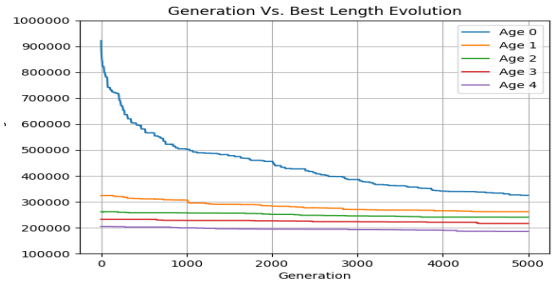

Fig. 4. Evolução na ilha 1.

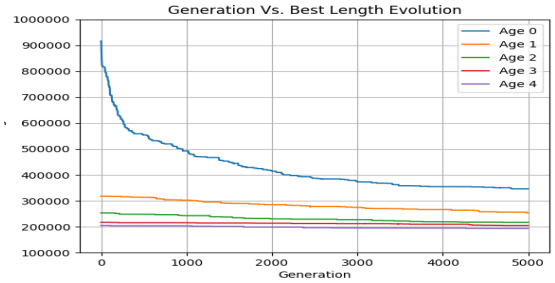

Fig. 5. Evolução na ilha 2.

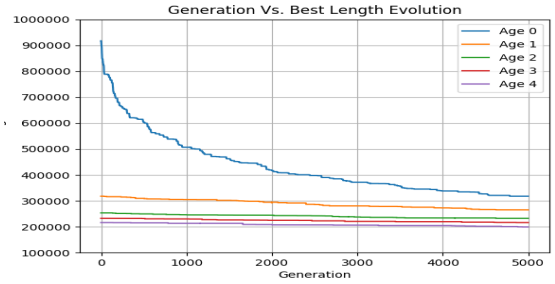

Fig. 6. Evolução na ilha 3.

A presença de uma linha predominantemente decrescente sem grandes variações é um bom sinal de que meu mecanismo seletor de roleta está funcionando de acordo, ou seja, minhas melhores soluções não estão sendo excluídas da população, o que causaria a melhor solução da geração a crescer. Existe sim, uma possibilidade de isso acontecer, porém elas devem de fato ser esporádicas.

A parte memética das gerações parece também ter papel crucial no desenvolvimento das populações, principalmente em fases iniciais ou onde o algoritmo se estagna, ou seja, onde a melhor solução estagna por algumas gerações. Isso pode ser verificado por quedas bruscas nas primeiras iterações do algoritmo, onde muitos cruzamentos são retirados pelo algoritmo 2-opt, e em fases onde há muitas gerações com a mesma solução e abruptamente a solução tem uma melhora grande. Isso pode ser verificado principalmente na primeira era.

A política de seleção dos imigrantes também parece levar uma quantidade significativa, nesse caso 3% da população máxima de 100 caminhos, fazendo com que as soluções continuem decrescendo numa quantidade contínua após a imigração. Esse comportamento já era esperado.

Uma interação que só dá para ser observada durante a execução do algoritmo, é observar que muitas vezes quando uma certa melhoria local é encontrada em uma das ilhas, após as imigrações, esses genes repassados a outras ilhas fazem com que essa melhoria local seja logo alcançada após algumas gerações. É como se fosse um vírus se espalhando por diferentes populações. Essa característica é o principal motivo das migrações ajudarem a evolução das populações.

VI. CONCLUSÃO

TENDO como comparação os meus relatórios e métodos anteriores, se percebe uma visível evolução das soluções e da sua robustez, ficando a lição de que não se deve tentar utilizar métodos exatos para achar soluções otimizadas (otimizadas, e não sempre ótimas) para problemas com complexidade combinatória dessa dimensão. Para certos problemas existem soluções exatas e provadas, como por exemplo problemas de fluxo em grafos ou menor caminho de "A" até "B", porém, para problemas combinatórios como da dimensão deste o mesmo não é verdade.

Fiquei satisfeito com as soluções encontradas e com os mecanismos utilizados, creio que meu algoritmo utilizando as técnicas que o mesmo utiliza se encontra próximo de uma otimalidade de desempenho, com funções otimizadas para cálculo de áreas críticas do algoritmo.

Demais dificuldades não foram encontradas.