**Otimização de topologias de rede por meio de algoritmo genético**

**Quanto ao problema**

A questão escolhida para otimização por meio de algoritmo genético é topologia de rede, onde busca-se estabelecer a melhor conexão possível dentro de um determinado número de nodes(terminais), mantendo assim uma rede 100% conectada, buscando baixo custo, baixa latência e alta largura de banda.

**Quanto à estrutura**

O projeto é dividido em 4 arquivos, sendo eles:

- **ga.py** – Arquivo com código declarativo, contendo as funções necessárias relativas ao algoritmo genético.

- **vars.py** – Arquivo com código declarativo, contendo as variáveis necessárias para a configuração e personalização do algoritmo genético. As variáveis podem ser personalizadas como desejável.

- **pg.py** – Arquivo com código declarativo, contendo as funções e variáveis necessárias para a ilustração do algoritmo com o pygame.

- **index.py** – Arquivo com o código executável do algoritmo, efetuando a unificação dos demais.

**Quanto à codificação**

Os cromossomos são compostos pelos links possíveis entre todos os nodes da rede, de forma binária, indicando a atividade do link (on/off), assim formando uma matriz de adjacência. A quantidade de links é definida pela quantidade de nodes constantes na rede, pela seguinte fórmula: (n \* (n – 1)) / 2, onde n = número de nodes da rede.

**Quanto à aptidão**

A função de fitness(aptidão) considera uma série de fatores:

1. **Conectividade**: Todos os nodes devem estar conectados entre si, formando assim uma rede 100% conectada. Caso o cromossomo não possua conectividade sofrerá uma alta penalidade de aptidão, o tornando impraticável.

2. **Largura de banda**: Cada link do cromossomo está atrelado a um valor de largura de banda, caso o mesmo esteja ativo e não atinja a largura mínima estabelecida a aptidão será penalizada.

3. **Latência**: Cada link possui uma latência associada, e a latência total do cromossomo(links ativos) será somada à função de aptidão, ou seja, quanto menor a latência, melhor será a aptidão.

4. **Custo**: Cada link possui um custo inerente, e uma vez ativo, seu custo será somado à aptidão, assim ocorrendo em penalidade para links custosos.

Fórmula da aptidão:

Penalidade de conectividade + Penalidade de largura de banda + Latência total + Custo total

Quanto mais baixa a aptidão, maior a sua eficiência.

**Quanto à população inicial**

O algoritmo dispõe de duas formas de inicialização da população, sendo elas aleatória e hot start. Com base em testes realizados, a inicialização aleatória se mostrou ineficiente, por iniciar cromossomos sem conectividade, acarretando assim em aptidões altamente penalizadas.

Já o hot start inicia um indivíduo com a técnica MST(Minimun Spanning Tree), que gera um indivíduo conectado e com um custo mínimo associado, gerando assim um indivíduo com aptidão viável. Porém este indivíduo desconsidera os demais fatores de aptidão, como a latência e a largura de banda, deixando grande espaço para melhoria. Baseado neste indivíduo inicial, a população inicial é completada com indivíduos contendo de 1 a 3 mutações aleatórias sobre o MST inicial.

**Quanto à seleção e ao cruzamento**

Para a otimização de topologias de rede, o elitismo se mostrou mais eficiente que a alta exploração, pelo motivo de que a alta exploração favorece o aparecimento de indivíduos sem conectividade, acarretando em aptidões altamente penalizadas com considerável frequência, sendo assim se mostra necessária a alta valorização de invíduos conectados. Sempre a cada geração preserva-se o indivíduo mais apto.

A seleção é processada da seguinte maneira: Primeiramente se ordena a população por ordem decrescente de aptidão e após separa-se os primeiros 30% da população para execução de seleção aleatória por torneio. Após a seleção de dois pais executa-se dois cruzamentos para a geração de dois novos filhos, assim sucessivamente até a completude da nova população.

Quanto ao cruzamento, a abordagem clássica para cruzamento binário por trechos de cada pai se mostrou altamente problemática, pois na maioria dos casos ela gera filhos sem conectividade, causando assim na maioria dos casos uma estagnação no progresso populacional. Por meio de testes buscou-se conciliar este cruzamento com a permanência da conectividade, mas ainda assim a solução se mostrou inviável, por causar estagnação no progresso do algoritmo.

**Edge Recombionation Crossover(ERO)** foi a solução encontrada para que se obtivesse progresso entre as gerações. Esta abordagem de cruzamento recombina os links dos cromossomos pais aleatoriamente, porém mantendo sua estrutura de conectividade(adjacência), assim gerando filhos viáveis.

**Quanto à mutação**

Baseado em testes realizados com o algoritmo, a mutação se mostrou altamente prejudicial para o progresso do algoritmo, mesmo em baixas porcentagens, pois com frequência pode acarretar na perda de conectividade do indivíduo, ocasionando assim em aumento substancial da penalidade na aptidão.

**Quanto à conclusão e aos resultados**

O algoritmo se mostrou eficiente em buscar redes otimizadas. Claro que sempre pode-se adicionar complexidade, pois o algoritmo considera uma rede com terminais comuns interconectados por rede cabeada e exclui a possibilidade de conexão wireless.

A demanda por hardware e a demora para se encontrar uma solução ótima cresce considerávelmente com o aumento dos nodes de rede.

Resultados, considerando um PC residencial comum e uma população de 100 indivíduos:

**5** nodes: Acha-se um resultado ótimo entre 2 e 5 gerações com até 2 segundos de execução.

**30** nodes: Acha-se excelentes resultados dentro de 300 gerações, com processamento rápido.

**70** nodes: O processamento é de cerca de 2 à 3 gerações por segundo, atingindo progressos frequentes até cerca de 500 à 600 gerações. Aqui já se nota um consumo considerável de hardware.

**90** nodes: Aqui o processamento já é bem mais custoso, cerca de 1.2 gerações por segundo. A otimização também é mais demorada, chegando a levar 50 gerações após a geração de número 200.

**100** nodes: Aqui o processamento é de mais de 1 segundo por geração, e o ritmo de localização de melhorias segue semelhante à 90 nodes e, mesmo num PC comum, o algoritmo segue viável e eficiente.

Com o uso do método de Edge Recombination Crossover para o cruzamento, o algoritmo se mostrou eficiente, alcançando soluções ótimas com presteza. O uso da mutação mostrou-se prejudicial. Para uma rede de 50 nodes, por exemplo, uma taxa de mutação acima de 1%(0.01) tornou o algoritmo altamente ineficiente.