

Resolução do problema de localização de regeneradores através de algoritmos de inteligência artificial

Pedro Ferreira

Rodrigo Pessoa

Junho 2018

Trabalho realizado no âmbito da unidade curricular de Projeto Informático

Trabalho realizado sob orientação das

Professoras Anabela Moreira Bernardino e Eugénia Moreira Bernardino.

Agradecimentos

Gostaríamos de agradecer às Professoras Anabela Moreira Bernardino e Eugénia Moreira Bernardino pelo acompanhamento no desenvolvimento do projeto. Foram verdadeiramente prestáveis e permitiram que a conclusão do projeto fosse um sucesso.

Gostaríamos também de agradecer a todos os Professores do Instituto Politécnico de Leiria pela formação de qualidade que nos prestaram ao longo do curso.

Resumo

No contexto do Projeto Informático de 2017/18 desenvolvemos uma aplicação em C++ que permite resolver o problema de localização de regeneradores.

Nas redes óticas, a força de um sinal ótico deteriora-se à medida que fica mais longe da fonte, devido a deficiências de transmissão na fibra (atenuação, dispersão, conversação). Ou seja, a distância a que um sinal ótico pode ser enviado sem perder ou deteriorar a informação é limitada. É por isso necessário regenerar os sinais periodicamente utilizando regeneradores. Dada uma rede ótica, o problema de localização de regeneradores (PLR – Problema da Localização dos Regeneradores) procura que o subconjunto de regeneradores seja instalado a um custo mínimo, de modo a que cada par de nós possa comunicar entre si.

A aplicação desenvolvida é bastante dinâmica, tendo integradas diversas funcionalidades. Permite que cada problema seja resolvido individualmente, ilustrando graficamente a evolução do *fitness* do melhor indivíduo (melhor solução encontrada até ao momento) ao longo de um conjunto específico de iterações. Oferece também a possibilidade de resolver um conjunto especificado de problemas, utilizando sempre os mesmos parâmetros inseridos pelo utilizador, aplicando ainda 50 *seeds* diferentes. Por fim, esta aplicação permite também testar diferentes combinações de parâmetros de um só algoritmo, para um conjunto especificado de problemas. Os dados resultantes dos nossos testes são guardados automaticamente em formato texto em ficheiros csv.

Acrescentamos que os resultados foram analisados com auxílio do programa “RapidMiner”, de forma a conseguir obter as médias de *fitness* (que determinam a qualidade das soluções), regeneradores, nós desconectados e tempo decorrido, para os diferentes *seeds* e instâncias dos problemas.

Abstract

For our final project (2017/2018) we developed an application, built in C++, capable of solving the Regenerator Location Problem.

In optical networks, a signal strength deteriorates as it gets further away from its source. This happens due to deficiencies in the fiber (attenuation, dispersion, conversion). Therefore, we can say that the distance a signal can travel without losing or corrupting information is limited. It is necessary to regenerate the signals periodically using regenerators. Given an optical network, the regenerator location problem tries to install a subset of regenerators with the minimum possible cost, in a way that each pair of nodes can communicate with each other.

The developed program is quite dynamic and integrates various functionalities. It allows for each problem to be solved individually, illustrating graphicly the evolution of the best individual’s fitness along the various generations. It also offers the possibility to solve a specified group of problems, using the same parameters declared by the user for each one, and applying 50 different seeds. The program is also cable of testing different combinations of parameters for a chosen algorithm. The resulting data from our tests is automatically stored as text in a csv file.

The software “RapidMiner” was used to help in the analysis of our data by obtaining the fitness averages, regenerators, disconnected nodes, time for the different seeds and problem instances.

Índice

[Agradecimentos v](#_Toc517701450)

[Resumo vii](#_Toc517701451)

[Abstract ix](#_Toc517701452)

[Índice xi](#_Toc517701453)

[Lista de Figuras xiii](#_Toc517701454)

[Lista de Tabelas xv](#_Toc517701455)

[Lista de Acrónimos xvii](#_Toc517701456)

[Capítulo 1 - Introdução 1](#_Toc517701457)

[1.1 Problema PLR 1](#_Toc517701458)

[1.2 Motivação 2](#_Toc517701539)

[1.3 Estrutura do Relatório 2](#_Toc517701540)

[1.4 Síntese 3](#_Toc517701541)

[Capítulo 2 - Enquadramento 5](#_Toc517701542)

[2.1 Descrição do Problema 5](#_Toc517701543)

[2.2 Problemas de Otimização](#_Toc517701545) *[NP-Complete](#_Toc517701545)* [10](#_Toc517701545)

[2.3 Algoritmos evolutivos de inteligência artificial (algoritmos baseados em populações) 11](#_Toc517701546)

[2.4 Algoritmos de swarm intelligence 12](#_Toc517701547)

[2.5 Síntese 12](#_Toc517701548)

[Capítulo 3 – Linguagens e Tecnologias 13](#_Toc517701549)

[3.1 C++/CLI 13](#_Toc517701550)

[3.2 Qt 14](#_Toc517701551)

[3.3 Git e Github 16](#_Toc517701552)

[3.4 RapidMiner 16](#_Toc517701553)

[3.5 Síntese 16](#_Toc517701554)

[Capítulo 4 - Aplicação 17](#_Toc517701555)

[4.1 Análise de Requisitos 17](#_Toc517701556)

[4.2 Metodologia 17](#_Toc517701561)

[4.3 Algoritmos Evolutivos 19](#_Toc517701562)

[4.4 Estrutura do Programa 27](#_Toc517701563)

[4.5 Síntese 35](#_Toc517701564)

[Capítulo 5 - Resultados 37](#_Toc517701565)

[5.1 Obtenção de dados 37](#_Toc517701566)

[5.2 Tratamento dos dados obtidos 38](#_Toc517701567)

[5.3 Resultados obtidos 39](#_Toc517701568)

[5.4 Comparação dos algoritmos implementados 52](#_Toc517701569)

[5.5 Síntese 53](#_Toc517701571)

[Capítulo 6 - Conclusão 55](#_Toc517701572)

[6.1 Considerações finais 55](#_Toc517701573)

[6.2 Trabalho futuro 56](#_Toc517701575)

[6.3 Síntese 57](#_Toc517701576)

[Bibliografia 59](#_Toc517701577)

[Anexo 61](#_Toc517701578)

Lista de Figuras

[Figura 1 - Exemplo de problema com 7 nós 6](#_Toc517701411)

[Figura 2 - Ficheiro de texto do problema exemplo 7](#_Toc517701412)

[Figura 3 - Vetor de pesos do problema exemplo 8](#_Toc517701413)

[Figura 4 - Exemplo de uma solução inviável 8](#_Toc517701414)

[Figura 5 - Representação da solução inviável 8](#_Toc517701415)

[Figura 6 - Cálculo do fitness da solução inviável 9](#_Toc517701416)

[Figura 7 - Exemplo de uma solução não otimizada 9](#_Toc517701417)

[Figura 8 - Representação da solução não otimizada 9](#_Toc517701418)

[Figura 9 - Cálculo do fitness da solução não otimizada 9](#_Toc517701419)

[Figura 10 - Exemplo de uma solução otimizada 10](#_Toc517701420)

[Figura 11 - Representação da solução otimizada 10](#_Toc517701421)

[Figura 12 - Cálculo do fitness da solução otimizada 10](#_Toc517701422)

[Figura 13 - Programa em C++/CLI 14](#_Toc517701423)

[Figura 14 - Programa em Qt 16](#_Toc517701424)

[Figura 15 - Desenvolvimento Iterativo e Incremental 18](#_Toc517701425)

[Figura 20 - Custom Dialog 30](#_Toc517701426)

[Figura 21 - Ant Colony Test Dialog 31](#_Toc517701427)

[Figura 22 - Main Window 33](#_Toc517701428)

Lista de Tabelas

[Tabela 1 – Primeiro conjunto de semanas de desenvolvimento 18](#_Toc517701581)

[Tabela 2 - Segundo conjunto de semanas de desenvolvimento 19](#_Toc517701582)

[Tabela 3 - Cinco melhores resultados do teste do Custom Algorithm 39](#_Toc517701583)

[Tabela 4 - Cinco melhores resultados do teste do Algoritmo Genético 40](#_Toc517701584)

[Tabela 5 - Cinco melhores resultados do teste do Bee Colony Optimization 41](#_Toc517701585)

[Tabela 6 - Cinco melhores resultados do teste do Ant Colony Optimization 41](#_Toc517701586)

[Tabela 7 – Resultados dos problemas random com o Custom Algorithm 42](#_Toc517701587)

[Tabela 8 – Resultados dos problemas weighted com o Custom Algorithm 43](#_Toc517701588)

[Tabela 9 – Resultados dos problemas random com o Algoritmo Genético 44](#_Toc517701589)

[Tabela 10 - Resultados dos problemas weighted com o Algoritmo Genético 46](#_Toc517701590)

[Tabela 11 - Resultados dos problemas random com Bee Colony Optimization 47](#_Toc517701591)

[Tabela 12 – Resultados dos problemas weighted com Bee Colony Optimization 48](#_Toc517701592)

[Tabela 13 - Resultado dos problemas random com Ant Colony Optimization 49](#_Toc517701593)

[Tabela 14 - Resultados dos problemas weighted com Ant Colony Optimization 50](#_Toc517701594)

[Tabela 15 - Resultados dos diferentes algoritmos para os diferentes tamanhos de problemas 52](#_Toc517701595)

[Tabela 16 - Médias dos resultados dos diferentes algoritmos 52](#_Toc517701596)

[Tabela 17 - Valor de tempo vezes fitness para os diferentes algoritmos 52](#_Toc517701597)

[Tabela 18 - Resultados dos testes do Custom Algorithm 61](#_Toc517701598)

[Tabela 19 - Resultados dos testes do Algoritmo Genético 63](#_Toc517701599)

[Tabela - Resultados dos testes do Bee Colony Optimization 67](#_Toc517701600)

[Tabela 21 - Resultados dos testes do Ant Colony Optimization 69](#_Toc517701601)

Lista de Acrónimos

ACO Ant Colony Optimization

AG Algoritmo Genético

BCO Bee Colony Optimization

CA Custom Algorithm

CLI Common Language Infrastructure

MOC Meta-Object Compiler

PLR Problema de Localização de Regeneradores

SI Swarm Intelligence

1. - Introdução

No âmbito da unidade curricular de Projeto Informático, do curso de Licenciatura em Engenharia Informática, da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria, foi elaborado o presente relatório com base no projeto: Resolução do problema de localização de regeneradores através de algoritmos de inteligência artificial.

Este capítulo está estruturado da seguinte forma: a secção 1.1 apresenta uma breve descrição do problema resolvido neste projeto; a secção 1.2 apresenta a motivação para o desenvolvimento deste projeto; a secção 1.3 apresenta a estrutura deste relatório, e a secção 1.4 sumariza os conceitos apresentados neste capítulo.

* 1. Problema PLR

A Inteligência Artificial tem vindo a ganhar imensa popularidade, devido às suas capacidades de permitir que máquinas aprendam e se ajustem a diferentes problemas . Tem já uma grande presença no nosso dia a dia, sendo aplicada muito frequentemente em jogos de computador, automóveis, assistentes digitais (como a *Amazon Echo*), *robots*, etc. No núcleo de toda a inteligência artificial estão os seus algoritmos, pois são estes que treinam a inteligência para que esta aprenda. No caso do PLR, o requisito mínimo era a implementação de dois algoritmos de inteligência artificial para resolver o problema, tendo o nosso grupo implementado quatro.

O PLR é um problema real, pois de facto existe um problema nas redes óticas. Quanto mais distante estiver o sinal ótico da sua fonte, mais sujeito este está a perder qualidade. Isto acontece devido a falhas na fibra em si. De modo a resolver este problema, existem máquinas chamadas regeneradores, cujo propósito é restaurar a qualidade do sinal. Infelizmente, estes equipamentos têm um custo bastante elevado.

O nosso objetivo é resolver o PLR, obtendo soluções com o menor custo possível, no menor tempo possível. Todo este processo foi feito recorrendo a diferentes algoritmos de Inteligência Artificial. Os algoritmos que escolhemos utilizar foram

o *Custom Algorithm* (CA), o Algoritmo Genético (AG), o *Ant Colony Optimization* (ACO) e o *Bee Colony Optimization* (BCO).

Foi feita uma comparação do desempenho dos algoritmos em termos de média de qualidade das soluções e de tempo para chegar à melhor solução encontrada.







* 1. Motivação

Com a evolução das tecnologias de comunicação, existe cada vez mais a necessidade de um serviço rápido, seguro e sem falhas. Como resposta a esta necessidade surgem as redes de fibra ótica.

Como mencionado previamente, o PLR é um problema do mundo real. Decidimos encará-lo pela sua relevância e pelo desafio de encontrar soluções, aplicando algoritmos baseados na natureza.

* 1. Estrutura do Relatório

Este relatório está estruturado da seguinte forma:

* Capítulo 1 - Introdução

Neste capítulo é feita a apresentação do nosso problema e trabalho, para que o leitor seja contextualizado e adquira uma ideia geral dos temas que serão tratados ao longo do relatório. Resumimos todo o projeto e explicamos as nossas motivações para este trabalho.

* Capítulo 2 – Enquadramento

No capítulo 2 analisamos pormenorizadamente o problema em estudo, explicando-o detalhadamente e fornecendo vários conceitos importantes para a compreensão do mesmo.

* Capítulo 3 – Linguagens e tecnologias

Neste capítulo abordamos a linguagem de programação aplicada e as tecnologias que tiveram mais impacto neste projeto, como o QT e o GitHub. Expomos também as várias dificuldades com que nos deparámos, nomeadamente a adaptação a uma linguagem de programação com que nunca tínhamos lidado.

* Capítulo 4 – Aplicação

Este é o capítulo mais extenso, onde entramos em detalhe relativamente à aplicação. Começamos por pormenorizar os requisitos impostos para este projeto, descrevendo de seguida a metodologia de trabalho aplicada. Abordamos igualmente a estrutura do nosso programa e, por fim, os algoritmos aplicados.

* Capítulo 5 – Resultados

Neste capítulo discutimos os resultados obtidos, mais concretamente o modo como foram alcançados e como foram tratados.

* Capítulo 6 - Conclusão

No capítulo 6 procedemos ao estudo comparativo, determinando a nossa melhor solução para o problema.

* 1. Síntese

Neste capítulo são identificados os pontos mais importantes deste relatório, bem como de todo o projeto em si. São brevemente explicados o problema e o modo como este foi resolvido. Deixamos clara a nossa motivação para trabalhar neste projeto e definimos a estrutura de todo o relatório.

1. - Enquadramento

Este capítulo está estruturado da seguinte forma: a secção 2.1 apresenta a descrição do problema resolvido neste projeto; a secção 2.2 apresenta o conceito de problemas otimização do tipo *NP-Complete*; a secção 2.3 apresenta os tipos de algoritmos que permitem a solução deste tipo de problema; a secção 2.4 apresenta o conceito de algoritmos de *swarm* *intelligence*, e a secção 2.5 sumariza os conceitos apresentados neste capítulo.

* 1. Descrição do Problema

O problema pretende garantir que numa rede de fibra ótica os sinais são regenerados de forma a manter a qualidade do sinal, que se deteriora proporcionalmente à distância que percorre na rede. Este problema pertence à categoria de problemas *NP-Complete* (ver secção 2.2) [xx].

Os problemas utilizados (instâncias) no nosso projeto estão presentes no *website* do centro de computadores da universidade de Viena, que também abordou previamente o PLR. Estes problemas são representações simplificadas de redes de fibra ótica, contendo um conjunto de nós interligados. Para encontrar a solução deste problema é necessário conseguir descobrir uma disposição de regeneradores de sinais num conjunto de nós interligados, de forma a que todos os nós contenham ou estejam ligados a pelo menos um regenerador. Este processo terá de ser feito utilizando o menor número de regeneradores possível. No caso de problemas em que a colocação de regeneradores num nó específico tem custos diferentes, o objetivo é ter o menor custo possível.

A aplicação ao mundo real da nossa forma de resolver o problema necessitaria de uma ferramenta capaz de converter uma rede de fibra ótica real numa rede simplificada como as que foram utilizadas.

* + 1. Estrutura do problema

Os problemas utilizados estão inicialmente sob o formato de ficheiros de texto, sendo estes obtidos a partir do *site* do “Centro de Computadores da Universidade de Viena” [1]. A principal informação contida nestes ficheiros é uma matriz que representa as ligações entre os diferentes nós que compõem o problema. Para além da referida matriz, contêm também o número de nós e o número de ligações.

Alguns dos problemas englobam também um vetor com pesos, referente ao custo da colocação de um regenerador em cada nó específico.

* + 1. Exemplo simplificado de um problema

De forma a ilustrar o problema, apresentamos um exemplo dum problema com 7 nós na .

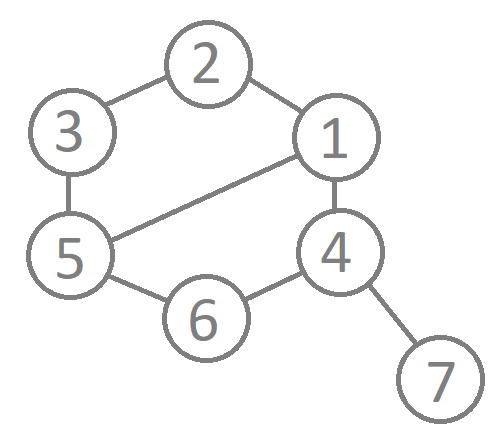


Figura 1 - Exemplo de problema com 7 nós

A ilustra o ficheiro de texto correspondente a este problema.

nNodes = 7

nPairs = 8

Weight

3

2

3

4

2

3

4

Cost

1 1 0 1 1 0 0

1 1 1 0 0 0 0

0 1 1 0 1 0 0

1 0 0 1 0 1 1

0 0 0 1 1 1 0

0 0 0 1 0 0 1

NDC Pairs

<1 2>

<1 4>

<1 5>

<2 3>

<3 5>

<4 6>

<4 7>

<5 6>

Figura 2 - Ficheiro de texto do problema exemplo

O vetor de pesos do problema é o seguinte:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nó | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Peso | 3 | 2 | 3 | 4 | 2 | 3 | 4 |

Figura 3 - Vetor de pesos do problema exemplo

Todos os nós devem estar conectados a um regenerador. O regenerador pode estar no próprio nó ou num dos nós adjacentes, isto é, num dos nós ligados a esse nó.

Um exemplo de uma solução inviável pode ser encontrado na .

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nó | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Regenerador | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

Figura 4 - Exemplo de uma solução inviável

A ilustra a representação da solução obtida.

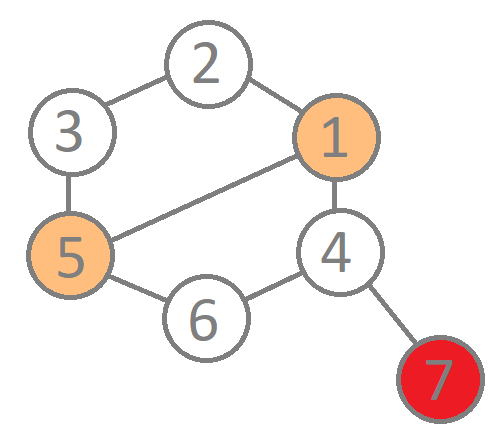


Figura 5 - Representação da solução inviável

Como se pode verificar no exemplo anterior o nó 7 não está ligado a nenhum regenerador, logo esta solução é inviável.

Para determinar a qualidade da solução é calculado um valor de fitness utilizando a seguinte fórmula:

*fitness = ND \* 500 + ∑RU Wi \* 100*

*ND* corresponde ao número de nós desligados, *RU* ao número de regeneradores utilizados e *Wi* corresponde ao peso do regenerador *i*.

De acordo com a solução em cima o *fitness* da solução é calculado da seguinte forma:

|  |  |
| --- | --- |
| Nós desligados (*ND*): 1  Regeneradores utilizados (*RU*): 2  W1: 3  W5: 2 |  |

Figura 6 - Cálculo do fitness da solução inviável

Um exemplo de uma solução não otimizada pode ser encontrado na .

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nó | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Regenerador | 0 | 1 | 0 | 0 | 1 | 0 | 1 |

Figura 7 - Exemplo de uma solução não otimizada

A ilustra a representação da solução obtida.

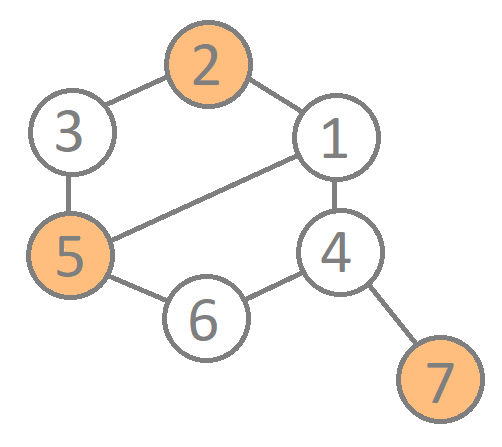


Figura 8 - Representação da solução não otimizada

Cálculo do *fitness* da solução:

|  |  |
| --- | --- |
| Nós desligados (*ND*): 0  Regeneradores utilizados (*RU*): 2  W2: 2  W5: 2  W7: 4 |  |

Figura 9 - Cálculo do fitness da solução não otimizada

Esta solução é válida, pois todos os nós estão ligados a pelo menos um regenerador, contudo pode ser obtida uma solução mais otimizada.

Um exemplo de uma solução otimizada pode ser encontrado na .

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Nó | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Regenerador | 0 | 0 | 1 | 1 | 0 | 0 | 0 |

Figura 10 - Exemplo de uma solução otimizada

A ilustra a representação da solução obtida.

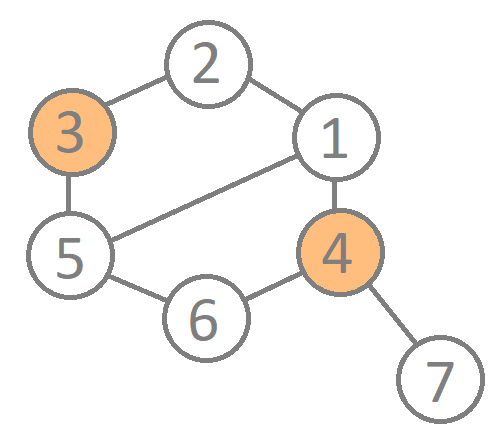


Figura 11 - Representação da solução otimizada

Cálculo do *fitness* da solução:

|  |  |
| --- | --- |
| Nós desligados (*ND*): 0  Regeneradores utilizados (*RU*): 2  W3: 3  W4: 4 |  |

Figura 12 - Cálculo do fitness da solução otimizada

Esta solução é válida, pois todos os nós estão ligados a pelo menos um regenerador.

* 1. Problemas de Otimização *NP-Complete*

O problema estudado é classificado como *NP-Complete* (***N****on-deterministic* ***P****olynomial time*). Um problema é classificado *NP-Complete* se for considerado *NP-Hard*. Um problema deste tipo pode ser considerado pelo menos tão difícil quanto o problema mais difícil do tipo *NP-Hard*. Por outras palavras, qualquer algoritmo utilizado para resolver um problema do tipo *NP-Hard* em tempo polinomial pode ser posteriormente traduzido para resolver qualquer problema também *NP-Hard*. Os problemas do tipo *NP-Complete* podem ser: problemas de decisão, problemas de pesquisa ou problemas de otimização.

Para resolver este tipo de problemas podem ser utilizados algoritmos de aproximação que rapidamente encontram uma solução não necessariamente ótima, contudo dentro de um certo intervalo de erro. Em alguns casos, encontrar uma boa aproximação é o suficiente para resolver o problema, porém nem todos os problemas *NP-Complete* têm bons algoritmos de aproximação. Em muitos casos são utilizadas meta-heurísticas como os algoritmos evolutivos e os algoritmos de *swarm intelligence* que trabalham razoavelmente bem em muitos casos encontrando boas soluções num tempo razoável.

* 1. Algoritmos evolutivos de inteligência artificial (algoritmos baseados em populações)

Os Algoritmos Evolutivos têm como objetivo trabalhar com o melhor de uma solução e melhorá-la. Esta ideia é baseada nos princípios da teoria da evolução biológica, isto é, pode-se simular o processo evolutivo da natureza de modo a encontrar soluções ótimas para os nossos problemas [xx]. Deste princípio, surgiram vários algoritmos, como por exemplo o Algoritmo Genético (AG).

Por norma, todos os algoritmos evolutivos começam com uma população inicial. Desta população, cada elemento (ou indivíduo) representa uma solução para o problema. Podemos deduzir que nem todos indivíduos são ótimos, sendo necessária uma avaliação. Desta avaliação irá resultar o *fitness***,** que representa a qualidade de um indivíduo como solução para o nosso problema. O próximo passo será comparar os *fitnesses*, identificando os melhores indivíduos para que sejam preservados e passem para a próxima geração, da qual irão ser membros e na qual serão utilizados como referência para gerar novos indivíduos. Através deste processo em que se tira proveito das qualidades dos melhores indivíduos para gerar soluções melhores ao longo das várias gerações irá resultar uma solução final que em princípio será a ótima para o problema. Todo este processo de procura pela melhor solução repete‑se várias vezes durante um certo número de iterações (chamadas gerações) até que uma condição seja alcançada, como, por exemplo, se “um objetivo foi cumprido” ou se “foi atingido o número máximo de gerações”.

* 1. Algoritmos de swarm intelligence

Baseado em *Jason Brownlee, Clever Algorithms: Nature Inspired Programming Recipes* [4], podemos concluir que os algoritmos de SI têm por base os comportamentos coletivos de vários agentes presentes num dado ambiente. Isto é, existindo um grupo de indivíduos menos inteligentes capazes de agir por si num certo ambiente, das suas interações cooperativas irá surgir esta inteligência coletiva. A inspiração para algoritmos deste tipo surge de sistemas presentes na natureza, como é o caso do *Ant Colony Optimization* (ACO), baseado na forma como as formigas usam feromonas para se influenciarem na procura de comida. Algoritmos deste tipo são considerados bastante adaptáveis e são especialmente efetivos em problemas de procura e otimização. No nosso projeto utilizamos dois algoritmos deste tipo, sendo estes o ACO e o *Bee Colony Optimization* (BCO).

* 1. Síntese

Com este capítulo explicamos detalhadamente o principal tema deste projeto, o problema da localização de regeneradores, bem como alguns conceitos pertinentes para a compreensão deste, como é o caso dos problemas *NP-Complete*, dos algoritmos evolutivos e dos algoritmos de *swam* *intelligence*.

1. – Linguagens e Tecnologias

Este capítulo está estruturado da seguinte forma: a secção 3.1 apresenta a linguagem de programação utilizada no desenvolvimento deste projeto; a secção 3.2 apresenta a *framework* utilizada no desenvolvimento deste projeto; a secção 3.3 apresenta a ferramenta Git e a plataforma GitHub, utilizadas para o controlo de versões no desenvolvimento deste projeto; a secção 3.4 apresenta a ferramenta de *data mining* RapidMiner, e a secção 3.5 sumariza os conceitos apresentados neste capítulo.

* 1. C++/CLI

Começámos por discutir a possibilidade de desenvolver o programa em C++, devido à rapidez de execução, aspeto que se revela importante na solução deste tipo de problemas. Já estávamos familiarizados com a linguagem C; contudo, foi necessário adaptarmo-nos à linguagem C++.

C++/CLI ou C++ modificado para *Common Language Infrastructure* é a integração de C++ com *Windows Forms*. No entanto, esta implementação é quase como uma outra linguagem de programação, diferente de C++. Grande parte das instruções não podem ser executadas, havendo grandes mudanças na sintaxe e diferenças nos objetos.

Inicialmente, o programa foi implementado em C++/CLI; contudo, devido às limitações que acompanhavam esta plataforma (*software* Visual Studio 2017), acabámos por optar pela *framework* de C++ “Qt”.

* + 1. A implementação com C++/CLI

Com C++/CLI, foi implementada uma janela com os parâmetros do algoritmo e 2 botões, um botão para ler um problema e outro para resolver o problema.

A leitura do problema é feita através da leitura de uma matriz de ligações entre nós, presente nos diferentes ficheiros de texto. Alguns ficheiros têm também pesos associados a cada um dos nós, em que cada peso representa o custo de colocar um regenerador nesse nó.

Na resolução do problema pode ser aplicado, por exemplo, o algoritmo que denominamos “*Custom Algorithm*” (cujos detalhes são explicados adiante, no capítulo 4). Para tal, uma nova *thread* é criada. Esta vai aplicando o algoritmo e a população vai evoluindo. A cada geração, a *thread* invoca um método do *form* que o atualiza, para que se mostrem os dados do melhor indivíduo da população atual, bem como um gráfico que retrata a evolução do *fitness* ao longo das várias iterações do algoritmo (ver Figura 13).



Figura 13 - Programa em C++/CLI

* 1. Qt

Qt é uma ferramenta de desenvolvimento para *desktop* compatível com diferentes plataformas. Não é só por si uma linguagem de programação, mas é uma *framework* escrita em C++ com características adicionais, como *signals and slots*, e o funcionamento do seu MOC permite converter todo o código escrito de forma a ser compilado por qualquer compilador de C++ [2].

* + 1. A implementação com a *framework* Qt

O programa desenvolvido em Qt é a versão final do nosso projeto.

Esta versão contém uma janela principal com a opção de selecionar um dos quatro algoritmos implementados, e ainda uma ferramenta extra para ordenar os resultados guardados nos ficheiros *csv*.

Cada algoritmo tem a sua janela própria, em que temos os parâmetros requeridos pelo algoritmo em questão, um botão para resolver um problema individualmente, um botão para resolver uma série de problemas e um botão para testar combinações de parâmetros.

Na solução de um problema individual, é selecionado o ficheiro do problema; este é integrado na classe problema e, de seguida, é criada uma *thread* específica ao algoritmo selecionado, que a cada geração envia um sinal à *thread* da interface visual, com a informação do melhor indivíduo da geração atual, de forma a atualizar os dados da janela e do gráfico da evolução do *fitness* ao longo das gerações.

Na solução de um conjunto de problemas, é selecionada uma pasta com diferentes problemas e criado um ficheiro *csv* onde são guardados os resultados da execução do algoritmo para cada um dos ficheiros e para cada *seed* diferente. São criadas várias *threads*, e cada uma vai resolvendo um problema diferente das restantes *threads* (ver Figura 14). Para tal, é lido o problema, criada a população e aplicado o algoritmo. Depois de obtida a solução, é enviado um sinal à *thread* original para escrever os resultados no ficheiro csv.

No teste de diferentes combinações de parâmetros, é selecionada uma pasta com os problemas a testar, introduzida uma combinação de valores a testar para cada parâmetro e é criado um ficheiro csv para os resultados da execução. São ainda criadas várias *threads*, resolvendo cada uma delas um problema diferente e aplicando ao mesmo problema as diferentes combinações de parâmetros. Sempre que a solução para um conjunto de parâmetros termina, é enviado um sinal à *thread* original para escrever os resultados no ficheiro csv.



Figura 14 - Programa em Qt

* 1. Git e Github

Com o auxílio da ferramenta de controlo de versões Git, foi utilizada a plataforma GitHub para hospedar os repositórios utilizados durante o desenvolvimento do projeto.

À medida que eram feitos progressos no desenvolvimento da aplicação, estes eram submetidos na plataforma. Foi utilizado apenas o *branch* *Master*, pois nunca chegou a haver conflitos, uma vez que o trabalho desenvolvido foi sempre repartido de forma a evitar a sua ocorrência.

* 1. RapidMiner

Com o auxílio deste *software* de *data science*, foi possível partir de uma grande quantidade de dados em bruto e obter informações relevantes sobre as diferentes combinações de parâmetros para os algoritmos, bem como condensar a informação referente à execução dos algoritmos para um grande número de problemas e *seeds*.

* 1. Síntese

Neste capítulo explicamos quais as linguagens/*software* com que trabalhámos durante o desenvolvimento deste projeto, tendo referido a necessidade de adaptação a uma nova linguagem e *framework*. Apesar da não familiaridade com estes tópicos, fomos capazes de provar a nossa capacidade para adaptação, tendo no longo-prazo conseguido alcançar o sucesso deste projeto.

1. - Aplicação

Este capítulo está estruturado da seguinte forma: a secção 4.1 apresenta a análise de requisitos efetuada para o desenvolvimento deste projeto; a secção 4.2 apresenta a metodologia utilizada no desenvolvimento deste projeto; a secção 4.3 apresenta os algoritmos evolutivos desenvolvidos; a secção 4.4 apresenta a estrutura do programa desenvolvido, e a secção 4.5 sumariza os conteúdos apresentados neste capítulo.

* 1. Análise de Requisitos

Os requisitos necessários para o desenvolvimento do programa foram os seguintes:

* Leitura de problemas a partir de ficheiros de texto;
* Criação de um conjunto de soluções para o problema;
* Melhoraria das soluções propostas com um algoritmo evolutivo;
* Observação da evolução das soluções em tempo real;
* Desenvolvimento de outro algoritmo evolutivo;
* Desenvolvimento de dois algoritmos de SI;
* Otimização do funcionamento dos algoritmos;
* Obtenção de combinações ideais de parâmetros para os algoritmos;
* Resolução de múltiplos problemas em simultâneo;
  1. Metodologia

A metodologia adotada no desenvolvimento deste projeto foi a de “desenvolvimento iterativo e incremental”. A cada etapa foram designados objetivos, depois foi feito o levantamento de requisitos, seguidos da implementação das funcionalidades e, por fim, foram aplicados os testes a cada funcionalidade nova implementada e testada a sua integração com as funcionalidades existentes.



Figura 15 - Desenvolvimento Iterativo e Incremental

* + 1. Diferentes Etapas de Desenvolvimento

A primeira etapa consistiu em implementar a leitura de problemas a partir de ficheiros de texto. Na segunda etapa foram criadas as classes Individual e Population, bem como a fórmula para o cálculo do *fitness* de um indivíduo. A terceira etapa consistiu na implementação do nosso primeiro algoritmo evolutivo (*Custom Algorithm*), bem como na forma de resolver os problemas utilizando o algoritmo implementado. Na quarta etapa foi implementada a forma de resolver múltiplos problemas em simultâneo em *threads* diferentes. Na quinta etapa foram implementados os recursos necessários para obter a melhor combinação de parâmetros para cada algoritmo. Na sexta e última etapa foram implementados os algoritmos evolutivos restantes.

1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Etapa | Semana 1 | Semana 2 | Semana 3 | Semana 4 | Semana 5 | Semana 6 | Semana 7 | Semana 8 |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |

2

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Etapa | Semana 9 | Semana 10 | Semana 11 | Semana 12 | Semana 13 | Semana 14 | Semana 15 | Semana 16 |
| 1 |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |

* + 1. Metodologia de Testes

Foram efetuados dois tipos de testes ao *software*: testes para validar o funcionamento correto e testes de *performance*.

De forma a validar o funcionamento correto do programa foi utilizada a ferramenta de *debugging* presente no IDE utilizado (*Qt Creator*).

De forma a testar a *performance* do programa foi utilizado o primeiro algoritmo implementado como base de comparação.

* 1. Algoritmos Evolutivos

Todos os algoritmos necessitam de uma população inicial. Esta não é gerada de forma totalmente aleatória, mas sim tendo em conta os nós com mais ligações, havendo maior probabilidade de colocar um regenerador nesses nós.

Todos os algoritmos são inicializados de forma diferente, mas o seu funcionamento principal passa pela execução da função *generateNewPopulation* até chegar à geração final introduzida nos parâmetros (condição de término).

* + 1. *Custom Algorithm*

O CA requer uma população inicial. A função de cálculo de *fitness* é aplicada a cada geração; para além de calcular o *fitness* dos indivíduos, também os ordena em ordem crescente de *fitness*.

Os parâmetros mais relevantes para este algoritmo são:

* Elitismo
* Probabilidade de mutação

O algoritmo funciona do seguinte modo:

Inicialização de parâmetros

Enquanto não é atingida a condição de termino

Cria uma nova população com estes passos:

* + - Elitismo
    - Recombinação
    - Mutação
    - Recombinação 2

Calcula fitness e ordena a nova população

Fim do ciclo

Pseudo-código 1 - Custom Algorithm

De seguida estão explicados os passos do algoritmo:

Elitismo

O elitismo é um parâmetro de entrada que age como uma forma de seleção. Aplicamos o elitismo à população presente no início de cada iteração do algoritmo de modo a manter os seus melhores indivíduos (no nosso caso temos a população organizada de melhor para pior). Ou seja, tendo uma população de 100 indivíduos e um elitismo de 20%, irão ser mantidos os primeiros 20 indivíduos. Estes indivíduos não são alterados na iteração atual.

Recombinação

Os restantes indivíduos não selecionados pelo elitismo são completamente refeitos, através de uma combinação dos diferentes genes dos indivíduos previamente selecionados, sendo que o gene na posição *n* do indivíduo a recombinar toma o valor do gene na posição *n* de um dos indivíduos selecionados por elitismo.

Mutação

De forma a saber onde aplicar mutações, é feito o cálculo para cada gene dos indivíduos selecionados por elitismo, para averiguar se algum gene apenas tem um valor; caso seja verdade, a primeira metade dos indivíduos não selecionados, segundo a probabilidade de mutação, pode ter o gene nessa posição alterado para o valor que não existe nos indivíduos selecionados.

A segunda metade dos indivíduos não selecionados, segundo a probabilidade de mutação, pode ter qualquer gene alterado para um valor aleatório dentro dos valores considerados no problema (0 e 1).

Recombinação 2

A segunda metade dos indivíduos selecionados é completamente refeita, através de uma combinação dos diferentes genes da primeira metade dos indivíduos selecionados por elitismo, sendo que o gene na posição *n* do indivíduo a recombinar toma o valor do gene na posição *n* de um dos indivíduos da primeira metade dos indivíduos selecionados por elitismo.

* + 1. Algoritmo Genético

O AG requer uma população inicial. Cada indivíduo desta população é avaliado utilizando a função de avaliação, cujo resultado deverá representar a qualidade do indivíduo como solução para o problema. Através do método de seleção elitismo e torneio, iremos obter uma nova população. De seguida, são aplicados os operadores genéticos de recombinação e mutação, respetivamente, com intuito de gerar novos indivíduos. Com cada geração, avaliamos a nova população gerada. No final de todo o processo evolutivo, o algoritmo devolve o melhor indivíduo encontrado.

Os parâmetros mais relevantes para este algoritmo são:

* Probabilidade de mutação
* Probabilidade de recombinação
* Tamanho de torneio
* Elitismo

O algoritmo funciona do seguinte modo:

Inicialização de parâmetros

Enquanto não é atingida a condição de termino

Cria uma nova população com estes passos:

* + - Elitismo
    - Torneio
    - Recombinação com um corte
    - Mutação

Calcula fitness e ordena a nova população

Cria nova população mantendo alguns dos melhores indivíduos da população inicial e os melhores da nova população

Calcula fitness e ordena a nova população original

Fim do ciclo

Pseudo-código 2 - Algoritmo Genético

De seguida estão explicados os passos do algoritmo:

Elitismo

O elitismo é um parâmetro de entrada que age como uma forma de seleção. Aplicamos o elitismo à população recebida no início de cada iteração do algoritmo, de modo a manter os seus melhores indivíduos (no nosso caso temos a população organizada de melhor para pior). Ou seja, tendo uma população de 100 indivíduos e um elitismo de 20%, irão ser mantidos os primeiros 20 indivíduos.

Os primeiros *x* indivíduos resultantes da aplicação do elitismo à população inicial passam a ser os primeiros *x* indivíduos da nova população.

Torneio

Em cada torneio, é escolhido aleatoriamente um elemento do conjunto de indivíduos inicial: este irá ser utilizado como "melhor indivíduo".

Para cada *n* iterações (sendo *n* o tamanho do torneio) é escolhido um outro indivíduo aleatoriamente do mesmo conjunto inicial, para que o seu *fitness* seja comparado com o do atual melhor indivíduo. Caso o *fitness* deste segundo indivíduo seja melhor, então este toma o lugar do anterior melhor indivíduo.

No fim do torneio, é devolvido o melhor indivíduo encontrado.

Nesta fase, teremos já alguns indivíduos na nova população graças ao elitismo e, através do torneio, serão atribuídos os restantes indivíduos a esta população.

Recombinação com um corte

Primeiramente, iremos percorrer todos os indivíduos da nova população. Em cada iteração começamos por testar se iremos recombinar (ou não) com base no parâmetro de probabilidade de recombinação; de seguida, determinamos aleatoriamente o corte a partir do número de elementos do indivíduo; por fim, iteramos o valor do corte e substituímos cada elemento com o elemento na mesma posição pertencente ao próximo indivíduo.

Ao concluir este processo, teremos a nova população recombinada.

Mutação

Primeiramente, iremos percorrer cada indivíduo da nova população. Com cada iteração, percorremos todos os elementos de cada um desses indivíduos. Para cada elemento, testamos a probabilidade de mutação para saber se iremos mutar (ou não). Caso seja para mutar, trocamos o valor desse elemento (se for 1 passa a 0, se for 0 passa a 1).

Ao concluir este processo, teremos a nova população mutada.

* + 1. Bee Colony Optimization

O BCO é um algoritmo proposto por Pham et al. (2005) [3]. Trata-se de um algoritmo de SI que imita o comportamento de abelhas produtoras de mel. Resumidamente, o algoritmo efetua um tipo de procura local e procura aleatória, semelhante ao reconhecimento das redondezas efetuado pelas abelhas da colónia.

O algoritmo funciona do seguinte modo:

Inicialização de parâmetros

Por cada iteração

Cálculo das probabilidades

Por cada abelha selecionada

Calcular o número de tentativas de otimização com base na probabilidade

Por cada tentativa

Otimizar solução

Fim do ciclo

Fim do ciclo

Calcula fitness e ordena a nova população

Fim do ciclo

Pseudo-código 3 - Bee Colony Optimization

De seguida são explicados os passos do algoritmo:

Inicialização de parâmetros

Nesta etapa iniciamos os parâmetros: número de abelhas selecionadas (*selectedSize*), número de abelhas consideradas melhores (*bestSize*), valor base de tentativas de otimização para as abelhas selecionadas (*valueSelection*), valor base de tentativas de otimização para abelhas consideradas melhores (*valueBest*) e número de modificações (*changeValue*).

Inicialização de probabilidades

A cada abelha selecionada (dentro do *selectedSize*) é atribuído um valor entre 0 e 1 com base na distância do seu *fitness* ao melhor *fitness* da população.

Em cada iteração, este valor de probabilidade é multiplicado pelos parâmetros *valueBest* (melhores abelhas) e *valueSelection* (restantes abelhas dentro de *selectedsize*), de modo a obter o número de tentativas de otimização para cada indivíduo numa iteração.

Otimizar solução

Otimizar solução consiste em, para um determinado número de modificações, remover ou adicionar um regenerador até o *fitness* do indivíduo melhorar; se o *fitness* não melhorar, o indivíduo não é alterado.

A probabilidade de remover um regenerador é o dobro da probabilidade de adicionar um regenerador.

A otimização pode ser feita de duas formas: considerar qualquer regenerador livre ou ocupado como alvo de otimização ou, à medida que o número de modificações aumenta, o mesmo número de melhores e piores regeneradores são considerados para serem modificados (sendo considerados como melhores regeneradores os que estão desligados e podem ligar mais nós e, como piores, os que estão ligados e ligam menos nós).

* + 1. Ant Colony Optimization

O ACO é baseado na forma como as formigas comunicam e se influenciam entre si através de trilhos de feromonas . Requer uma população inicial e vários parâmetros que serão referidos de seguida. O elemento mais importante é a variável “trilho de feromonas”, que contém toda a base do sistema, pois irá influenciar as outras formigas, sendo atualizado conforme a experiência do algoritmo. Acrescentamos, também, que foi utilizado um método de Busca Local, de modo a melhorar as soluções encontradas.

O algoritmo funciona do seguinte modo:

Inicialização de parâmetros

Função de Avaliação

Inicialização do trilho de feromonas

Por cada iteração

Por cada formiga na população

Clonar a atual formiga para um individuo auxiliar

Por cada modificação pretendida

Testa probabilidade exploração

Fim do ciclo

Testa intensificação

Fim do ciclo

Busca local

Função de Avaliação

Atualiza o trilho de feromonas

Fim do ciclo

Pseudo-código 4 - Ant Colony Optimization

De seguida são explicados os passos do algoritmo:

Inicialização de parâmetros

Nesta etapa iniciamos os parâmetros: valor que inicia o trilho de feromonas (*Q*), probabilidade de exploração (*probability\_q*), número de modificações (*number\_mods*), evaporação do trilho de feromonas (*evaporation*) e influência do trilho de feromonas (*influence*).

Iniciamos também o trilho de feromonas em si, que consiste num vetor de 3 dimensões .

Função de Avaliação

Primeiramente, é calculado o *fitness* de cada formiga, de modo a encontrar a melhor (que é depois guardada como a melhor formiga da iteração atual). Caso seja a primeira iteração ou caso o *fitness* da melhor formiga da iteração seja melhor que o *fitness* da melhor formiga já encontrada ao longo das várias iterações, então guardamos a melhor formiga, a melhor iteração e ativamos a intensificação.

Inicialização do trilho de feromonas

Aqui estabelecemos os valores do trilho de feromonas. O valor atribuído para cada trilho é o mesmo:

T = 1/(Q \* Fitness(G))

sendo G o melhor indivíduo encontrado.

Clonar a atual formiga para um indivíduo auxiliar

Desta forma, modificamos o clone da formiga em vez da formiga em si, testando melhoramentos sem consequências.

Testa probabilidade exploração

O algoritmo faz determinado número (indicado pelo utilizador) de modificações. Para cada uma destas, começamos por testar a probabilidade *q*. Independentemente do valor obtido, a formiga atual irá sempre interagir com o trilho de feromonas.

Testa intensificação

A intensificação é um parâmetro que é desativado quando não foi encontrada uma melhor solução naquela iteração e ativado quando é encontrada uma melhor solução. Isto permite uma exploração mais complexa. Após testar a intensificação, caso o *fitness* do indivíduo auxiliar seja melhor que o do melhor correspondente nas formigas, então o indivíduo auxiliar vai substituir o que está nas formigas.

Busca Local

De modo a melhorar os nossos resultados, implementámos uma busca local. Essencialmente, tentámos trocar o valor de um dos elementos de uma formiga e verificámos se melhora o *fitness* da solução.

Atualiza o trilho de feromonas

Os trilhos de feromonas são atualizados com as seguintes equações:

sendo G a melhor solução encontrada.

Notas:

Inicialmente, foi aplicado um sistema que contava as iterações passadas sem melhorias e que era utilizado como critério de paragem do algoritmo. Não é usado agora, dada a importância de ter o mínimo de regeneradores.

* 1. Estrutura do Programa

Com este capítulo pretende-se clarificar o modo como funciona o *software*. A utilização típica do programa consiste em selecionar um algoritmo, introduzir os parâmetros para o algoritmo e, em seguida, resolver um ou vários problemas, sendo que os resultados do problema individual são apresentados no programa enquanto os resultados da solução de uma série de problemas podem ser consultados num ficheiro csv.

* + 1. Classes

As classes que compõem o programa são as seguintes:

* AntColonyAlgorithm
* AntColonyDialog
* AntColonyMultiThread
* AntColonyTestDialog
* AntColonyTestMultiThread
* AntColonyThread
* BeeColonyAlgorithm
* BeeColonyDialog
* BeeColonyMultiThread
* BeeColonyTestDialog
* BeeColonyTestMultiThread
* BeeColonyThread
* CustomAlgorithm
* CustomDialog
* CustomMultiThread
* CustomTestDialog
* CustomTestMultiThread
* CustomThread
* GeneticAlgorithm
* GeneticDialog
* GeneticMultiThread
* GeneticTestDialog
* GeneticTestMultiThread
* GeneticThread
* Individual
* MainWindow
* Population
* Problem

A razão para haver tantas classes semelhantes para algoritmos diferentes é que cada algoritmo tem diferentes parâmetros, pelo que optámos por ter uma janela distinta para cada um, ao invés de uma janela dinâmica que se altera conforme é selecionado um algoritmo específico. Por outro lado, não é fácil implementar o polimorfismo em C++, devido ao *splicing* de objetos, e também seria prejudicial ao desempenho dos algoritmos.

Nas próximas secções é feita a descrição das diferentes classes, tendo algumas sido agrupadas devido às semelhanças que apresentam.

Algorithm

As classes Algorithm (AntColonyAlgorithm, BeeColonyAlgorithm, CustomAlgorithm, GeneticAlgorithm) contêm os seguintes métodos principais:

**setUpAlgorithm** **-** Funciona como o construtor da classe; o construtor está vazio de forma a poderem ser instanciadas sem argumentos. Com esta função, a mesma instância pode ser utilizada para uma diferente configuração de parâmetros.

**generateNewPopulation** **-** É a função que contém o funcionamento principal do algoritmo; conforme o funcionamento deste, será gerada uma nova população (solução), com base na população anterior. No subcapítulo anterior está apresentado em detalhe o funcionamento dos diferentes algoritmos.

Para além destes métodos principais, certos algoritmos têm ainda métodos adicionais, que auxiliam o método *generateNewPopulation*.

Dialog

Cada Dialog (AntColonyDialog, BeeColonyDialog, CustomDialog, GeneticDialog) é composto por uma série de *inputs* e *labels* correspondentes aos diferentes parâmetros de que o algoritmo necessita para correr, um *drop down* com o número de *threads* a utilizar e um *drop down* com o intervalo de *seeds* a executar na solução de múltiplos problemas. Os Dialogs têm também 3 botões cada um com uma função diferente, tendo estes os *labels* “Solve”, “Batch Solve” e “Test”. Para além de *inputs* e botões, os Dialogs têm também *labels* para informações do problema atual, uma zona chamada GridLayout que contém ou um gráfico do *fitness* em relação às gerações ou um conjunto de *labels* com informação sobre as diferentes *threads* em execução e uma barra de progresso representativa da execução atual do algoritmo.



Figura 16 - Custom Dialog

MultiThread

As classes MultiThread (AntColonyMultiThread, BeeColonyMultiThread, CustomMultiThread, GeneticMultiThread) contêm os seguintes métodos principais:

* run
* newProblem
* problemEnded

run é a função principal da *thread*; é esta que é executada quando se faz thread->start() dentro da *thread* original, atuando face a cada problema dentro da pasta selecionada. Se este for um dos problemas que esta *thread* tem de tratar, aplica o algoritmo ao mesmo problema com 10 *seeds* diferentes e envia os resultados de cada solução para a *thread* original.

newProblem é um sinal que é enviado à *thread* original quando a MultiThread começa a tratar um novo problema; é enviado o número da *thread* e o nome do problema que está a tratar, para ser mostrado no *grid layout* da janela.

problemEnded é um sinal que é enviado à *thread* original quando a MultiThread acaba de tratar uma *seed* para um problema; é enviada a *string* formatada de forma a ser escrita no ficheiro csv que está aberto na *thread* original.

TestDialog

As classes TestDialog (AntColonyTestDialog, BeeColonyTestDialog, CustomTestDialog, GeneticTestDialog) consistem em janelas com *labels* e *inputs* com valores de início, fim e incrementos para os diferentes parâmetros a variar; todos estes *inputs* têm *getters* a ser chamados pela janela original de forma a obter os valores introduzidos, sendo estes utilizados para construir as classes TestMultiThread.



Figura 17 - Ant Colony Test Dialog

TestMultiThread

As classes TestMultiThread (AntColonyTestMultiThread, BeeColonyTestMultiThread, CustomTestMultiThread, GeneticTestMultiThread) contêm os mesmos métodos principais que as classes MultiThread; contudo, a sua função *run,* ao invés de aplicar diferentes *seeds* ao mesmo problema, aplica diferentes combinações de parâmetros para o algoritmo.

Thread

As classes Thread (AntColonyThread, BeeColonyThread, CustomThread, GeneticThread) existem com o intuito de a janela original poder ser atualizada à medida que um algoritmo é aplicado a um problema. Contêm os seguintes métodos principais:

* Run
* dataChanged
* singleProblem

run é a função principal da *thread* e é esta que é executada quando se faz thread->start() na *thread* original; a função instancia o problema, a população e o algoritmo, notifica a *thread* original dos dados do problema e, à medida que o algoritmo vai gerando novas populações, notifica a *thread* original com os dados do melhor indivíduo da população atual.

dataChanged é o sinal que é enviado a cada iteração do algoritmo, enviando todos os dados necessários para atualizar a interface (*fitness*, nós desconectados, regeneradores, geração e progresso).

singleProblem é o sinal que é enviado quando um novo problema é iniciado; este contém os dados do problema e da população inicial (*fitness*, nós desconectados, regeneradores, geração, total de nós e total de ligações).

Individual

A classe Individual representa uma solução do problema; o seu construtor recebe o problema e prioriza colocar regeneradores nos nós com mais ligações. A classe tem os seguintes atributos:

* QVector<int> solution
* int fitness
* int disconnected
* int regenerators

solution é um vetor de inteiros, com tamanho *N*, sendo *N* o número de nós no problema; cada índice do vetor toma o valor 1 ou 0 conforme esse nó tem ou não um regenerador.

fitness é o valor inteiro obtido do cálculo do *fitness* do indivíduo.

disconnected é o valor inteiro do número de nós desconectados na solução.

regenerators é o valor inteiro do total de regeneradores na solução.

A classe Individual tem os seguintes métodos principais:

* calculateFitness
* clone

calculateFitness é uma função que verifica cada ligação do problema conforme a solução apresentada; por cada nó desconectado são somados 500 ao valor do *fitness*, por cada regenerador utilizado são somados 100 ao valor do *fitness*, ou 100 vezes o peso do nó para os problemas com pesos.

clone é uma função que cria uma nova instância da classe Individual, mas com novas referências de memória.

MainWindow

A classe main window apenas contém quatro botões que iniciam uma nova janela com o algoritmo selecionado e uma ferramenta que aplica a função sort, cuja finalidade é ler um ficheiro de resultados gerados pelo programa e ordená-los de forma a criar um ficheiro de mais fácil interpretação.



Figura 18 - Main Window

Population

O principal atributo da classe Population é QVector<Individual> individuals, um vetor de indivíduos.

Os principais métodos da classe Population são:

* setUpPopulation
* calculateFitnesses
* getBestIndividual

setUpPopulation permite criar uma nova população sem ter de criar uma nova instância da classe, permitindo ainda instanciar a classe sem saber inicialmente os parâmetros.

calculateFitnesses funciona da mesma forma que a função calculateFitness da classe Individual. Contudo, a utilização desta função torna-se mais rápida do que executar individualmente para cada indivíduo e apresenta ainda a vantagem de ordenar os indivíduos por ordem de *fitness*, pelo que acaba por ser mais usada que a anterior.

getBestIndividual é uma função que só deve ser chamada depois de os *fitnesses* terem sido calculados e ordenados, pois esta retorna o primeiro Individual da variável individuals.

Problem

Os atributos da classe Problem são os seguintes:

* QVector<QVector<int>> nodes
* QVector<int> weights
* QVector<float> connectionsWeight
* int hasWeight
* int total
* int connections

nodes é uma matriz de inteiros que contém as ligações entre os nós do problema, ou seja, sendo a coluna x e a linha y, se o elemento da matriz na posição x, y tiver o valor 1, significa que o nó no índice x está ligado ao nó no índice y; como tal, é uma matriz simétrica.

weights é um vetor de inteiros em que, caso o problema tenha pesos, cada índice contém o peso de colocar um regenerador no nó desse índice.

connectionsWeight é um vetor que toma valores entre 0 e 1, sendo 0 um nó que não tem ligações e 1 o nó com mais ligações.

hasWeight é um inteiro que toma o valor 1 quando o problema tem pesos.

total é o número de nós do problema.

connections é o número de ligações entre nós.

O principal método da classe Problem é setUpProblem, que permite criar um novo problema sem ter de criar uma nova instância da classe, e que permite que o objeto seja instanciado sem saber o argumento necessário. Este método recebe o caminho para o ficheiro do problema a ler, e desse ficheiro obtém o tamanho do problema, o número de ligações, os pesos dos nós (se existirem) e a matriz de ligações entre nós.

* 1. Síntese

A metodologia utilizada foi a que considerámos mais apropriada para o tamanho da nossa equipa e para os requisitos levantados. Cada algoritmo permite obter soluções para os problemas; todos eles têm semelhanças, como a utilização de populações, mas acabam por otimizar as populações de forma diferente. O programa tem uma utilização intuitiva e um funcionamento otimizado para a solução do PLR.

1. - Resultados

Este capítulo está estruturado da seguinte forma: a secção 5.1 apresenta a forma como foram obtidos os dados para análise; a secção 5.2 apresenta a forma como foi feito o tratamento dos dados obtidos; a secção 5.3 apresenta os resultados obtidos; a secção 5.4 apresenta a comparação dos algoritmos implementados; a secção 5.5 apresenta a escolha dos melhores algoritmos, e a secção 5.6 sumariza os conteúdos apresentados neste capítulo.

* 1. Obtenção de dados

Esta secção apresenta as etapas por que passámos de forma a obter os resultados prontos a serem tratados. Iremos explicar como obtivemos os dados das soluções e as combinações de parâmetros ideais, através de testes e do *software* RapidMiner.

* + 1. Obtenção de dados com a execução do programa desenvolvido

Os dados utilizados para analisar os algoritmos foram obtidos através da execução do programa com um processador Intel i5-8600k a correr com a frequência de 4.6GHz.

Os dados obtidos estão mais à frente neste capítulo, e os dados completos dos testes estão presentes em anexo. (FAÇAM AQUI REFERÊNCIA INDICANDO OS NOMES DOS ANEXOS OU o NÚMERO DAS TABELAS).

* + 1. Teste de parâmetros dos algoritmos

Para cada algoritmo foram testadas diferentes combinações de parâmetros para o funcionamento do algoritmo. Estas diferentes combinações foram aplicadas a 36 problemas e os resultados da execução foram escritos para ficheiros *csv*. Os dados escritos nos ficheiros consistem no valor dos parâmetros escolhidos, o tempo que o algoritmo demorou a obter a melhor solução, a geração/iteração em que o algoritmo chegou à melhor solução, o número de regeneradores da melhor solução e o número de nós desligados na melhor solução. Foi também criado um ficheiro de texto com os parâmetros do algoritmo que são constantes a todas as execuções dos testes do algoritmo.

* + 1. Recolha de dados de execução dos algoritmos

Para cada algoritmo foram executados 480 diferentes instâncias de problemas, cada uma com 50 *seeds* diferentes. Os resultados destas execuções foram escritos para ficheiros *csv*. Os dados escritos nos ficheiros consistem no tamanho do problema, o número do problema, a instância do problema, o tempo que o algoritmo demorou a obter a melhor solução, a geração em que o algoritmo chegou à melhor solução, o *fitness* da melhor solução, o número de regeneradores da melhor solução, e o número de nós desligados na melhor solução. Foram também criados ficheiros de texto com os parâmetros utilizados na execução dos algoritmos.

* 1. Tratamento dos dados obtidos

Os dados obtidos foram importados para a ferramenta *RapidMiner*, onde foram aplicadas funções de agregação, de forma a condensar milhares de linhas de dados em poucas linhas de informação relevante.

* + 1. Tratamento dos dados de teste de parâmetros

Após a importação dos dados para a ferramenta *RapidMiner*, foi aplicada a função de agregação, em que os dados são agrupados pelos diferentes atributos que foram variados durante os testes, sendo calculadas as médias dos atributos, tempo, gerações, regeneradores e nós desligados. As médias obtidas são ordenadas por nós desligados de forma crescente, número de regeneradores de forma crescente, e tempo de forma crescente.

A primeira linha do *dataset* obtido contém a melhor combinação de atributos para o algoritmo testado.

* + 1. Tratamento dos dados de execução dos algoritmos

Após a importação dos dados para a ferramenta *RapidMiner*, foi aplicada a função de agregação, em que os dados são agrupados pelos atributos número do problema e tamanho do problema, são filtradas as entradas em que o problema não ficou resolvido (entradas com nós desligados) e são calculadas as médias dos atributos, tempo, gerações, *fitness*, regeneradores e nós desligados.

Assim, os dados dos diferentes *seeds* e das diferentes instâncias de cada problema ficam agrupados, de forma a obter um sumário da execução do algoritmo para cada um dos problemas.

* 1. Resultados obtidos

Os resultados obtidos são apresentados sob a forma de tabelas. Para os resultados de testes, apenas são apresentadas as cinco melhores combinações de parâmetros; contudo, em anexo estão as tabelas com todos os parâmetros testados ().

* + 1. Resultados dos testes

Nesta secção estão presentes os resultados das melhores combinações de parâmetros para os seguintes algoritmos:

Custom Algorithm

Para a obtenção dos melhores parâmetros do CA, foram variados os seguintes parâmetros:

* Elitismo: de 5% a 50% com incrementos de 5%
* Mutação: de 1% a 10% com incrementos de 1%

Na Tabela 3 estão presentes as cinco melhores combinações de parâmetros para o CA.

3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Elitism | Mutation | Time(ms) | Regenerators | Disconnected |
| 20 | 5 | 320.8889 | 4.666666667 | 0 |
| 30 | 7 | 410.1389 | 4.666666667 | 0 |
| 20 | 9 | 180.25 | 4.694444444 | 0 |
| 30 | 6 | 217.9722 | 4.694444444 | 0 |
| 25 | 4 | 228.8056 | 4.694444444 | 0 |

Algoritmo Genético

Para a obtenção dos melhores parâmetros do AG, foram variados os seguintes parâmetros:

* Elitismo: de 5% a 20% com incrementos de 5%
* Mutação: de 20% a 40% com incrementos de 10%
* Recombinação: de 40% a 80% com incrementos de 20%
* Torneio: de 2 a 10 com incrementos de 2

Na Tabela 4 estão presentes as cinco melhores combinações de parâmetros para o AG.

**4**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Elitism | Mutation | Recombination | Tournament | Time(ms) | Regenerators | Disconnected |
| 10 | 30 | 60 | 10 | 2540.639 | 4.888888889 | 0 |
| 10 | 30 | 80 | 10 | 2544.917 | 4.888888889 | 0 |
| 20 | 20 | 40 | 10 | 4128.361 | 4.888888889 | 0 |
| 15 | 30 | 60 | 10 | 2995.306 | 4.916666667 | 0 |
| 15 | 30 | 80 | 10 | 2578.472 | 4.916666667 | 0 |

Bee Colony Optimization

Para a obtenção dos melhores parâmetros do BCO, foram variados os seguintes parâmetros:

* *Selected Value*: de 20 a 50 com incrementos de 10
* *Best Value*: de 30 a 60 com incrementos de 10
* *Change Value*: de 3 a 7 com incrementos de 1

Na Tabela 5 estão presentes as cinco melhores combinações de parâmetros para o BCO.

Tabela 5 - Cinco melhores resultados do teste do Bee Colony Optimization

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Select | Best | Change | Time(ms) | Regenerators | Disconnected |
| 30 | 50 | 5 | 3001.944 | 4.666666667 | 0 |
| 50 | 60 | 4 | 5773.667 | 4.666666667 | 0 |
| 40 | 50 | 5 | 5588.444 | 4.666666667 | 0 |
| 50 | 50 | 7 | 1860.083 | 4.694444444 | 0 |
| 30 | 60 | 6 | 1979.583 | 4.694444444 | 0 |

Ant Colony Optimization

Para a obtenção dos melhores parâmetros do ACO, foram variados os seguintes parâmetros:

* Q probability: de 60% a 100% com incrementos de 20%
* Modificações: de 5 a 7 com incrementos de 1
* Evaporação: de 30% a 70% com incrementos de 20%
* Influencia: de 60% a 100% com incrementos de 20%

Na Tabela 6 estão presentes as cinco melhores combinações de parâmetros para o ACO.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Q probability | Modifications | Evaporation | Influence | Time(ms) | Regenerators | Disconnected |
| 80 | 5 | 50 | 60 | 733.3889 | 4.861111111 | 0 |
| 60 | 5 | 50 | 80 | 1997.361 | 4.861111111 | 0 |
| 60 | 5 | 50 | 100 | 1999.028 | 4.861111111 | 0 |
| 80 | 5 | 50 | 80 | 1225.833 | 4.888888889 | 0 |
| 80 | 5 | 50 | 100 | 1230.194 | 4.888888889 | 0 |

* + 1. Resultados da solução de problemas

Nesta secção estão presentes os resultados das soluções das 480 instâncias de problemas com 50 *seeds* diferentes:

Custom Algorithm – Problemas *Random*

Os parâmetros utilizados para a solução das 280 instâncias de problemas *random* com o CA foram os seguintes:

* População: 500
* Gerações: 500
* Elitismo: 20%
* Mutação: 5%
* *Seeds*: todos os valores pares entre 2 e 100

7

| Size | Problem | Time(ms) | Generations | Fitness | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- | --- |
| 40 | 1 | 113.332 | 17.288 | 1108.8 | 11.088 | 0 |
| 40 | 2 | 101.448 | 15.248 | 605.2 | 6.052 | 0 |
| 40 | 3 | 110.356 | 16.142 | 453.6 | 4.536 | 0 |
| 40 | 4 | 63.992 | 8.864 | 382.4 | 3.824 | 0 |
| 40 | 5 | 50.788 | 6.944 | 300 | 3 | 0 |
| 40 | 6 | 29.972 | 3.64 | 300 | 3 | 0 |
| 40 | 7 | 39.914 | 5.6 | 200 | 2 | 0 |
| 40 | 8 | 25.04 | 3.768 | 200 | 2 | 0 |
| 40 | 9 | 21.204 | 3.178 | 200 | 2 | 0 |
| 60 | 1 | 457.618 | 56.584 | 1165.6 | 11.656 | 0 |
| 60 | 2 | 221.812 | 25.188 | 713.6 | 7.136 | 0 |
| 60 | 3 | 248.096 | 27.294 | 505.8 | 5.058 | 0 |
| 60 | 4 | 150.444 | 15.872 | 400 | 4 | 0 |
| 60 | 5 | 392.52 | 41.18 | 322.4 | 3.224 | 0 |
| 60 | 6 | 58.684 | 6.704 | 300 | 3 | 0 |
| 60 | 7 | 195.374 | 21.828 | 210 | 2.1 | 0 |
| 60 | 8 | 45.36 | 5.804 | 200 | 2 | 0 |
| 60 | 9 | 37.19 | 4.926 | 200 | 2 | 0 |
| 80 | 1 | 536.952 | 49.894 | 1296.4 | 12.964 | 0 |
| 80 | 2 | 504.532 | 45.428 | 755.6 | 7.556 | 0 |
| 80 | 3 | 489.296 | 39.956 | 550.2 | 5.502 | 0 |
| 80 | 4 | 699.364 | 51.908 | 445.2 | 4.452 | 0 |
| 80 | 5 | 370.012 | 27.17 | 383.4 | 3.834 | 0 |
| 80 | 6 | 125.912 | 10.816 | 300 | 3 | 0 |
| 80 | 7 | 204.66 | 17.306 | 281 | 2.81 | 0 |
| 80 | 8 | 74.444 | 7.76 | 200 | 2 | 0 |
| 80 | 9 | 58.12851 | 6.52409639 | 200 | 2 | 0 |
| 100 | 1 | 731.042 | 52.76 | 1339.2 | 13.392 | 0 |
| 100 | 2 | 948.032 | 61.856 | 811.6 | 8.116 | 0 |
| 100 | 3 | 1344.022 | 80.47 | 600.6 | 6.006 | 0 |
| 100 | 4 | 875.756 | 50.232 | 475.2 | 4.752 | 0 |
| 100 | 5 | 279.988 | 17.302 | 400 | 4 | 0 |
| 100 | 6 | 327.02 | 20.292 | 300 | 3 | 0 |
| 100 | 7 | 488.938 | 30.432 | 271.2 | 2.712 | 0 |
| 100 | 8 | 131.076 | 10.524 | 200 | 2 | 0 |
| 100 | 9 | 89.948 | 7.908 | 200 | 2 | 0 |

Custom Algorithm – Problemas *Weighted*

Os parâmetros utilizados para a solução das 200 instâncias de problemas *weighted* com o CA foram os seguintes:

* População: 500
* Gerações: 500
* Elitismo: 20%
* Mutação: 5%
* *Seeds*: todos os valores pares entre 2 e 100

8

| Size | Problem | Time(ms) | Generations | Fitness | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- | --- |
| 40 | 1 | 183.334 | 31.958 | 3054.2 | 11.322 | 0 |
| 40 | 3 | 57.872 | 8.784 | 1062.4 | 4.782 | 0 |
| 40 | 5 | 113.016 | 18.88 | 684.2 | 3.21 | 0 |
| 40 | 7 | 59.022 | 9.51 | 471.6 | 2.206 | 0 |
| 40 | 9 | 24.504 | 3.768 | 400 | 2 | 0 |
| 60 | 1 | 600.048 | 78.248 | 3033.6 | 12.026 | 0 |
| 60 | 3 | 270.76 | 30.478 | 1184.8 | 5.422 | 0 |
| 60 | 5 | 128.686 | 14.774 | 763 | 3.758 | 0 |
| 60 | 7 | 247.088 | 27.194 | 523.8 | 2.438 | 0 |
| 60 | 9 | 43.6 | 5.67 | 400 | 2 | 0 |
| 80 | 1 | 698.264 | 66.208 | 3351 | 13.598 | 0 |
| 80 | 3 | 574.44 | 47.42 | 1295.2 | 6.138 | 0 |
| 80 | 5 | 177.306 | 14.98 | 799.4 | 3.982 | 0 |
| 80 | 7 | 78.818 | 7.916 | 600 | 3 | 0 |
| 80 | 9 | 63.11647 | 7.09638554 | 400 | 2 | 0 |
| 100 | 1 | 900.5 | 67.85 | 3414.6 | 14.196 | 0 |
| 100 | 3 | 748.334 | 50.846 | 1366.8 | 6.66 | 0 |
| 100 | 5 | 519.838 | 33.052 | 807.6 | 4.008 | 0 |
| 100 | 7 | 230.66 | 17.218 | 594.4 | 2.944 | 0 |
| 100 | 9 | 92.172 | 8.706 | 400 | 2 | 0 |

Algoritmo Genético – Problemas *Random*

Os parâmetros utilizados para a solução das 280 instâncias de problemas *random* com o AG foram os seguintes:

* População: 500
* Gerações: 500
* Elitismo: 10%
* Mutação: 30%
* Recombinação: 60%
* Torneio: 10
* *Seeds*: todos os valores pares entre 2 e 100

9

| Size | Problem | Time(ms) | Generations | Fitness | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- | --- |
| 40 | 1 | 240.146 | 22.276 | 1105.8 | 11.058 | 0 |
| 40 | 2 | 259.036 | 23.312 | 638.8 | 6.388 | 0 |
| 40 | 3 | 339.088 | 29.766 | 473.8 | 4.738 | 0 |
| 40 | 4 | 189.632 | 17.172 | 393.6 | 3.936 | 0 |
| 40 | 5 | 411.502 | 36.744 | 306.2 | 3.062 | 0 |
| 40 | 6 | 88.432 | 8.324 | 300 | 3 | 0 |
| 40 | 7 | 550.324 | 51.722 | 201.4 | 2.014 | 0 |
| 40 | 8 | 95.728 | 10.128 | 200 | 2 | 0 |
| 40 | 9 | 74.926 | 8.238 | 200 | 2 | 0 |
| 60 | 1 | 836.478 | 61.056 | 1194.2 | 11.942 | 0 |
| 60 | 2 | 906.56 | 62.712 | 751.2 | 7.512 | 0 |
| 60 | 3 | 848.412 | 55.336 | 551.4 | 5.514 | 0 |
| 60 | 4 | 1201.948 | 75.144 | 435.6 | 4.356 | 0 |
| 60 | 5 | 718.66 | 45.142 | 377.2 | 3.772 | 0 |
| 60 | 6 | 412.124 | 28.56 | 300 | 3 | 0 |
| 60 | 7 | 963.758 | 67.4 | 257.4 | 2.574 | 0 |
| 60 | 8 | 271.768 | 23.368 | 200 | 2 | 0 |
| 60 | 9 | 166.498 | 16.092 | 200 | 2 | 0 |
| 80 | 1 | 1718.846 | 93.532 | 1370.2 | 13.702 | 0 |
| 80 | 2 | 1769.452 | 75.208 | 838.4 | 8.384 | 0 |
| 80 | 3 | 1802.14 | 80.454 | 620 | 6.2 | 0 |
| 80 | 4 | 1110.968 | 53.028 | 495.2 | 4.952 | 0 |
| 80 | 5 | 1174.76 | 55.122 | 398 | 3.98 | 0 |
| 80 | 6 | 1910.516 | 97.052 | 311.2 | 3.112 | 0 |
| 80 | 7 | 470.1 | 30.616 | 297.6 | 2.976 | 0 |
| 80 | 8 | 823.284 | 55.3 | 201.6 | 2.016 | 0 |
| 80 | 9 | 330.4056 | 27.815261 | 200 | 2 | 0 |
| 100 | 1 | 2631.874 | 112.894 | 1472 | 14.72 | 0 |
| 100 | 2 | 2353.244 | 98.912 | 916 | 9.16 | 0 |
| 100 | 3 | 2213.058 | 87.024 | 684 | 6.84 | 0 |
| 100 | 4 | 3102.552 | 111.308 | 525.6 | 5.256 | 0 |
| 100 | 5 | 2698.68 | 99.108 | 423.6 | 4.236 | 0 |
| 100 | 6 | 2335.768 | 96.252 | 360.4 | 3.604 | 0 |
| 100 | 7 | 1072.41 | 55.81 | 298 | 2.98 | 0 |
| 100 | 8 | 2260.832 | 121.32 | 224.8 | 2.248 | 0 |
| 100 | 9 | 681.67 | 49.454 | 200 | 2 | 0 |

Algoritmo Genético – Problemas *Weighted*

Os parâmetros utilizados para a solução das 200 instâncias de problemas *weighted* com o AG foram os seguintes:

* População: 500
* Gerações: 500
* Elitismo: 10%
* Mutação: 30%
* Recombinação: 60%
* Torneio: 10
* *Seeds*: todos os valores pares entre 2 e 100

10

| Size | Problem | Time(ms) | Generations | Fitness | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- | --- |
| 40 | 1 | 465.298 | 40.046 | 3034.2 | 11.35 | 0 |
| 40 | 3 | 513.378 | 42.92 | 1103.6 | 4.902 | 0 |
| 40 | 5 | 547.866 | 46.724 | 718.6 | 3.376 | 0 |
| 40 | 7 | 519.13 | 44.08 | 498.4 | 2.376 | 0 |
| 40 | 9 | 99.73 | 9.79 | 400 | 2 | 0 |
| 60 | 1 | 1303.688 | 85.666 | 3088.2 | 12.14 | 0 |
| 60 | 3 | 1348.474 | 76.266 | 1266.2 | 5.694 | 0 |
| 60 | 5 | 822.144 | 46.568 | 794.8 | 3.882 | 0 |
| 60 | 7 | 712.496 | 45.976 | 563.2 | 2.748 | 0 |
| 60 | 9 | 219.132 | 19.136 | 400 | 2 | 0 |
| 80 | 1 | 2169.326 | 118.024 | 3472.8 | 13.938 | 0 |
| 80 | 3 | 2094.072 | 97.162 | 1409 | 6.444 | 0 |
| 80 | 5 | 2107.706 | 97.132 | 838.2 | 4.058 | 0 |
| 80 | 7 | 701.002 | 43.216 | 600.2 | 3 | 0 |
| 80 | 9 | 398.6245 | 32.62851406 | 400 | 2 | 0 |
| 100 | 1 | 4490.504 | 173.892 | 3598.8 | 14.932 | 0 |
| 100 | 3 | 3711.526 | 132.232 | 1503 | 7.04 | 0 |
| 100 | 5 | 3518.2 | 122.434 | 911.6 | 4.406 | 0 |
| 100 | 7 | 1915.04 | 86.842 | 603.8 | 3.002 | 0 |
| 100 | 9 | 883.972 | 58.1 | 400 | 2 | 0 |

Bee Colony Optimization – Problemas *Random*

Os parâmetros utilizados para a solução das 280 instâncias de problemas *random* com o BCO foram os seguintes:

* População: 40
* Iterações: 500
* *Select Size*: 20
* *Best Size*: 10
* Select Value: 30
* Best Value: 50
* Change Value: 5
* *Seeds*: todos os valores pares entre 2 e 100

11

| Size | Problem | Time(ms) | Generations | Fitness | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- | --- |
| 40 | 1 | 2623.128 | 27.122 | 1081 | 10.81 | 0 |
| 40 | 2 | 1466.244 | 15.036 | 602.8 | 6.028 | 0 |
| 40 | 3 | 381.772 | 3.944 | 450.2 | 4.502 | 0 |
| 40 | 4 | 167.908 | 1.848 | 380 | 3.8 | 0 |
| 40 | 5 | 83.19 | 1.052 | 300 | 3 | 0 |
| 40 | 6 | 74.468 | 1 | 300 | 3 | 0 |
| 40 | 7 | 68.824 | 1.008 | 200 | 2 | 0 |
| 40 | 8 | 64.652 | 1 | 200 | 2 | 0 |
| 40 | 9 | 63.036 | 1 | 200 | 2 | 0 |
| 60 | 1 | 10748.53 | 70.568 | 1139.6 | 11.396 | 0 |
| 60 | 2 | 3159.484 | 20.528 | 701.6 | 7.016 | 0 |
| 60 | 3 | 747.09 | 4.858 | 500 | 5 | 0 |
| 60 | 4 | 283.144 | 2.04 | 400 | 4 | 0 |
| 60 | 5 | 3650.714 | 20.79 | 314.4 | 3.144 | 0 |
| 60 | 6 | 90.18 | 1.004 | 300 | 3 | 0 |
| 60 | 7 | 378.57 | 2.772 | 210 | 2.1 | 0 |
| 60 | 8 | 74.18 | 1 | 200 | 2 | 0 |
| 60 | 9 | 73.254 | 1 | 200 | 2 | 0 |
| 80 | 1 | 26429.82 | 125.854 | 1282.8 | 12.828 | 0 |
| 80 | 2 | 17422.69 | 78.468 | 737.6 | 7.376 | 0 |
| 80 | 3 | 9921.9 | 41.246 | 525.4 | 5.254 | 0 |
| 80 | 4 | 8733.152 | 33.144 | 408.4 | 4.084 | 0 |
| 80 | 5 | 8692.816 | 32.054 | 352.2 | 3.522 | 0 |
| 80 | 6 | 118.148 | 1.092 | 300 | 3 | 0 |
| 80 | 7 | 230.792 | 1.572 | 280 | 2.8 | 0 |
| 80 | 8 | 83.828 | 1.004 | 200 | 2 | 0 |
| 80 | 9 | 75.30924 | 1 | 200 | 2 | 0 |
| 100 | 1 | 39227.02 | 139.752 | 1353 | 13.53 | 0 |
| 100 | 2 | 19045.99 | 65.872 | 784 | 7.84 | 0 |
| 100 | 3 | 7832.966 | 24.726 | 590.6 | 5.906 | 0 |
| 100 | 4 | 11006.81 | 32.14 | 454.8 | 4.548 | 0 |
| 100 | 5 | 235.316 | 1.536 | 400 | 4 | 0 |
| 100 | 6 | 290.66 | 1.78 | 300 | 3 | 0 |
| 100 | 7 | 1053.738 | 4.066 | 270.2 | 2.702 | 0 |
| 100 | 8 | 196.108 | 1.544 | 200 | 2 | 0 |
| 100 | 9 | 80.628 | 1.016 | 200 | 2 | 0 |

**Bee Colony Optimization – Problemas *Weighted***

Os parâmetros utilizados para a solução das 200 instâncias de problemas *weighted* com o BCO foram os seguintes:

* População: 40
* Iterações: 500
* *Select Size*: 20
* *Best Size*: 10
* Select Value: 30
* Best Value: 50
* Change Value: 5
* *Seeds*: todos os valores pares entre 2 e 100

12

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Size | Problem | Time(ms) | Generations | Fitness | Regenerators | Disconnected |
| 40 | 1 | 2640.256 | 36.396 | 2984.4 | 11.094 | 0 |
| 40 | 3 | 468.35 | 6.146 | 1060 | 4.714 | 0 |
| 40 | 5 | 319.794 | 4.21 | 670 | 3.022 | 0 |
| 40 | 7 | 63.806 | 1.16 | 470 | 2.132 | 0 |
| 40 | 9 | 46.198 | 1.004 | 400 | 2 | 0 |
| 60 | 1 | 12428.27 | 105.646 | 2936.4 | 11.756 | 0 |
| 60 | 3 | 9988.838 | 74.946 | 1141.6 | 5.202 | 0 |
| 60 | 5 | 973.772 | 7.702 | 741.2 | 3.502 | 0 |
| 60 | 7 | 351.09 | 3.044 | 520 | 2.376 | 0 |
| 60 | 9 | 63.964 | 1.076 | 400 | 2 | 0 |
| 80 | 1 | 26007.45 | 163.724 | 3275 | 13.256 | 0 |
| 80 | 3 | 16247.17 | 90.898 | 1274.8 | 5.896 | 0 |
| 80 | 5 | 2714.716 | 14.65 | 782.2 | 3.822 | 0 |
| 80 | 7 | 234.65 | 2.072 | 600 | 2.986 | 0 |
| 80 | 9 | 75.51807 | 1.15662651 | 400 | 2 | 0 |
| 100 | 1 | 43405.91 | 195.014 | 3382.8 | 13.958 | 0 |
| 100 | 3 | 22089.21 | 87.392 | 1337 | 6.17 | 0 |
| 100 | 5 | 8551.19 | 30.878 | 803 | 4 | 0 |
| 100 | 7 | 431.136 | 2.822 | 590 | 2.9 | 0 |
| 100 | 9 | 183.85 | 1.806 | 400 | 2 | 0 |

Ant Colony Optimization – Problemas *Random*

Os parâmetros utilizados para a solução das 280 instâncias de problemas *random* com o ACO foram os seguintes:

* População: 50
* Iterações: 500
* *Q probability*: 80%
* *Q*: 35
* Modificações: 5
* Evaporação: 50%
* Influência: 60%
* *Seeds*: todos os valores pares entre 2 e 100



Tabela 13 - Resultado dos problemas random com Ant Colony Optimization

| Size | Problem | Time(ms) | Fitness | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- |
| 40 | 1 | 333.744 | 1086.6 | 10.866 | 0 |
| 40 | 2 | 281.076 | 609.6 | 6.096 | 0 |
| 40 | 3 | 185.372 | 462.2 | 4.622 | 0 |
| 40 | 4 | 144.216 | 388.4 | 3.884 | 0 |
| 40 | 5 | 161.458 | 305.8 | 3.058 | 0 |
| 40 | 6 | 30.084 | 300 | 3 | 0 |
| 40 | 7 | 167.38 | 223 | 2.23 | 0 |
| 40 | 8 | 29.3 | 200 | 2 | 0 |
| 40 | 9 | 29.818 | 200 | 2 | 0 |
| 60 | 1 | 3084.79 | 1148.4 | 11.484 | 0 |
| 60 | 2 | 1875.224 | 723.2 | 7.232 | 0 |
| 60 | 3 | 2544.218 | 517 | 5.17 | 0 |
| 60 | 4 | 1773.312 | 420.4 | 4.204 | 0 |
| 60 | 5 | 615.534 | 389.8 | 3.898 | 0 |
| 60 | 6 | 480.096 | 300 | 3 | 0 |
| 60 | 7 | 456.34 | 270.2 | 2.702 | 0 |
| 60 | 8 | 81.824 | 200 | 2 | 0 |
| 60 | 9 | 69.808 | 200 | 2 | 0 |
| 80 | 1 | 12194 | 1305.4 | 13.054 | 0 |
| 80 | 2 | 8363.872 | 794.8 | 7.948 | 0 |
| 80 | 3 | 5766.328 | 587 | 5.87 | 0 |
| 80 | 4 | 1081.952 | 488.4 | 4.884 | 0 |
| 80 | 5 | 761.77 | 398.8 | 3.988 | 0 |
| 80 | 6 | 2385.32 | 310 | 3.1 | 0 |
| 80 | 7 | 209.47 | 298.4 | 2.984 | 0 |
| 80 | 8 | 416.756 | 202.8 | 2.028 | 0 |
| 80 | 9 | 141.3273 | 200 | 2 | 0 |
| 100 | 1 | 24488.22 | 1406.4 | 14.064 | 0 |
| 100 | 2 | 22391.71 | 880.4 | 8.804 | 0 |
| 100 | 3 | 15933.47 | 656.8 | 6.568 | 0 |
| 100 | 4 | 15057.92 | 506 | 5.06 | 0 |
| 100 | 5 | 4741.38 | 409 | 4.09 | 0 |
| 100 | 6 | 4150.104 | 347.6 | 3.476 | 0 |
| 100 | 7 | 373.662 | 299.4 | 2.994 | 0 |
| 100 | 8 | 1423.48 | 252 | 2.52 | 0 |
| 100 | 9 | 265.686 | 200 | 2 | 0 |

Ant Colony Optimization – Problemas *Weighted*

Os parâmetros utilizados para a solução das 200 instâncias de problemas *weighted* com o ACO foram os seguintes:

* População: 50
* Iterações: 500
* *Q probability*: 80%
* *Q*: 35
* Modificações: 5
* Evaporação: 50%
* Influência: 60%
* *Seeds*: todos os valores pares entre 2 e 100



Tabela 14 - Resultados dos problemas weighted com Ant Colony Optimization

| Size | Problem | Time(ms) | Fitness | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- |
| 40 | 1 | 1682.534 | 3012.4 | 11.132 | 0 |
| 40 | 3 | 683.708 | 1083 | 4.708 | 0 |
| 40 | 5 | 970.106 | 781 | 3.384 | 0 |
| 40 | 7 | 259.832 | 544 | 2.532 | 0 |
| 40 | 9 | 147.856 | 406.2 | 2 | 0 |
| 60 | 1 | 8609.668 | 3055.2 | 11.77 | 0 |
| 60 | 3 | 7654.778 | 1281.4 | 5.46 | 0 |
| 60 | 5 | 6795.852 | 882.6 | 3.938 | 0 |
| 60 | 7 | 3320.496 | 603.2 | 2.86 | 0 |
| 60 | 9 | 1522.784 | 422.2 | 2 | 0 |
| 80 | 1 | 19679.34 | 3640.8 | 13.646 | 0 |
| 80 | 3 | 23094.08 | 1468.6 | 6.074 | 0 |
| 80 | 5 | 15789.94 | 939 | 4.034 | 0 |
| 80 | 7 | 5147.544 | 650.6 | 3 | 0 |
| 80 | 9 | 1583.245 | 435.94 | 2 | 0 |
| 100 | 1 | 40550.72 | 3911.1 | 14.66003976 | 0 |
| 100 | 3 | 55088.56 | 1655.8 | 6.702 | 0 |
| 100 | 5 | 30588.99 | 1042.4 | 4.232 | 0 |
| 100 | 7 | 9464.18 | 678.6 | 3.002 | 0 |
| 100 | 9 | 1140.922 | 421.8 | 2 | 0 |

* 1. Comparação dos algoritmos implementados



Tabela 15 - Resultados dos diferentes algoritmos para os diferentes tamanhos de problemas

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Custom Algorithm | | Genetic Algorithm | | Bee Colony Optimization | | Ant Colony Optimization | |
| Size | Time(ms) | Fitness | Time(ms) | Fitness | Time(ms) | Fitness | Time(ms) | Fitness |
| 40 | **70.99** | 673.03 | 313.87 | 683.89 | 609.40 | **664.17** | 364.75 | 685.87 |
| 60 | **221.23** | 708.76 | 766.58 | 741.39 | 3072.22 | **693.20** | 2777.48 | 743.83 |
| 80 | **332.52** | 775.53 | 1327.23 | 818.03 | 8356.28 | **758.46** | 6901.07 | 837.18 |
| 100 | **550.52** | 798.66 | 2419.24 | 865.83 | 10973.61 | **790.39** | 16118.50 | 904.81 |



Tabela 16 - Médias dos resultados dos diferentes algoritmos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algoritmo | Time(ms) | Generations | Fitness | Regenerators |
| Bee Colony Optimization | 5752.878 | 28.5210469 | **726.554** | **4.820928571** |
| Custom Algorithm | **293.8151** | **25.4490443** | 738.993 | 4.918928571 |
| Ant Colony Optimization | 6540.449 | 53.3997277 | 792.923 | 5.071643567 |
| Algoritmo Genético | 1206.73 | 61.8131388 | 777.282 | 5.187857143 |



Tabela 17 - Valor de tempo vezes fitness para os diferentes algoritmos

|  |  |
| --- | --- |
| Algoritmo | Tempo \* Fitness |
| Custom Algorithm | **2.17E+05** |
| Genetic Algorithm | 9.38E+05 |
| Bee Colony Optimization | 4.18E+06 |
| Ant Colony Optimization | 5.19E+06 |

O algoritmo que apresenta os melhores resultados é o algoritmo *Bee Colony Optimization*.

O algoritmo que apresenta resultados no menor espaço de tempo é o *Custom Algorithm*.

O algoritmo que apresenta resultados com o melhor valor é o *Custom Algorithm*.

Nem todos os algoritmos são apropriados a solução do RLP, o BCO obtém os melhores resultados, mas para problemas de grande dimensão é aconselhada a utilização do CA.

Com mais otimização será possível melhorar os resultados obtidos e também reduzir o tempo de execução.

Com a utilização de mais iterações será também possível obter melhores soluções, pois nem sempre é encontrada a solução ideal.

* 1. Síntese

Com a ferramenta *RapidMiner*, foi possível concentrar a informação obtida da execução de problemas no programa desenvolvido, de forma a ser mais fácil tirar conclusões dos dados alcançados. A grande quantidade de dados em bruto recolhidos permitiu, após o seu tratamento obter informação importante sobre o desempenho dos diferentes algoritmos.

1. - Conclusão

Este capítulo apresenta uma conclusão ao trabalho desenvolvido (secção 6.1) e aponta futuras direções para trabalho futuro (secção 6.2).

* 1. Considerações finais

Este projeto focou-se na resolução do problema de localização de regeneradores. Foram utilizados vários algoritmos de inteligência artificial: dois algoritmos evolutivos (CA e AG) e dois algoritmos de *swarm intellligence* (ACO e BCO).

Os resultados mostram que o melhor algoritmo depende dos critérios em que baseamos a nossa análise (tempo e qualidade das soluções). Contudo, o melhor em termos de *fitness* e consequentemente menos uso de regeneradores é o BCO. Os restantes algoritmos também se provaram eficientes no alcance de boas soluções, mas nenhum demonstra tão bons resultados em termos de qualidade de soluções. Em termos de tempo e alcance da melhor solução em menos tempo, o melhor algoritmo é o CA. Ao longo de todo este projeto corremos vários testes que apoiam estas afirmações, estando estes presentes neste relatório.

Durante este projeto trabalhámos pela primeira vez com algoritmos de *swarm intelligence*, que são bastante mais complexos e difíceis de implementar quando comparados com os restantes algoritmos implementados (ambos evolutivos). Ao início tivemos dificuldades com as implementações do ACO e do BCO, tendo sido necessário um extenso estudo do seu funcionamento. Contudo, conseguimos perceber o funcionamento destes algoritmos, o que levou ao sucesso das suas implementações e, consequentemente, ao sucesso deste projeto. Destacamos que este tipo de algoritmos foi o que permitiu os melhores resultados.

Este trabalho foi um desafio para nós, pois durante a nossa licenciatura em engenharia informática não tinha ainda surgido a oportunidade de lidarmos com certos aspetos importantes para este projeto. Com efeito, esta foi a nossa primeira oportunidade para trabalhar com algoritmos do tipo *Swarm Intelligence* mas, não obstante, conseguimos aplicá-los com sucesso e alcançar bons resultados. Na nossa licenciatura não abordámos C++, pelo que tivemos de aprender propositadamente a linguagem para este projeto, de modo a alcançar a melhor *performance* possível. Também não estávamos familiarizados com a plataforma QT, que foi a base do nosso trabalho.

A unidade curricular de Projetco Informático foi por isso essencial para a nossa formação académica e para a aprendizagem de competências técnicas, teóricas e práticas, de gestão de projeto e colaboração em equipa.

Finalizado o projeto e o presente relatório, que aborda todo o processo prático de desenvolvimento do projeto, bem como os conhecimentos teóricos que lhe serviram de base, podemos concluit que aTendo em conta todo o processo de desenvolvimento do projeto, considera-se que um dos principais aspetos foi a importância da aplicação prática dos conhecimentos previamente adquiridos, bem como a aprendizagem de novos conhecimentos e ainda o desenvolvimento de competências de relacionamento interpessoal em equipa, que garantiu uma gestão eficaz do projeto.

* 1. Trabalho futuro

Devido ao êxito deste trabalho, é equacionada a possibilidade da sua continuação. Um possível futuro projeto seria a criação de uma aplicação que mapeasse redes de fibra ótica do mundo real para o formato aceite pela nossa aplicação, possibilitando, deste modo, a utilização do nosso programa para problemas do mundo real. Poderíamos também fazer evoluir a nossa aplicação para que se torne apta a resolver mais problemas, em menos tempo, com mais algoritmos, com a obtenção do *hardware* necessário.

O uso ideal da nossa aplicação seria ao nível do planeamento da rede de fibra ótica, antes mesmo de esta ser instalada. Ou seja, o engenheiro responsável por determinar o seu mapeamento no mundo real leria o seu planeamento com o nosso *software*. Deste modo, poderá estimar de forma realista os custos, evitando gastos inesperados resultantes de queixas por falta de qualidade por parte de clientes do serviço.

Dado o tempo que nos foi disponibilizado para concluir este projeto, não foi possível fazer estudos comparativos com resoluções do PLR já existentes na literatura. De modo a complementar o trabalho já implementado por nós teríamos de realizar este estudo comparativo, pois iria fornecer uma base para a avaliação da qualidade da nossa resolução e poderia até mesmo possibilitar o melhoramento da nossa implementação.

* 1. Síntese

Todo este projeto foi um grande desafio que acabou por ter sucesso. Apesar de, inicialmente, desconhecermos a linguagem e alguns aspetos importantes, conseguimos superar essas dificuldades. Foi na totalidade uma experiência muito positiva, pois tivemos a oportunidade de lidar com um problema real e atual, e foi possível trabalhar na área de Inteligência Artificial, pela qual ambos temos interesse. Sentimos que contribuímos positivamente para a resolução do PLR e que nos desenvolvemos como estudantes de informática, tendo expandido o nosso conhecimento e competências na área da Inteligência Artificial.

# Bibliografia

|  |  |
| --- | --- |
| [1] | B. Akar e M. Tekalp, “Transport methods in 3DTV ...",” *IEEE Trans. On Circuits and Systems On Video Technology,* vol. 17, Novembro 2007. |



Anexo

Resultados dos testes

18

| Elitism | Mutation | Time(ms) | Regenerators | Disconnected |
| --- | --- | --- | --- | --- |
| 20 | 5 | 320.8889 | 4.666666667 | 0 |
| 30 | 7 | 410.1389 | 4.666666667 | 0 |
| 20 | 9 | 180.25 | 4.694444444 | 0 |
| 30 | 6 | 217.9722 | 4.694444444 | 0 |
| 25 | 4 | 228.8056 | 4.694444444 | 0 |
| 35 | 6 | 291.0556 | 4.694444444 | 0 |
| 45 | 8 | 318.6389 | 4.694444444 | 0 |
| 50 | 6 | 398.6111 | 4.694444444 | 0 |
| 45 | 6 | 428.7222 | 4.694444444 | 0 |
| 45 | 5 | 406.5556 | 4.694444444 | 0 |
| 45 | 9 | 421.9167 | 4.694444444 | 0 |
| 15 | 4 | 137.9722 | 4.722222222 | 0 |
| 10 | 8 | 190.5556 | 4.722222222 | 0 |
| 30 | 5 | 223.9722 | 4.722222222 | 0 |
| 25 | 8 | 238.3611 | 4.722222222 | 0 |
| 15 | 10 | 298.2778 | 4.722222222 | 0 |
| 35 | 9 | 275.9167 | 4.722222222 | 0 |
| 25 | 9 | 305 | 4.722222222 | 0 |
| 50 | 1 | 284.0833 | 4.722222222 | 0 |
| 15 | 6 | 298.6389 | 4.722222222 | 0 |
| 45 | 3 | 318.3611 | 4.722222222 | 0 |
| 30 | 10 | 323.5833 | 4.722222222 | 0 |
| 30 | 8 | 333.6667 | 4.722222222 | 0 |
| 35 | 2 | 318.8889 | 4.722222222 | 0 |
| 50 | 3 | 361.25 | 4.722222222 | 0 |
| 45 | 10 | 346.3333 | 4.722222222 | 0 |
| 25 | 3 | 333.7778 | 4.722222222 | 0 |
| 40 | 8 | 379.5278 | 4.722222222 | 0 |
| 45 | 4 | 386.5833 | 4.722222222 | 0 |
| 40 | 4 | 411.0556 | 4.722222222 | 0 |
| 25 | 1 | 414.5833 | 4.722222222 | 0 |
| 15 | 7 | 408.0833 | 4.722222222 | 0 |
| 30 | 9 | 432.9167 | 4.722222222 | 0 |
| 45 | 2 | 341.1667 | 4.722222222 | 0 |
| 50 | 10 | 423.25 | 4.722222222 | 0 |
| 40 | 10 | 441.3333 | 4.722222222 | 0 |
| 40 | 5 | 482.6389 | 4.722222222 | 0 |
| 10 | 2 | 106.8889 | 4.75 | 0 |
| 10 | 1 | 189.7778 | 4.75 | 0 |
| 10 | 5 | 185 | 4.75 | 0 |
| 20 | 4 | 196.6389 | 4.75 | 0 |
| 30 | 2 | 183.6111 | 4.75 | 0 |
| 15 | 8 | 196 | 4.75 | 0 |
| 15 | 5 | 230.9444 | 4.75 | 0 |
| 35 | 5 | 212.3333 | 4.75 | 0 |
| 15 | 2 | 205.3611 | 4.75 | 0 |
| 25 | 10 | 232.3056 | 4.75 | 0 |
| 35 | 3 | 229.2778 | 4.75 | 0 |
| 5 | 5 | 274.7778 | 4.75 | 0 |
| 40 | 2 | 248.5833 | 4.75 | 0 |
| 50 | 5 | 240.25 | 4.75 | 0 |
| 50 | 2 | 247.4444 | 4.75 | 0 |
| 50 | 4 | 272.3333 | 4.75 | 0 |
| 40 | 9 | 281.75 | 4.75 | 0 |
| 35 | 1 | 292.1389 | 4.75 | 0 |
| 35 | 4 | 273.5 | 4.75 | 0 |
| 20 | 3 | 303.3333 | 4.75 | 0 |
| 30 | 3 | 340.25 | 4.75 | 0 |
| 40 | 7 | 309.6111 | 4.75 | 0 |
| 20 | 7 | 345.5278 | 4.75 | 0 |
| 45 | 1 | 320.3889 | 4.75 | 0 |
| 40 | 6 | 404.1111 | 4.75 | 0 |
| 40 | 3 | 419.8611 | 4.75 | 0 |
| 50 | 8 | 425.8056 | 4.75 | 0 |
| 45 | 7 | 429.3611 | 4.75 | 0 |
| 50 | 7 | 480.0833 | 4.75 | 0 |
| 5 | 2 | 150.6389 | 4.777777778 | 0 |
| 25 | 2 | 163.7222 | 4.777777778 | 0 |
| 30 | 1 | 173.6111 | 4.777777778 | 0 |
| 30 | 4 | 183.1667 | 4.777777778 | 0 |
| 5 | 3 | 226.6667 | 4.777777778 | 0 |
| 20 | 6 | 201.1944 | 4.777777778 | 0 |
| 10 | 9 | 215.6389 | 4.777777778 | 0 |
| 20 | 1 | 222.6389 | 4.777777778 | 0 |
| 20 | 8 | 241.1111 | 4.777777778 | 0 |
| 5 | 6 | 267.5556 | 4.777777778 | 0 |
| 25 | 7 | 233.7222 | 4.777777778 | 0 |
| 20 | 10 | 240.3611 | 4.777777778 | 0 |
| 5 | 4 | 271.6667 | 4.777777778 | 0 |
| 50 | 9 | 279.8611 | 4.777777778 | 0 |
| 35 | 7 | 316.1111 | 4.777777778 | 0 |
| 40 | 1 | 345.0556 | 4.777777778 | 0 |
| 35 | 10 | 363.1389 | 4.777777778 | 0 |
| 10 | 6 | 396.4722 | 4.777777778 | 0 |
| 10 | 10 | 426.2222 | 4.777777778 | 0 |
| 20 | 2 | 141.5278 | 4.805555556 | 0 |
| 15 | 1 | 137.75 | 4.805555556 | 0 |
| 5 | 7 | 149.9167 | 4.805555556 | 0 |
| 10 | 7 | 192.9444 | 4.805555556 | 0 |
| 15 | 9 | 216.2222 | 4.805555556 | 0 |
| 25 | 5 | 213.9722 | 4.805555556 | 0 |
| 10 | 4 | 241.3056 | 4.805555556 | 0 |
| 25 | 6 | 247.0278 | 4.805555556 | 0 |
| 15 | 3 | 228.3056 | 4.805555556 | 0 |
| 35 | 8 | 339.8056 | 4.805555556 | 0 |
| 5 | 9 | 387.5556 | 4.805555556 | 0 |
| 5 | 1 | 106.2222 | 4.833333333 | 0 |
| 10 | 3 | 108.5 | 4.833333333 | 0 |
| 5 | 8 | 284.7222 | 4.833333333 | 0 |
| 5 | 10 | 228.9167 | 4.861111111 | 0 |

19

| Elitism | Mutation | Recombination | Tournament | Time(ms) | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 60 | 10 | 2540.639 | 4.888888889 | 0 |
| 10 | 30 | 80 | 10 | 2544.917 | 4.888888889 | 0 |
| 20 | 20 | 40 | 10 | 4128.361 | 4.888888889 | 0 |
| 15 | 30 | 60 | 10 | 2995.306 | 4.916666667 | 0 |
| 15 | 30 | 80 | 10 | 2578.472 | 4.916666667 | 0 |
| 20 | 30 | 60 | 6 | 2646.944 | 4.916666667 | 0 |
| 20 | 30 | 80 | 6 | 2636.361 | 4.916666667 | 0 |
| 20 | 30 | 60 | 8 | 3364.694 | 4.916666667 | 0 |
| 20 | 30 | 80 | 8 | 3297.611 | 4.916666667 | 0 |
| 15 | 20 | 40 | 6 | 4313.833 | 4.916666667 | 0 |
| 15 | 20 | 40 | 4 | 5665.083 | 4.916666667 | 0 |
| 15 | 30 | 60 | 8 | 2599.361 | 4.944444444 | 0 |
| 15 | 30 | 80 | 8 | 2739.611 | 4.944444444 | 0 |
| 15 | 30 | 60 | 6 | 2973.083 | 4.944444444 | 0 |
| 15 | 30 | 80 | 6 | 2821.778 | 4.944444444 | 0 |
| 20 | 20 | 40 | 8 | 4170.75 | 4.944444444 | 0 |
| 20 | 20 | 60 | 10 | 4557.5 | 4.944444444 | 0 |
| 20 | 20 | 80 | 10 | 4548.111 | 4.944444444 | 0 |
| 10 | 20 | 60 | 4 | 5386.75 | 4.944444444 | 0 |
| 10 | 20 | 80 | 4 | 5270.306 | 4.944444444 | 0 |
| 15 | 30 | 40 | 6 | 1713.722 | 4.972222222 | 0 |
| 20 | 40 | 40 | 10 | 1536 | 4.972222222 | 0 |
| 10 | 30 | 40 | 8 | 1965.194 | 4.972222222 | 0 |
| 10 | 30 | 60 | 4 | 2354.806 | 4.972222222 | 0 |
| 10 | 30 | 80 | 4 | 2387.556 | 4.972222222 | 0 |
| 10 | 20 | 40 | 10 | 4016.167 | 4.972222222 | 0 |
| 15 | 30 | 60 | 4 | 3557.417 | 4.972222222 | 0 |
| 15 | 30 | 80 | 4 | 4284.722 | 4.972222222 | 0 |
| 5 | 30 | 40 | 8 | 3816.083 | 4.972222222 | 0 |
| 20 | 20 | 60 | 8 | 4259.889 | 4.972222222 | 0 |
| 20 | 20 | 80 | 8 | 4310.75 | 4.972222222 | 0 |
| 20 | 30 | 40 | 10 | 1728.167 | 5 | 0 |
| 15 | 30 | 40 | 10 | 2030.083 | 5 | 0 |
| 20 | 30 | 60 | 4 | 2155.028 | 5 | 0 |
| 20 | 30 | 80 | 4 | 2156.167 | 5 | 0 |
| 20 | 40 | 40 | 8 | 2356.111 | 5 | 0 |
| 20 | 40 | 40 | 4 | 2112.472 | 5 | 0 |
| 15 | 30 | 40 | 8 | 3214.167 | 5 | 0 |
| 20 | 30 | 40 | 6 | 2491.222 | 5 | 0 |
| 20 | 30 | 60 | 2 | 2494.056 | 5 | 0 |
| 20 | 30 | 80 | 2 | 2500.083 | 5 | 0 |
| 5 | 30 | 60 | 4 | 2538.917 | 5 | 0 |
| 5 | 30 | 80 | 4 | 2631.083 | 5 | 0 |
| 20 | 30 | 40 | 4 | 3178.028 | 5 | 0 |
| 10 | 40 | 60 | 2 | 3024.639 | 5 | 0 |
| 10 | 40 | 80 | 2 | 3140.083 | 5 | 0 |
| 15 | 20 | 60 | 8 | 2929.278 | 5 | 0 |
| 15 | 20 | 80 | 8 | 3017.028 | 5 | 0 |
| 10 | 20 | 40 | 8 | 3893.139 | 5 | 0 |
| 10 | 20 | 60 | 6 | 3676.917 | 5 | 0 |
| 10 | 20 | 80 | 6 | 3509.5 | 5 | 0 |
| 20 | 20 | 60 | 6 | 3899.861 | 5 | 0 |
| 20 | 20 | 80 | 6 | 4010.056 | 5 | 0 |
| 10 | 30 | 40 | 2 | 3312.583 | 5 | 0 |
| 10 | 20 | 60 | 10 | 4010.667 | 5 | 0 |
| 10 | 20 | 80 | 10 | 3969.417 | 5 | 0 |
| 10 | 20 | 40 | 6 | 4326.889 | 5 | 0 |
| 20 | 30 | 40 | 2 | 3799.083 | 5 | 0 |
| 15 | 20 | 60 | 4 | 4499.722 | 5 | 0 |
| 15 | 20 | 80 | 4 | 4413.028 | 5 | 0 |
| 20 | 20 | 40 | 4 | 4680.5 | 5 | 0 |
| 20 | 20 | 40 | 2 | 6056.278 | 5 | 0 |
| 10 | 40 | 40 | 8 | 1546.944 | 5.027777778 | 0 |
| 10 | 40 | 60 | 8 | 2238.528 | 5.027777778 | 0 |
| 10 | 40 | 80 | 8 | 2328.083 | 5.027777778 | 0 |
| 20 | 30 | 40 | 8 | 2190.889 | 5.027777778 | 0 |
| 10 | 40 | 60 | 6 | 2168.111 | 5.027777778 | 0 |
| 10 | 40 | 80 | 6 | 2262.861 | 5.027777778 | 0 |
| 10 | 30 | 60 | 8 | 1964.167 | 5.027777778 | 0 |
| 10 | 30 | 80 | 8 | 1941.861 | 5.027777778 | 0 |
| 15 | 40 | 40 | 6 | 2478.25 | 5.027777778 | 0 |
| 20 | 40 | 40 | 6 | 2143.667 | 5.027777778 | 0 |
| 15 | 40 | 60 | 8 | 2621.389 | 5.027777778 | 0 |
| 15 | 40 | 80 | 8 | 2661.944 | 5.027777778 | 0 |
| 20 | 40 | 60 | 2 | 2274.611 | 5.027777778 | 0 |
| 20 | 40 | 80 | 2 | 2222.306 | 5.027777778 | 0 |
| 15 | 20 | 40 | 10 | 3126.278 | 5.027777778 | 0 |
| 15 | 20 | 60 | 10 | 2900.944 | 5.027777778 | 0 |
| 15 | 20 | 80 | 10 | 2931.25 | 5.027777778 | 0 |
| 20 | 40 | 40 | 2 | 2694.083 | 5.027777778 | 0 |
| 20 | 40 | 60 | 4 | 2774 | 5.027777778 | 0 |
| 20 | 40 | 80 | 4 | 2743.556 | 5.027777778 | 0 |
| 15 | 20 | 60 | 6 | 3793.25 | 5.027777778 | 0 |
| 15 | 20 | 80 | 6 | 3938.722 | 5.027777778 | 0 |
| 20 | 20 | 40 | 6 | 4181.194 | 5.027777778 | 0 |
| 15 | 20 | 40 | 8 | 4576.972 | 5.027777778 | 0 |
| 20 | 20 | 60 | 2 | 5745.167 | 5.027777778 | 0 |
| 20 | 20 | 80 | 2 | 5829.694 | 5.027777778 | 0 |
| 20 | 40 | 60 | 6 | 1832.25 | 5.055555556 | 0 |
| 20 | 40 | 80 | 6 | 1678.056 | 5.055555556 | 0 |
| 10 | 40 | 60 | 10 | 1759.833 | 5.055555556 | 0 |
| 10 | 40 | 80 | 10 | 1815.972 | 5.055555556 | 0 |
| 5 | 30 | 40 | 10 | 1696.167 | 5.055555556 | 0 |
| 20 | 30 | 60 | 10 | 1585 | 5.055555556 | 0 |
| 20 | 30 | 80 | 10 | 1581.889 | 5.055555556 | 0 |
| 5 | 40 | 40 | 8 | 1940.75 | 5.055555556 | 0 |
| 20 | 40 | 60 | 8 | 2025.528 | 5.055555556 | 0 |
| 20 | 40 | 80 | 8 | 1956.667 | 5.055555556 | 0 |
| 5 | 30 | 60 | 8 | 1785.806 | 5.055555556 | 0 |
| 5 | 30 | 80 | 8 | 2088.833 | 5.055555556 | 0 |
| 10 | 40 | 40 | 4 | 1875.472 | 5.055555556 | 0 |
| 20 | 40 | 60 | 10 | 2166.861 | 5.055555556 | 0 |
| 20 | 40 | 80 | 10 | 2078.528 | 5.055555556 | 0 |
| 15 | 40 | 60 | 2 | 2017.75 | 5.055555556 | 0 |
| 15 | 40 | 80 | 2 | 2037.556 | 5.055555556 | 0 |
| 10 | 30 | 40 | 4 | 2511.194 | 5.055555556 | 0 |
| 20 | 20 | 60 | 4 | 4410.528 | 5.055555556 | 0 |
| 20 | 20 | 80 | 4 | 4446.333 | 5.055555556 | 0 |
| 10 | 20 | 40 | 4 | 4681.917 | 5.055555556 | 0 |
| 10 | 20 | 40 | 2 | 5435.278 | 5.055555556 | 0 |
| 10 | 20 | 60 | 2 | 5673.333 | 5.055555556 | 0 |
| 10 | 20 | 80 | 2 | 5380.611 | 5.055555556 | 0 |
| 15 | 20 | 60 | 2 | 5545.806 | 5.055555556 | 0 |
| 15 | 20 | 80 | 2 | 5654.806 | 5.055555556 | 0 |
| 5 | 40 | 40 | 10 | 1694.194 | 5.083333333 | 0 |
| 15 | 40 | 60 | 4 | 1749.528 | 5.083333333 | 0 |
| 15 | 40 | 80 | 4 | 1809.667 | 5.083333333 | 0 |
| 10 | 40 | 60 | 4 | 2297.722 | 5.083333333 | 0 |
| 10 | 40 | 80 | 4 | 2063.861 | 5.083333333 | 0 |
| 15 | 30 | 40 | 4 | 2150.028 | 5.083333333 | 0 |
| 10 | 40 | 40 | 6 | 2280.583 | 5.083333333 | 0 |
| 15 | 30 | 60 | 2 | 2383.472 | 5.083333333 | 0 |
| 15 | 30 | 80 | 2 | 2664.111 | 5.083333333 | 0 |
| 5 | 40 | 60 | 8 | 2947.278 | 5.083333333 | 0 |
| 5 | 40 | 80 | 8 | 2802.194 | 5.083333333 | 0 |
| 5 | 30 | 60 | 6 | 2994.028 | 5.083333333 | 0 |
| 5 | 30 | 80 | 6 | 3094.972 | 5.083333333 | 0 |
| 10 | 30 | 60 | 2 | 2744.889 | 5.083333333 | 0 |
| 10 | 30 | 80 | 2 | 2738.167 | 5.083333333 | 0 |
| 10 | 40 | 40 | 2 | 2951.889 | 5.083333333 | 0 |
| 5 | 30 | 60 | 2 | 2920.833 | 5.083333333 | 0 |
| 5 | 30 | 80 | 2 | 2884.278 | 5.083333333 | 0 |
| 5 | 20 | 60 | 10 | 3732.583 | 5.083333333 | 0 |
| 5 | 20 | 80 | 10 | 3765.611 | 5.083333333 | 0 |
| 5 | 20 | 40 | 6 | 4188.444 | 5.083333333 | 0 |
| 5 | 40 | 60 | 10 | 1508.417 | 5.111111111 | 0 |
| 5 | 40 | 80 | 10 | 1495.667 | 5.111111111 | 0 |
| 15 | 40 | 40 | 4 | 1334.944 | 5.111111111 | 0 |
| 15 | 40 | 60 | 10 | 1967 | 5.111111111 | 0 |
| 15 | 40 | 80 | 10 | 1932.222 | 5.111111111 | 0 |
| 10 | 30 | 40 | 10 | 1474.056 | 5.111111111 | 0 |
| 10 | 40 | 40 | 10 | 1700.944 | 5.111111111 | 0 |
| 15 | 40 | 60 | 6 | 1540 | 5.111111111 | 0 |
| 15 | 40 | 80 | 6 | 1553.361 | 5.111111111 | 0 |
| 15 | 40 | 40 | 8 | 2247.111 | 5.111111111 | 0 |
| 5 | 30 | 40 | 6 | 2759.611 | 5.111111111 | 0 |
| 15 | 30 | 40 | 2 | 2466.028 | 5.111111111 | 0 |
| 15 | 20 | 40 | 2 | 5764.028 | 5.111111111 | 0 |
| 15 | 40 | 40 | 10 | 1423.528 | 5.138888889 | 0 |
| 5 | 40 | 40 | 4 | 1663.417 | 5.138888889 | 0 |
| 10 | 30 | 40 | 6 | 1973.306 | 5.138888889 | 0 |
| 15 | 40 | 40 | 2 | 2129.417 | 5.138888889 | 0 |
| 5 | 30 | 60 | 10 | 2351.083 | 5.138888889 | 0 |
| 5 | 30 | 80 | 10 | 2368.889 | 5.138888889 | 0 |
| 5 | 40 | 40 | 6 | 3095.472 | 5.138888889 | 0 |
| 5 | 30 | 40 | 4 | 3272.222 | 5.138888889 | 0 |
| 5 | 20 | 60 | 6 | 3956.306 | 5.138888889 | 0 |
| 5 | 20 | 80 | 6 | 4381.472 | 5.138888889 | 0 |
| 5 | 20 | 40 | 8 | 4077.417 | 5.138888889 | 0 |
| 5 | 20 | 40 | 4 | 4947.944 | 5.138888889 | 0 |
| 10 | 30 | 60 | 6 | 1978.25 | 5.166666667 | 0 |
| 10 | 30 | 80 | 6 | 1955.944 | 5.166666667 | 0 |
| 10 | 20 | 60 | 8 | 2820.417 | 5.166666667 | 0 |
| 10 | 20 | 80 | 8 | 2731.75 | 5.166666667 | 0 |
| 5 | 40 | 60 | 6 | 1891.778 | 5.194444444 | 0 |
| 5 | 40 | 80 | 6 | 2073.028 | 5.194444444 | 0 |
| 5 | 40 | 60 | 2 | 1903.056 | 5.194444444 | 0 |
| 5 | 40 | 80 | 2 | 2176.333 | 5.194444444 | 0 |
| 5 | 40 | 60 | 4 | 2443.333 | 5.194444444 | 0 |
| 5 | 40 | 80 | 4 | 2696.694 | 5.194444444 | 0 |
| 5 | 20 | 40 | 10 | 2915.417 | 5.194444444 | 0 |
| 5 | 20 | 60 | 8 | 3899 | 5.194444444 | 0 |
| 5 | 20 | 80 | 8 | 4260.25 | 5.194444444 | 0 |
| 5 | 20 | 60 | 4 | 3666.611 | 5.222222222 | 0 |
| 5 | 20 | 80 | 4 | 4346.278 | 5.222222222 | 0 |
| 5 | 20 | 40 | 2 | 5482.861 | 5.25 | 0 |
| 5 | 30 | 40 | 2 | 3642.833 | 5.277777778 | 0 |
| 5 | 20 | 60 | 2 | 5146.75 | 5.305555556 | 0 |
| 5 | 20 | 80 | 2 | 6178.25 | 5.305555556 | 0 |
| 5 | 40 | 40 | 2 | 3169.194 | 5.333333333 | 0 |

Tabela 20 - Resultados dos testes do Bee Colony Optimization

| Select | Best | Change | Time(ms) | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- |
| 30 | 50 | 5 | 3001.944 | 4.666666667 | 0 |
| 50 | 60 | 4 | 5773.667 | 4.666666667 | 0 |
| 40 | 50 | 5 | 5588.444 | 4.666666667 | 0 |
| 50 | 50 | 7 | 1860.083 | 4.694444444 | 0 |
| 30 | 60 | 6 | 1979.583 | 4.694444444 | 0 |
| 30 | 60 | 5 | 1464.861 | 4.694444444 | 0 |
| 30 | 60 | 7 | 3348.444 | 4.694444444 | 0 |
| 40 | 50 | 7 | 4141.361 | 4.694444444 | 0 |
| 40 | 40 | 7 | 3995.278 | 4.694444444 | 0 |
| 30 | 30 | 7 | 3807.472 | 4.694444444 | 0 |
| 20 | 50 | 5 | 2951.278 | 4.694444444 | 0 |
| 50 | 60 | 3 | 2696.028 | 4.694444444 | 0 |
| 20 | 50 | 4 | 2402.417 | 4.694444444 | 0 |
| 20 | 40 | 7 | 4162.917 | 4.694444444 | 0 |
| 50 | 50 | 6 | 7591.389 | 4.694444444 | 0 |
| 40 | 60 | 7 | 10289.81 | 4.694444444 | 0 |
| 30 | 40 | 3 | 2696 | 4.694444444 | 0 |
| 20 | 30 | 6 | 4722.583 | 4.694444444 | 0 |
| 50 | 40 | 6 | 8962.917 | 4.694444444 | 0 |
| 50 | 60 | 6 | 1481.194 | 4.722222222 | 0 |
| 50 | 60 | 5 | 1728.25 | 4.722222222 | 0 |
| 50 | 40 | 5 | 1372.389 | 4.722222222 | 0 |
| 20 | 60 | 5 | 1169.333 | 4.722222222 | 0 |
| 20 | 50 | 7 | 1787.694 | 4.722222222 | 0 |
| 50 | 50 | 5 | 1805.139 | 4.722222222 | 0 |
| 20 | 40 | 6 | 1430.611 | 4.722222222 | 0 |
| 40 | 50 | 4 | 1633.556 | 4.722222222 | 0 |
| 30 | 30 | 6 | 1972.889 | 4.722222222 | 0 |
| 50 | 40 | 3 | 1597.5 | 4.722222222 | 0 |
| 30 | 50 | 7 | 3887.694 | 4.722222222 | 0 |
| 40 | 50 | 6 | 3345.722 | 4.722222222 | 0 |
| 40 | 60 | 3 | 1650.611 | 4.722222222 | 0 |
| 40 | 30 | 4 | 1729.194 | 4.722222222 | 0 |
| 30 | 60 | 4 | 2543.028 | 4.722222222 | 0 |
| 30 | 40 | 7 | 3500.139 | 4.722222222 | 0 |
| 50 | 40 | 7 | 5211.333 | 4.722222222 | 0 |
| 20 | 60 | 6 | 3803.222 | 4.722222222 | 0 |
| 50 | 50 | 3 | 2337.806 | 4.722222222 | 0 |
| 20 | 30 | 5 | 2004.472 | 4.722222222 | 0 |
| 30 | 30 | 4 | 2062.139 | 4.722222222 | 0 |
| 30 | 30 | 5 | 2808.667 | 4.722222222 | 0 |
| 20 | 40 | 5 | 3211.972 | 4.722222222 | 0 |
| 40 | 60 | 6 | 9054.583 | 4.722222222 | 0 |
| 40 | 40 | 4 | 3307.167 | 4.722222222 | 0 |
| 40 | 30 | 7 | 626.9167 | 4.75 | 0 |
| 40 | 30 | 6 | 704.6111 | 4.75 | 0 |
| 50 | 30 | 6 | 1293.167 | 4.75 | 0 |
| 40 | 60 | 5 | 1198.25 | 4.75 | 0 |
| 20 | 60 | 7 | 1785.944 | 4.75 | 0 |
| 20 | 30 | 7 | 1244.222 | 4.75 | 0 |
| 50 | 30 | 3 | 901.5 | 4.75 | 0 |
| 40 | 40 | 3 | 1050 | 4.75 | 0 |
| 40 | 40 | 6 | 2600.972 | 4.75 | 0 |
| 30 | 50 | 6 | 2107.722 | 4.75 | 0 |
| 40 | 30 | 3 | 796.4167 | 4.75 | 0 |
| 20 | 50 | 6 | 1878.972 | 4.75 | 0 |
| 20 | 60 | 4 | 1402.722 | 4.75 | 0 |
| 40 | 40 | 5 | 1966.611 | 4.75 | 0 |
| 50 | 30 | 5 | 2263.083 | 4.75 | 0 |
| 20 | 40 | 3 | 825.0833 | 4.75 | 0 |
| 50 | 50 | 4 | 2438 | 4.75 | 0 |
| 50 | 60 | 7 | 4007.278 | 4.75 | 0 |
| 30 | 60 | 3 | 1226.833 | 4.75 | 0 |
| 20 | 60 | 3 | 1260.944 | 4.75 | 0 |
| 30 | 40 | 6 | 3481.222 | 4.75 | 0 |
| 30 | 40 | 5 | 2415.472 | 4.75 | 0 |
| 50 | 30 | 7 | 5058.556 | 4.75 | 0 |
| 40 | 60 | 4 | 2819.639 | 4.75 | 0 |
| 40 | 30 | 5 | 2837.25 | 4.75 | 0 |
| 30 | 40 | 4 | 2636.667 | 4.75 | 0 |
| 30 | 50 | 3 | 2039.139 | 4.75 | 0 |
| 20 | 30 | 4 | 2159.167 | 4.75 | 0 |
| 30 | 50 | 4 | 3914.444 | 4.75 | 0 |
| 30 | 30 | 3 | 795.6667 | 4.777777778 | 0 |
| 20 | 50 | 3 | 1008.222 | 4.777777778 | 0 |
| 40 | 50 | 3 | 1389.944 | 4.777777778 | 0 |
| 50 | 40 | 4 | 2287.472 | 4.777777778 | 0 |
| 20 | 30 | 3 | 1015.306 | 4.777777778 | 0 |
| 20 | 40 | 4 | 367.4722 | 4.805555556 | 0 |
| 50 | 30 | 4 | 1170.417 | 4.805555556 | 0 |

21

| Q probability | Modifications | Evaporation | Influence | Time(ms) | Regenerators | Disconnected |
| --- | --- | --- | --- | --- | --- | --- |
| 80 | 5 | 50 | 60 | 733.3889 | 4.861111111 | 0 |
| 60 | 5 | 50 | 80 | 1997.361 | 4.861111111 | 0 |
| 60 | 5 | 50 | 100 | 1999.028 | 4.861111111 | 0 |
| 80 | 5 | 50 | 80 | 1225.833 | 4.888888889 | 0 |
| 80 | 5 | 50 | 100 | 1230.194 | 4.888888889 | 0 |
| 60 | 5 | 50 | 60 | 2028.5 | 4.888888889 | 0 |
| 100 | 6 | 70 | 80 | 726.5833 | 4.916666667 | 0 |
| 100 | 6 | 70 | 60 | 728.4444 | 4.916666667 | 0 |
| 100 | 6 | 70 | 100 | 728.9167 | 4.916666667 | 0 |
| 100 | 5 | 50 | 60 | 939.7222 | 4.916666667 | 0 |
| 80 | 7 | 70 | 100 | 960.4444 | 4.916666667 | 0 |
| 60 | 5 | 30 | 60 | 1291.25 | 4.916666667 | 0 |
| 60 | 5 | 30 | 80 | 1294.222 | 4.916666667 | 0 |
| 60 | 5 | 30 | 100 | 1320.806 | 4.916666667 | 0 |
| 80 | 6 | 50 | 60 | 1759.639 | 4.916666667 | 0 |
| 80 | 6 | 50 | 80 | 1791.278 | 4.916666667 | 0 |
| 80 | 7 | 50 | 100 | 2139.083 | 4.916666667 | 0 |
| 80 | 5 | 30 | 100 | 728.1389 | 4.944444444 | 0 |
| 80 | 5 | 30 | 60 | 733.1389 | 4.944444444 | 0 |
| 80 | 5 | 30 | 80 | 741.1111 | 4.944444444 | 0 |
| 100 | 5 | 50 | 80 | 954.25 | 4.944444444 | 0 |
| 80 | 7 | 70 | 80 | 959.6389 | 4.944444444 | 0 |
| 100 | 5 | 50 | 100 | 961.0833 | 4.944444444 | 0 |
| 80 | 7 | 70 | 60 | 961.8889 | 4.944444444 | 0 |
| 60 | 6 | 50 | 100 | 1557.278 | 4.944444444 | 0 |
| 60 | 6 | 50 | 80 | 1568.861 | 4.944444444 | 0 |
| 60 | 6 | 50 | 60 | 1583.167 | 4.944444444 | 0 |
| 80 | 6 | 50 | 100 | 1798.306 | 4.944444444 | 0 |
| 80 | 7 | 50 | 80 | 2154.611 | 4.944444444 | 0 |
| 80 | 6 | 70 | 80 | 1036.139 | 4.972222222 | 0 |
| 80 | 6 | 70 | 100 | 1036.194 | 4.972222222 | 0 |
| 80 | 6 | 70 | 60 | 1046.861 | 4.972222222 | 0 |
| 60 | 7 | 70 | 100 | 1176.444 | 4.972222222 | 0 |
| 60 | 7 | 50 | 100 | 1176.5 | 4.972222222 | 0 |
| 60 | 7 | 30 | 60 | 1177.333 | 4.972222222 | 0 |
| 60 | 7 | 30 | 80 | 1177.611 | 4.972222222 | 0 |
| 60 | 7 | 30 | 100 | 1184 | 4.972222222 | 0 |
| 60 | 7 | 50 | 60 | 1187.194 | 4.972222222 | 0 |
| 60 | 7 | 70 | 80 | 1188.556 | 4.972222222 | 0 |
| 60 | 7 | 70 | 60 | 1190.528 | 4.972222222 | 0 |
| 60 | 7 | 50 | 80 | 1206.083 | 4.972222222 | 0 |
| 60 | 5 | 70 | 60 | 1329.861 | 4.972222222 | 0 |
| 100 | 6 | 50 | 60 | 1495.667 | 4.972222222 | 0 |
| 60 | 6 | 70 | 60 | 1499.722 | 4.972222222 | 0 |
| 100 | 7 | 50 | 100 | 1580.278 | 4.972222222 | 0 |
| 100 | 7 | 50 | 80 | 1624.361 | 4.972222222 | 0 |
| 80 | 7 | 50 | 60 | 2149.5 | 4.972222222 | 0 |
| 60 | 6 | 30 | 100 | 3834.361 | 4.972222222 | 0 |
| 60 | 6 | 30 | 60 | 3870.306 | 4.972222222 | 0 |
| 60 | 6 | 30 | 80 | 3898.5 | 4.972222222 | 0 |
| 80 | 7 | 30 | 80 | 953.4444 | 5 | 0 |
| 80 | 7 | 30 | 60 | 962.4167 | 5 | 0 |
| 80 | 7 | 30 | 100 | 970.0278 | 5 | 0 |
| 100 | 6 | 30 | 100 | 1201.861 | 5 | 0 |
| 100 | 6 | 30 | 60 | 1206 | 5 | 0 |
| 100 | 6 | 30 | 80 | 1219.528 | 5 | 0 |
| 60 | 6 | 70 | 100 | 1493.889 | 5 | 0 |
| 60 | 6 | 70 | 80 | 1504.194 | 5 | 0 |
| 100 | 6 | 50 | 100 | 1508.167 | 5 | 0 |
| 100 | 6 | 50 | 80 | 1515.222 | 5 | 0 |
| 100 | 7 | 50 | 60 | 1587 | 5 | 0 |
| 80 | 6 | 30 | 100 | 1589.722 | 5 | 0 |
| 80 | 6 | 30 | 80 | 1601.5 | 5 | 0 |
| 80 | 6 | 30 | 60 | 1601.667 | 5 | 0 |
| 60 | 5 | 70 | 80 | 1231.583 | 5.027777778 | 0 |
| 60 | 5 | 70 | 100 | 1345.25 | 5.027777778 | 0 |
| 100 | 5 | 30 | 60 | 1742.028 | 5.027777778 | 0 |
| 100 | 5 | 30 | 80 | 1750 | 5.027777778 | 0 |
| 100 | 5 | 30 | 100 | 1761.25 | 5.027777778 | 0 |
| 100 | 5 | 70 | 100 | 953.0556 | 5.055555556 | 0 |
| 100 | 5 | 70 | 60 | 958.4444 | 5.055555556 | 0 |
| 100 | 5 | 70 | 80 | 966.1111 | 5.055555556 | 0 |
| 100 | 7 | 70 | 100 | 1875.056 | 5.055555556 | 0 |
| 100 | 7 | 70 | 80 | 1892.139 | 5.055555556 | 0 |
| 100 | 7 | 70 | 60 | 1908.306 | 5.055555556 | 0 |
| 80 | 5 | 70 | 60 | 733.0833 | 5.083333333 | 0 |
| 100 | 7 | 30 | 60 | 2260 | 5.083333333 | 0 |
| 100 | 7 | 30 | 80 | 2268 | 5.083333333 | 0 |
| 100 | 7 | 30 | 100 | 2295.639 | 5.083333333 | 0 |
| 80 | 5 | 70 | 80 | 729.7778 | 5.111111111 | 0 |
| 80 | 5 | 70 | 100 | 737.1389 | 5.111111111 | 0 |