**Trabalho Prático 2**

**An artificial bee colony algorithm for the maximally diverse grouping problem.**



Licenciatura em Engenharia Informática

Ano letivo 2020/2021

Unidade Curricular de Introdução à Inteligência Artificial

2ºano - 1ºsemetre

André Lima Domingues 2019127839

Filipe Daniel Ramos Morgado 2019137625

Índice

[Introdução 4](#_Toc61801010)

[Algoritmos de Pesquisa Local 5](#_Toc61801011)

[Algoritmo Trepa Colinas com 1 Vizinhança 5](#_Toc61801012)

[Variáveis da experiência em análise: 6](#_Toc61801013)

[Análise do efeito da variação de parâmetros: 6](#_Toc61801014)

[Análise do impacto das componentes do algoritmo 6](#_Toc61801015)

[Algoritmo Trepa Colinas com 1 Vizinhança e aceitando soluções de custo igual 6](#_Toc61801016)

[Variáveis da experiência em análise: 7](#_Toc61801017)

[Análise do impacto das componentes do algoritmo: 7](#_Toc61801018)

[Algoritmo Trepa Colinas com 2 Vizinhanças e aceitando soluções de custo igual 7](#_Toc61801019)

[Variáveis da experiência em análise: 7](#_Toc61801020)

[Análise do impacto das componentes do algoritmo: 8](#_Toc61801021)

[Algoritmo Trepa Colinas Probabilístico 8](#_Toc61801022)

[Variáveis da experiência em análise: 8](#_Toc61801023)

[Análise do efeito da variação de parâmetros 8](#_Toc61801024)

[Análise do impacto das componentes do algoritmo: 8](#_Toc61801025)

[Algoritmo Recristalização Simulada 9](#_Toc61801026)

[Variáveis da experiência em análise: 9](#_Toc61801027)

[Análise do efeito da variação de parâmetros: 10](#_Toc61801028)

[Análise do impacto das componentes do algoritmo: 10](#_Toc61801029)

[Algoritmo Tabu 10](#_Toc61801030)

[Variáveis da experiência em análise: 10](#_Toc61801031)

[Análise do efeito da variação de parâmetros: 11](#_Toc61801032)

[Análise do impacto das componentes dos algoritmos: 11](#_Toc61801033)

[Algoritmo Evolutivo 11](#_Toc61801034)

[Variáveis da experiência em análise: 12](#_Toc61801035)

[Análises gerais do efeito da variação de parâmetros em todos os algoritmos evolutivos e híbridos: 12](#_Toc61801036)

[Algoritmo Evolutivo: Mutação binária + Recombinação de 1 ponto de corte 14](#_Toc61801037)

[Análise do impacto das componentes do algoritmo: 14](#_Toc61801038)

[Algoritmo Evolutivo: Mutação binária + Recombinação de 1 ponto de corte + Penalização 15](#_Toc61801039)

[Análise do impacto das componentes do algoritmo: 15](#_Toc61801040)

[Algoritmo Evolutivo: Mutação binária + Recombinação Uniforme 16](#_Toc61801041)

[Análise do impacto das componentes do algoritmo: 16](#_Toc61801042)

[Algoritmo Evolutivo: Mutação binária + Recombinação com 2 pontos de corte 17](#_Toc61801043)

[Análise do impacto das componentes do algoritmo: 17](#_Toc61801044)

[Algoritmo Evolutivo: Mutação por troca + Recombinação com 2 pontos de corte 18](#_Toc61801045)

[Análise do impacto das componentes do algoritmo: 18](#_Toc61801046)

[Algoritmo Híbrido 18](#_Toc61801047)

[Variáveis da experiência em análise: 19](#_Toc61801048)

[Algoritmo Híbrido: Mutação binária + Recombinação de 1 ponto de corte + Trepa Colinas 19](#_Toc61801049)

[Análise do impacto das componentes do algoritmo: 19](#_Toc61801050)

[Algoritmo Híbrido: Mutação binária + Recombinação de 1 ponto de corte + Trepa Colinas 20](#_Toc61801051)

[Análise do impacto das componentes do algoritmo: 20](#_Toc61801052)

[Algoritmo Híbrido: Mutação binária + Recombinação de 1 ponto de corte + Tabu 20](#_Toc61801053)

[Análise do impacto das componentes do algoritmo: 20](#_Toc61801054)

[Conclusão 21](#_Toc61801055)

[Bibliografia 22](#_Toc61801056)

# Introdução

A proposta de realização deste trabalho prático no âmbito da unidade curricular do 2ºano, 1ºsemetre de Introdução a Inteligência Artificial tem como objetivo a implementação de métodos de otimização que visem dividir um conjunto de elementos em subconjuntos menores, denominados grupos, de modo a que a diversidade entre os elementos em cada grupo seja a maior possível.

Dessa forma, os problemas de otimização são caracterizados por um grande universo de soluções o que dificulta a obtenção de uma solução de custo ótimo através de um algoritmo exato em tempo computacional admissível.

Neste trabalho são solucionadas algumas metodologias para gerar soluções para o problema mencionado:

* **Algoritmo de pesquisa local**
  + Trepa Colinas (1 vizinhança);
  + Trepa Colinas (1 vizinhança e aceitando soluções de custo igual);
  + Trepa Colinas (2 vizinhanças e aceitando soluções de custo igual);
  + Trepa Colinas (probabilístico);
  + Recristalização Simulada
  + Pesquisa Tabu
* **Algoritmo evolutivo**
  + Mutação binária + recombinação com 1 ponto de corte;
  + Mutação binária + recombinação com 1 ponto de corte (com penalização);
  + Mutação binária + recombinação com 2 pontos de corte;
  + Mutação binária + recombinação uniforme;
  + Mutação por troca + recombinação com 2 pontos de troca;
  + Estratégia de Penalização;
* **Algoritmo Híbrido**
  + Algoritmo Evolutivo + Trepa Colinas Simples;
  + Algoritmo Evolutivo + Trepa Colinas 2 vizinhos;
  + Algoritmo Evolutivo + Tabu;

# Algoritmos de Pesquisa Local

O algoritmo de pesquisa local, tem por objetivo procurar as melhores soluções explorando a vizinhança da solução inicial de modo a obter a melhor solução.

Este algoritmo parte de um estado inicial dado ou gerado aleatoriamente, para avaliar os seus estados sucessores. O algoritmo consta com a estratégia da **melhor vizinhança,** que a partir de uma solução corrente, avalia cada estado gerado e escolhe o de maior valor. Em alguns algoritmos específicos, tal com o Trepa Colinas probabilístico, é possível aceitar soluções piores como forma de escapar a máximos locais. É também possível em alguns algoritmos ser aceite soluções iguais, caso o algoritmo se encontre numa planície, ou seja, num máximo local na maior parte das vezes, podendo assim escapar desse mesmo máximo local. O processo é repetido até quando atingir o número de reinícios máximo.

Assim foram implementados os seguintes algoritmos:

* **Trepa Colinas (1 vizinhança) –** neste algoritmo pretende-se avaliar o custo de uma solução atual (inicialmente solução inicial). Para isso, gera-se um vizinho, avalia-se o seu custo, e compara-se esse custo com o custo da solução atual. De seguida, como estamos perante um **problema de maximização** aceita-se o custo da vizinhança, caso este seja superior ao custo da solução atual.

É repetido este processo um certo número de iterações predefinido.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NUM ELEMENTOS E SUBCONJUNTOS |  | 100  iterações | 1000  iterações | 5000  iterações | 10000  iterações |
| 10,2: n010.txt | **Melhor** | **1228** | **1228** | **1228** | **1228** |
| **MBF** | **1228** | **1228** | **1228** | **1228** |
| 12,4: n012.txt | **Melhor** | **1000** | **1000** | **1000** | **1000** |
| **MBF** | **966,7** | **980,24** | **984,92** | **987,5** |
| 30,5: n030.txt | **Melhor** | **4858** | **5179** | **5194** | **5194** |
| **MBF** | **4547,28** | **5018,76** | **5037,56** | **5064,74** |
| 60,6: n060.txt | **Melhor** | **16489** | **18012** | **18386** | **18520** |
| **MBF** | **15795,48** | **17560** | **18069,92** | **18189,54** |
| 120,10: n120.txt | **Melhor** | **37518** | **42427** | **44461** | **44756** |
| **MBF** | **36572,519** | **41483,8** | **43667,89** | **44118,55** |
| 240,12: n240.txt | **Melhor** | **124347** | **136519** | **143384** | **146191** |
| **MBF** | **120895,28** | **134411,78** | **142229,23** | **144448,18** |

## Algoritmo Trepa Colinas com 1 Vizinhança

### Variáveis da experiência em análise:

* Número de iterações entre 100 e 10000;
* Realização de 50 testes para cada ficheiro.

### Análise do efeito da variação de parâmetros:

Através da tabela de resultados, verifica-se que um maior número de iterações promove uma melhor solução em todos os ficheiros testados, uma vez que existe uma maior probabilidade de encontrar a diversidade máxima do problema.

Como o número de elementos e subconjuntos vai aumentando progressivamente, cada vez mais o número de iterações influencia a qualidade da melhor solução. De maneira que, existindo vários elementos e subconjuntos ,100 iterações podem não ser suficiente para determinar a solução ótima.

# Análise do impacto das componentes do algoritmo

Tendo em conta que este algoritmo é bastante simples uma vez que só tem em conta um vizinho e apenas substitui a melhor solução se o vizinho for maior que a solução atual, tem uma maior probabilidade de ficar preso num máximo de local.

* **Trepa Colinas (1 vizinhança e aceitando de soluções de custo igual) –** neste algoritmo pretende-se avaliar o custo de uma solução atual (inicialmente solução inicial). Para isso, gera-se um vizinho, avalia-se o seu custo, e compara-se esse custo com o custo da solução atual. De seguida, como estamos perante um **problema de maximização** aceita-se o custo da vizinhança, caso este seja superior ou igual ao custo da solução atual.

É repetido este processo um certo número de iterações predefinido.

## Algoritmo Trepa Colinas com 1 Vizinhança e aceitando soluções de custo igual

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NUM ELEMENTOS E SUBCONJUNTOS |  | 100  iterações | 1000  iterações | 5000  iterações | 10000  iterações |
| 10,2: n010.txt | **Melhor** | **1228** | **1228** | **1228** | **1228** |
| **MBF** | **1228** | **1228** | **1228** | **1228** |
| 12,4: n012.txt | **Melhor** | **1000** | **1000** | **1000** | **1000** |
| **MBF** | **968,18** | **982,53** | **987,7** | **988,89** |
| 30,5: n030.txt | **Melhor** | **4820** | **5169** | **5194** | **5194** |
| **MBF** | **4551,48** | **5028,82** | **5050,32** | **5066,82** |
| 60,6: n060.txt | **Melhor** | **16282** | **17952** | **18484** | **18461** |
| **MBF** | **15750,94** | **17486** | **18133,64** | **18192,84** |
| 120,10: n120.txt | **Melhor** | **37933** | **42603** | **44366** | **44730** |
| **MBF** | **36732,74** | **41460** | **43591,74** | **44200,58** |
| 240,12: n240.txt | **Melhor** | **122806** | **136738** | **143647** | **146712** |
| **MBF** | **120765,44** | **134607,3** | **142115,4** | **144590** |

### Variáveis da experiência em análise:

* Número de iterações entre 100 e 10000;
* Realização de 50 testes para cada ficheiro.

### Análise do impacto das componentes do algoritmo:

Este algoritmo é ligeiramente melhor que o anterior já que substitui a melhor solução se o vizinho for ***maior ou igual*** que a solução atual, aumentando assim a probabilidade de não ficar bloqueado num máximo local.

* **Trepa Colinas (2 vizinhanças) –** neste algoritmo pretende-se avaliar o custo de uma solução atual (inicialmente solução inicial). Comparativamente com os métodos anteriores, este método gera duas vizinhanças avaliando os seus custos e comparando esse custo com a solução atual. Seguidamente, como se trata de um **problema de maximização**, apenas se aceita novas soluções caso o custo da vizinhança 1 seja superior ou igual ao custo da solução atual e superior ou igual ao custo da vizinhança 2 ou caso o custo da vizinhança 2 seja superior ou igual ao custo da solução atual e superior ou igual ao custo da vizinhança 1.

É repetido este processo o número de interações predefinido, e como espectável este algoritmo exige um tempo computacional mais alargado devido ao elevado número de comparações entre soluções.

## Algoritmo Trepa Colinas com 2 Vizinhanças e aceitando soluções de custo igual

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NUM ELEMENTOS E SUBCONJUNTOS |  | 100  iterações | 1000  iterações | 5000  iterações | 10000  iterações |
| 10,2: n010.txt | **Melhor** | **1228** | **1228** | **1228** | **1228** |
| **MBF** | **1228** | **1228** | **1228** | **1228** |
| 12,4: n012.txt | **Melhor** | **1000** | **1000** | **1000** | **1000** |
| **MBF** | **986,2** | **991,88** | **992,2** | **995,85** |
| 30,5: n030.txt | **Melhor** | **4912** | **5194** | **5194** | **5194** |
| **MBF** | **4710,98** | **5032,62** | **5062,32** | **5067,4** |
| 60,6: n060.txt | **Melhor** | **17187** | **18096** | **18524** | **18579** |
| **MBF** | **16344,62** | **17808,76** | **18175,51** | **18258,62** |
| 120,10: n120.txt | **Melhor** | **39027** | **43015** | **44817** | **45226** |
| **MBF** | **38038,12** | **42433,86** | **44201,32** | **44731,4** |
| 240,12: n240.txt | **Melhor** | **126073** | **139418** | **146003** | **148021** |
| **MBF** | **124054,9** | **138002** | **144306,42** | **146776** |

### Variáveis da experiência em análise:

* Número de iterações entre 100 e 10000;
* Realização de 50 testes para cada ficheiro.

### Análise do impacto das componentes do algoritmo:

Este algoritmo é melhor que o algoritmo em que esta incluído apena um vizinho pois tem em conta ***dois vizinhos***, duplicando assim a chance de encontrar uma melhor solução, aumentando obviamente assim a probabilidade de encontrar o ótimo global.

* **Trepa Colinas (probabilístico) –** este algoritmo apresenta o mesmo método de avaliação de soluções do algoritmo “Trepa Colinas (1 vizinhança)”, mas aceita também soluções piores consoante uma certa probabilidade predefinida.

É repetido este processo um certo número de interações predefinido.

## Algoritmo Trepa Colinas Probabilístico

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NUM ELEMENTOS E SUBCONJUNTOS |  | 100 iterações Prob = 0.01 | 5000 iterações Prob = 0.01 | 100 iterações Prob = 0.005 | 5000 iterações Prob = 0.005 |
| 10,2: n010.txt | **Melhor** | **1228** | **1221** | **1228** | **1228** |
| **MBF** | **1218,74** | **1228** | **1213,98** | **1218,22** |
| 12,4: n012.txt | **Melhor** | **1000** | **1000** | **1000** | **1000** |
| **MBF** | **966,85** | **981,72** | **955,74** | **969,27** |
| 30,5: n030.txt | **Melhor** | **4814** | **5194** | **4782** | **5153** |
| **MBF** | **4564,98** | **5030,1** | **4513,97** | **4900** |
| 60,6: n060.txt | **Melhor** | **16717** | **18322** | **16362** | **18018** |
| **MBF** | **15813,14** | **17885** | **15708,64** | **17428,86** |
| 120,10: n120.txt | **Melhor** | **38026** | **44106** | **38296** | **43085** |
| **MBF** | **36626,94** | **43236,8** | **36563** | **41889,98** |
| 240,12: n240.txt | **Melhor** | **123691** | **142625** | **123473** | **141178** |
| **MBF** | **121202,5** | **141390,44** | **120773,48** | **139298,16** |

### Variáveis da experiência em análise:

* Número de iterações entre 100 e 5000;
* Variação da probabilidade de 0.001 e de 0,005.
* Realização de 50 testes para cada ficheiro.

### Análise do efeito da variação de parâmetros

É possível verificar nesta experimentação que a melhor solução é quando a probabilidade se encontra em 0.01% em vez de 0.05%. Isto comprova o facto de aceitar demasiados vizinhos com uma solução pior, leva a uma pior solução final, como é lógico. As iterações, confirmam novamente também que o quanto maior o número de iterações for, o algoritmo melhora consideravelmente.

### Análise do impacto das componentes do algoritmo:

Este algoritmo é acaba por ser melhor que o algoritmo “Trepa Colinas com 1 vizinhança” pois tem a probabilidade de em algumas das vezes que analisa uma vizinhança escolher uma vizinhança pior, em que aparentemente é uma pior decisão. No entanto, como a probabilidade de isto acontecer é bastante baixa, não corrompe o objetivo do algoritmo e tem uma pequena probabilidade de ao selecionar o vizinho pior, esse mesmo vizinho ter um outro vizinho que tenha a melhor solução encontrada até ao momento.

**Recristalização simulada –** O algoritmo local Recristalização Simulada é bastante semelhante ao algoritmo local “Trepa Colinas”, contudo pretende ultrapassar máximos locais tendo em conta o processo de arrefecimento. A temperatura contribui para a melhor solução possível, visto que a probabilidade de aceitar soluções piores é influenciada diretamente pela temperatura, e a probabilidade de aceitar soluções piores pode se revelar essencial para o algoritmo não ficar preso num ótimo local. No entanto como a temperatura baixa ao longo da sua execução, o algoritmo cada vez mais se torna semelhante ao “Trepa Colinas”. A variação lenta da temperatura é determinada com o aumento do número de iterações do algoritmo.

* Neste algoritmo, a probabilidade de aceitar vizinhos piores deteriora-se seguindo a seguinte fórmula:
  + - **eprob = exp((custo - custo\_viz) / temperatura)**
* Seguidamente aceita-se o custo da vizinhança, caso este seja superior ou igual ao custo da solução atual, uma vez que estamos perante um problema de maximização.

Contudo, caso seja inferior aceitasse o custo, mas apenas com uma certa probabilidade ***(random < eprob)***, que vai ter em conta o “*eprob*” entrando aqui o fator do arrefecimento.

* Este algoritmo baixa constantemente a temperatura com o número de iterações, proporcionado obter uma temperatura muito baixa e que a escolha de estados de pior avaliação quase nunca aconteça, tornando o algoritmo “quase” como o Trepa-Colinas.

## Algoritmo Recristalização Simulada

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NUM ELEMENTOS E SUBCONJUNTOS |  | Iterações = 1000 Tmax = 100 Tmin = 5 | Iterações = 1000 Tmax = 10 Tmin = 0.5 | Iterações = 1000 Tmax = 1 Tmin = 0.05 | Iterações = 10000 Tmax = 10 Tmin = 0.5 | Iterações = 10000 Tmax = 100 Tmin = 0.01 | Iterações = 10000 Tmax = 0.99 Tmin = 0.0001 |
| 10,2: n010.txt | **Melhor** | **1228** | **1228** | **1228** | **1228** | **1228** | **1228** |
| **MBF** | **1228** | **1228** | **1228** | **1228** | **1228** | **1228** |
| 12,4: n012.txt | **Melhor** | **1000** | **1000** | **1000** | **1000** | **1000** | **1000** |
| **MBF** | **935,1** | **931.3** | **935,6** | **988.9** | **1000** | **1000** |
| 30,5: n030.txt | **Melhor** | **4502** | **4402** | **4414** | **4487** | **4623** | **4617** |
| **MBF** | **41178,2** | **4179,1** | **4186.8** | **4335.8** | **4490.9** | **4462,1** |
| 60,6: n060.txt | **Melhor** | **15182** | **14917** | **15383** | **15281** | **15522** | **15418** |
| **MBF** | **14640,2** | **14574,3** | **14636,0** | **14984.2** | **15193.1** | **15177,6** |
| 120,10: n120.txt | **Melhor** | **35710** | **35456** | **35313** | **35643** | **35972** | **366177** |
| **MBF** | **34489,1** | **345553,3** | **34456,7** | **35019,0** | **35476,3** | **35514.7** |
| 240,12: n240.txt | **Melhor** | **118270** | **118959** | **119136** | **119983** | **120260** | **120704** |
| **MBF** | **117158,6** | **117146,3** | **117242,9** | **118416,2** | **119214,5** | **119201.8** |

### Variáveis da experiência em análise:

* Número de iterações entre 1000 e 10000;
* Variação da temperatura máxima entre 0.99 e 100;
* Variação da temperatura mínima entre 0.0001 e 5;
* Realização de 50 testes para cada ficheiro.

### Análise do efeito da variação de parâmetros:

Devido ao aumento da temperatura máxima ser proporcional à diminuição da temperatura mínima, podemos verificar que os resultados acabam por ser bastantes semelhantes uns dos outros, no entanto, é visível que uma temperatura mais baixa e o aumento de iterações contribui para uma maior probabilidade de chegar ao ótimo global.

### Análise do impacto das componentes do algoritmo:

Relativamente ao algoritmo em si, comparativamente com os outros, é bastante visível que o algoritmo tem resultados piores, devido ao facto de quando encontra um máximo (podendo ser máximo local), o algoritmo prossegue “durante algum tempo”, aceitando soluções piores, com perspetiva de ultrapassar um ótimo local, no entanto, no final de contas, acaba por se tornar pouco eficaz na sua procura pelo ótimo global.

**Pesquisa Tabu –** o algoritmo local Tabu é um algoritmo que visa a exploração de novas zonas do espaço de procurar evitando dessa forma a entrada em ciclos. Dessa forma, verifica-se que a obtenção de uma solução ótima é calculada de forma rápida, utilizando um tempo computacional satisfatório.

O seu funcionamento consiste na procura de possíveis soluções, partindo de uma solução inicial. A cada iteração tenta avançar para outra solução na sua vizinhança que satisfaça um determinado parâmetro de paragem.

Assim, caso seja uma solução válida marca essa solução numa lista de soluções visitadas denominada lista tabu, evitando assim que seja visitada mais tarde. Desse modo, apresenta um comportamento determinista escolhendo sempre o melhor vizinho.

## Algoritmo Tabu

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| NUM ELEMENTOS E SUBCONJUNTOS |  | 1000 iterações Tabu Descidas=5 | 5000 iterações Tabu Descidas = 5 | 1000 iterações Tabu Descidas=10 | 5000 iterações Tabu Descidas=10 | 1000 iterações Tabu Descidas=1 | 5000 iterações Tabu Descidas=1 |
| 10,2: n010.txt | **Melhor** | **1228** | **1228** | **1228** | **1228** | **1228** | **1228** |
|  | **MBF** | **1228** | **1228** | **1228** | **1228** | **1228** | **1228** |
| 12,4: n012.txt | **Melhor** | **1000** | **1000** | **1000** | **1000** | **1000** | **1000** |
|  | **MBF** | **993.2** | **985.2** | **986,6** | **988.2** | **987.7** | **990.7** |
| 30,5: n030.txt | **Melhor** | **5194** | **5194** | **5194** | **5194** | **5194** | **5194** |
|  | **MBF** | **5013.4** | **5080.6** | **4998.6** | **5133,8** | **4998.6** | **5077.6** |
| 60,6: n060.txt | **Melhor** | **17938** | **18451** | **18185** | **18955** | **18029** | **18447** |
|  | **MBF** | **17587,2** | **18135,9** | **17679.0** | **18528.1** | **17565.0** | **18119,1** |
| 120,10: n120.txt | **Melhor** | **42631** | **44303** | **42917** | **44703** | **42466** | **44186** |
|  | **MBF** | **41886.9** | **43726,3** | **42173.7** | **44158.5** | **41612.8** | **433680.8** |
| 240,12: n240.txt | **Melhor** | **138668** | **144503** | **139056** | **144676** | **136487** | **143349** |
|  | **MBF** | **1368833,7** | **142780,0** | **137986.4** | **143803.3** | **135128.5** | **142143.7** |

### Variáveis da experiência em análise:

* Número de iterações entre 1000 e 5000;
* Variação do parâmetro tabu descidas entre 1 e 10;
* Realização de 30 testes para cada ficheiro.

### Análise do efeito da variação de parâmetros:

Através da tabela de resultados verifica-se que para um maior número de iterações e de descidas o algoritmo é mais preciso obtendo melhores soluções. É preciso ter em conta também que o número de descidas não pode ser elevado demais senão o algoritmo acaba por ficar sem soluções possíveis.

### Análise do impacto das componentes dos algoritmos:

Estes mesmos resultados conseguem ser bastante semelhantes aos anteriores devido ao facto de o algoritmo guardar uma lista de soluções já antes visitadas, como já mencionado anteriormente, possibilitando desta forma, o algoritmo não ficar preso em ótimos locais, e, em simultâneo poder aceitar soluções piores da lista de vizinhos para fugir destes mesmo máximos locais, a junção destas qualidades num só algoritmo é essencial para a eficácia dele.

# Algoritmo Evolutivo

Os algoritmos evolutivos são algoritmos que apresentam mecanismos de evolução biológica, visto que permitem determinar a quantidade de recursos necessária para resolver “problemas” de sobrevivência. Assim têm como base a teoria da Evolução de Darwin, uma vez que contam com processos de **recombinação** e **mutação**.

A **recombinação** ou **crossover** são operadores genéticos usados para variar o processo de programação de cromossomas de uma geração para a próxima, ou seja, alteram a composição genética dos filhos durante a reprodução. Os operadores trabalham sobre a codificação de possíveis soluções(genótipo) e não sobre as soluções(fenótipos) propriamente ditas.

A **recombinação** de pais pode ser considerada uma operação de custo computacional alta para o problema de diversidade máxima de grupo uma vez que é possível calcular a diferença da qualidade entre duas soluções sem te que que necessariamente calcular a qualidade destas duas soluções, ou seja, a recombinação entre dois pais acontece com uma determinada probabilidade na descendência de um filho.

A **mutação** é um operador que após ser aplicado a um genótipo gera um filho. Similar a recombinação, a mutação é um operador sempre estocástico: o seu resultado filho depende dos resultados de uma série de escolhas aleatórias.   
Assim, a variação de parâmetros influência o comportamento dos algoritmos genéticos, para que se possa conforme as necessidades do problema e recursos disponíveis estabelecê-los da melhor forma.

* **Tamanho da População(popsize)** – o tamanho da população afeta de forma significativa o desempenho global dos algoritmos genéticos.

Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de procura do problema. Por outro lado, uma grande população fornece uma cobertura já considerada representativa do domínio do problema. Contudo, grandes populações exigem maiores recursos computacionais.

* **Probabilidade de mutação(pm\_swap**) – uma alta probabilidade de mutação procura-se essencialmente uma maior aleatoriedade entre filhos.

Por outro lado, uma menor probabilidade previne que uma dada posição fique estagnada em determinado valor, para alem de que possibilita que se chegue em qualquer ponto do espaço de procura.

* **Probabilidade de Cruzamento(pr)** – quanto maior for a probabilidade de cruzamento, mais rapidamente novas estruturas serão introduzidas na população. Contudo, uma elevada probabilidade pode provocar a perda de estruturas de alta aptidão, ou seja, grande parte da população com boas aptidões é substituída.

Por outro lado, uma menor probabilidade de cruzamento, torna o algoritmo mais lento e com mais dificuldades de obter novas estruturas.

* **Número de gerações(numGenerations)** – O número de gerações contribui diretamente no tempo de computação necessário para o fim da pesquisa. Não influenciando diretamente a convergência dos algoritmos genéticos.
* **Tamanho do torneio(t\_size) –** um grupo de indivíduos são escolhidos aleatoriamente de uma população e o melhor deles é escolhido. Este processo é repetido até que se obtenha uma nova população.

### Variáveis da experiência em análise:

* Número de iterações 1000;
* Tamanho do torneio 5;
* Número de gerações entre 2500 e 25000;
* Tamanho da população entre 10 e 100;
* Probabilidade de cruzamento entre 0.3 e 0.7;
* Probabilidade de mutação entre 0.0 e 0.05;
* Realização de 30 testes para cada ficheiro.

### Análises gerais do efeito da variação de parâmetros em todos os algoritmos evolutivos e híbridos:

Relativamente a probabilidade de cruzamento, é possível verificar que uma probabilidade muito alta ou uma probabilidade muito baixa é o menos ideal, sendo a probabilidade mediana, de 0.5% a melhor, estas alterações divergem pouco entre si.

Comparativamente à probabilidade de mutação, podemos verificar que uma probabilidade de mutação, de 0.01%, é a melhor probabilidade a ser usada, pois aquando aumentando a mutação, as soluções acabam por se danificar devido à elevada aleatoriedade. Por outro lado, com uma probabilidade de mutação nula verifica-se que o algoritmo apresenta dificuldade em chegar ao ótimo uma vez que o algoritmo fica de certa forma limitado não existindo possibilidade de divergir para encontrar o ótimo.

As últimas 3 linhas da tabela têm como objetivo testar o impacto do número da população e do número de gerações existentes no algoritmo. Ora, podemos verificar que quanto maior o número de população maior é a performance do algoritmo e mais fácil é de chegar ao máximo ótimo do problema. No entanto, as gerações também têm um certo impacto, só que não é tão significativo, daí o teste que tem mais população ter uma melhor solução do que o teste que se foca mais no maior número de gerações. A veracidade desta afirmação pode ser comprovada através do artigo “***Large Population or Many Generations for Genetic Algorithms? Im plications in Information Retrieval”***, "by increasing the population size, the tree linking the initial population to the optimal solution can gain width and loose depth, which would make the search more balanced and increase the chances for good performance."

Através dos resultados obtidos é também possível verificar que soluções de ficheiros que apresentam um menor número de elementos e subconjuntos, chegam facilmente ao ótimo independentemente dos parâmetros a variar. Por outro lado, o contrário já não se verifica existindo uma maior dificuldade de chegar ao ótimo em ficheiros com um maior número de elementos e subconjuntos.

Através dos inúmeros testes realizados a todos os algoritmos verifica-se que quanto maior o número de runs efetuados maior consistência de resultados, uma vez que são procuradas e consequentemente equiparadas uma maior quantidade de soluções.

**Algoritmo Evolutivo Base (Mutação Binária com 1 Ponto de Corte):** a implementação deste algoritmo tem como principal objetivo a mutação aleatória de genes e recombinação dos subconjuntos com 1 ponto de corte.  
Para alem disso, este algoritmo rejeita soluções inválidas, ou seja, soluções em que o número de elementos de subconjunto esperado é diferente do número de elementos obtido.   
  
Este algoritmo conta com o processo de **mutação binária** que consiste em percorrer todos os indivíduos da população e para cada um percorrer os seus elementos do conjunto. De seguida, é gerado um número aleatório e comparado com a probabilidade de mutação(**pm\_swap).** Caso o número gerado seja menor que a probabilidade de mutação é trocado um bit aleatoriamente com uma certa probabilidade.

Para além deste processo, conta também com a **recombinação com 1 ponto de corte**, em que é percorrida toda a população e seguidamente se um número aleatoriamente sorteado for menor que a probabilidade de cruzamento cria um ponto de corte e faz o devido corte nesse ponto. Este corte consiste em criar o cromossoma do filho tendo em conta metade dos genes dos seus pais, ou seja, metade dos genes do pai e metade dos genes da mãe. Os filhos gerados vão ter ainda bastantes semelhanças com os seus pais.  
  
O algoritmo evolutivo base implementado tem em conta a variação de alguns dos parâmetros acima referidos como é possível observar no quadro seguinte:

## Algoritmo Evolutivo: Mutação binária + Recombinação de 1 ponto de corte

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo: Mutação binária + Recombinação de 1 ponto de corte** | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 1000,0 | 890.7 | 4265 | 4202.1 | 15198 | 149918.9 | 34519 | 34473.5 | 118659 | 117360.7 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 1000,0 | 872.7 | 4466 | 4221.8 | 15408 | 14539.5 | 35113 | 34751.5 | 118903 | 117354.2 |
| pm = 0.01 | pr = 0.7 | 1228,0 | 1228,0 | 961 | 886.7 | 4280 | 4161.7 | 15185 | 14792.5 | 34682 | 34302.3 | 118828 | 117291.6 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1226.5 | 887 | 856.2 | 4183 | 4047.8 | 14575 | 14330.7 | 34989 | 34296.5 | 118145 | 117151.9 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228.0 | 984 | 899.5 | 4275 | 4166.2 | 15051 | 14858.5 | 35139 | 34850.3 | 118334 | 117192.8 |
| t\_size=5 | pm=0.01 | 1228,0 | 1228,0 | 961 | 886.7 | 4280 | 4161.7 | 15185 | 14792.5 | 35103 | 34702.3 | 118828 | 117291.6 |
| pr = 0.7 | pm = 0.05 | 1228 | 1228 | 1000 | 903.9 | 4149 | 4076.1 | 14766 | 14491.5 | 34387 | 34235.3 | 118069 | 116986.7 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228 | 1228 | 945 | 807.5 | 4366 | 4105.6 | 14525 | 13980 | 34703 | 33896.0 | 116817 | 115903.6 |
| pm = melhor valor obtido=0.001 | pop = 50 (ger = 5K) | 1228 | 1228 | 967 | 866.9 | 4397 | 4155.5 | 15153 | 14884.5 | 34971 | 34299.1 | 117870 | 116978.2 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228 | 1228 | 961 | 864.7 | 4324 | 4162.2 | 15147 | 14639.4 | 34761 | 34351.1 | 117442 | 117053.6 |

### Análise do impacto das componentes do algoritmo:

No algoritmo base (mutação binária com recombinação com 1 ponto de corte) é visível a inferioridade perante os outros algoritmos evolutivos devido a pouca diversidade que ele apresenta. Com apenas um ponto de corte, todos os descendentes vão acabar por ser bastante semelhantes ao seu “*pai*” e à sua “*mãe*”, limitando assim a criação de novas soluções em que possibilitam melhorias na solução ótima.

**Algoritmo Evolutivo (Mutação Binária com 1 Ponto de Corte) com penalização:** este algoritmo tem como principal diferença do algoritmo anterior a penalização de soluções inválidas. Assim, este algoritmo penaliza soluções que apresentem soluções em que o número de elementos de subconjunto esperado é diferente do número de elementos obtido. Dessa forma, são penalizadas soluções deste tipo com o a expressão:

O algoritmo tem em conta a variação de alguns dos parâmetros acima referidos como é possível observar no quadro seguinte:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo: Mutação binária + Recombinação de 1 ponto de corte + Penalização** | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 1000,0 | 890.7 | 4245 | 4152.4 | 15234 | 14613.5 | 34412 | 34132.6 | 118843 | 116411.3 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 1000,0 | 892.5 | 4456 | 4321.3 | 15423 | 14239.5 | 35198 | 34521.1 | 119014 | 117432.1 |
| pm = 0.01 | pr = 0.7 | 1228,0 | 1228,0 | 963 | 894.3 | 4321 | 4231.1 | 15185 | 14483.2 | 34682 | 34305.1 | 118953 | 116991.3 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1226.5 | 900 | 866 | 4153 | 4027.5 | 14632 | 14123.3 | 34989 | 34053.3 | 118232 | 117004.8 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228.0 | 970 | 902 | 4256 | 4146.4 | 15103 | 14348.2 | 35139 | 34341.2 | 118415 | 117011.3 |
| t\_size=5 | pm=0.01 | 1228,0 | 1228,0 | 961 | 886.7 | 4293 | 4043.2 | 15243 | 14432.3 | 35090 | 34302.3 | 118699 | 117041.9 |
| pr = 0.7 | pm = 0.05 | 1228 | 1228 | 1000 | 903.9 | 4132 | 4025.3 | 14766 | 14131.2 | 34445 | 34031 | 118123 | 116786.3 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228 | 1228 | 955 | 837.1 | 4376 | 4105.6 | 14655 | 13780 | 34712 | 33496.0 | 117654 | 115303.9 |
| pm = melhor valor obtido=0.01 | pop = 50 (ger = 5K) | 1228 | 1228 | 981 | 876.1 | 4415 | 4032.3 | 15235 | 14584.5 | 34892 | 34093.4 | 117732 | 116743.1 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228 | 1228 | 971 | 869 | 4374 | 4085.5 | 15111 | 14543.1 | 34861 | 33991.4 | 117621 | 116934.1 |

## Algoritmo Evolutivo: Mutação binária + Recombinação de 1 ponto de corte + Penalização

### Análise do impacto das componentes do algoritmo:

Através dos resultados obtidos verifica-se, existem bastantes semelhanças entre o algoritmo com e sem penalização, no entanto, a média de soluções (MBF) é bastante baixa em prol da eventual penalização aplicada. As soluções inválidas continuam a ser aceites, no entanto nunca vão ser as melhores devido à penalização aplicada, porém continuam a contribuir para a média. As melhores soluções continuam a ser atingidas uma vez que as soluções válidas não contam com essa penalização.

**Algoritmo Evolutivo (Mutação binária com recombinação Uniforme):**  este algoritmo tem como principal foco a recombinação uniforme dos subconjuntos tal como a mutação aleatória de genes.

O funcionamento do **algoritmo de recombinação** **uniforme** passa pela criação da sua “Crossover Mask” e de seguida, a criação dos seus descendentes é decidida conforme cada bit da “Crossover Mask”, caso o *1º* bit da “Crossover Mask” seja **‘1’**, vai ao progenitor “pai” buscar o seu *1º* bit, caso o *2º* bit seja **‘0’** vai buscar o *2º* bit ao progenitor “mãe”, e vise versa.

## Algoritmo Evolutivo: Mutação binária + Recombinação Uniforme

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo mutação binaria +recombinação uniforme** | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 1000,0 | 907,7 | 4378 | 4243,3 | 15634 | 15070,3 | 36305 | 35246 | 119111 | 118017,1 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 1000 | 901,5 | 4356 | 4218,9 | 15724 | 14871,8 | 35628 | 35366 | 119355 | 118420,4 |
| pm = 0.01 t\_size=5 | pr = 0.7 | 1228,0 | 1228,0 | 1000,0 | 897,0 | 4375 | 4199,2 | 15566 | 14966,1 | 35671 | 35282 | 118850 | 118028,5 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1228,0 | 967 | 852,3 | 4111 | 4057,7 | 14725 | 14480,8 | 35028 | 34325 | 118924 | 118294,5 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228,0 | 956 | 869,8 | 4286 | 4152 | 15248 | 14801,2 | 35197 | 34907 | 118772 | 118391,2 |
| t\_size=5 | pm=0.01 | 1228,0 | 1228,0 | 1000,0 | 897,0 | 4375 | 4199,2 | 15566 | 14966,1 | 35671 | 34682 | 119100 | 118028,5 |
| pr = 0.7 | pm = 0.05 | 1228,0 | 1228,0 | 1000 | 915,2 | 4119 | 4040,2 | 14522,3 | 14352,1 | 35221 | 34770 | 118959 | 118399,8 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228,0 | 1228,0 | 918 | 776,4 | 4215 | 4069,9 | 14341 | 14019,8 | 35109 | 34558 | 117732 | 116935,1 |
| pm = melhor valor obtido = 0.001 | pop = 50 (ger = 5K) | 1228,0 | 1228,0 | 1000 | 870 | 4247 | 4093,5 | 15043 | 14811,7 | 35660 | 35130.2 | 118102 | 117874,2 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228,0 | 1228,0 | 1000 | 891 | 4367 | 4235,1 | 15258 | 15019 | 35247 | 34758 | 118500 | 118050 |

### Análise do impacto das componentes do algoritmo:

Através do quadro acima evidenciado verifica-se que com uma recombinação uniforme obtém-se ligeiramente melhores resultados aquando comparado com a recombinação com 2 pontos de corte. Assim, verifica-se uma ligeira melhoria a nível de soluções obtidas, devido a uma maior aleatoriedade na criação dos seus descentes relativamente aos seus progenitores, aumentando assim ligeiramente a probabilidade de obter uma solução ótima.

**Algoritmo Evolutivo (Mutação binária com 2 pontos de corte):** a implementação deste algoritmo tem como principal objetivo a mutação aleatória de genes a e recombinação dos subconjuntos com 2 pontos de corte.

Relativamente ao algoritmo com 1 ponto de corte, pressupõe-se uma melhoria uma vez que existe uma **recombinação** mais elaborada com **2 pontos de corte**, em que o cromossoma filho recebe genes dos seus pais alternadamente. Assim, invés de receber a mesma quantidade de genes de cada parente, recebe genes do pai, seguidamente genes da mãe e por fim novamente genes do pai. Este processo pode também ser realizado de forma inversa sendo os genes iniciais do cromossoma filho os genes do seu pai. Os filhos gerados com 2 pontos de corte em vez de apenas 1, vão garantir uma aleatoriedade um pouco maior consequentemente gerando melhores soluções.  
Este algoritmo evolutivo implementado tem em conta a variação de alguns dos parâmetros acima referidos como é possível observar no quadro seguinte:

## Algoritmo Evolutivo: Mutação binária + Recombinação com 2 pontos de corte

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo: mutação binaria + recombinação com 2 pontos de corte** | | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 954,0 | 885.2 | 4402 | 4229.0 | 15295 | 14930.9 | 34810 | 34358.3 | 118586 | 117908.5 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 957,0 | 893.5 | 4470 | 4258,5 | 15514 | 14962,6 | 34922 | 34470 | 118990 | 117801 |
| pm = 0.01 t\_size=5 | pr = 0.7 | 1228,0 | 1228,0 | 1000 | 909 | 4417 | 4205 | 14998 | 14728,4 | 34889 | 34372 | 118774 | 117871,1 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1228,0 | 984 | 874 | 4077 | 4034,8 | 14582 | 14356,4 | 34521 | 34238 | 118153 | 117340,5 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228,0 | 1000 | 911 | 4313 | 4191.7 | 15094 | 14747.2 | 34558 | 34268 | 118712 | 118068.1 |
| t\_size=5 | pm=0.01 | 1228,0 | 1228,0 | 1000 | 909 | 4417 | 4205 | 15298 | 14928,4 | 34889 | 34372 | 118974 | 118471,1 |
| pr = 0.7 | pm = 0.05 | 1228 | 1228 | 1000 | 910,9 | 4117 | 4035.6 | 14665 | 14409,8 | 34717 | 34345 | 118382 | 117428.2 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228 | 1228 | 1000 | 819.0 | 4280 | 4026 | 14584 | 14060.59 | 34003 | 33490.8 | 117469 | 116072.2 |
| pm = melhor valor obtido = 0.01 | pop = 50 (ger = 5K) | 1228 | 1228 | 967 | 882.2 | 4255 | 4113.7 | 14981 | 14508.5 | 34631 | 34085.5 | 118957 | 117846.7 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228 | 1228 | 1000 | 892.0 | 4459 | 4262.20 | 15128 | 14956.5 | 34959 | 34473.8 | 118364 | 11753.3 |

### Análise do impacto das componentes do algoritmo:

Este algoritmo quando comparado com o algoritmo base (mutação binária + recombinação com 1 pontos de corte) atinge uma solução ótima melhor, em prol da sua recombinação ser tripartida com a alternância de genes dos seus pais. Ora, o algoritmo desta forma apresenta uma maior variedade de genes no cromossoma filho, aumentando a possibilidade de encontrar a solução ótima, como se verifica na tabela de resultados uma ligeira melhoria em todos os ficheiros, mas principalmente em ficheiros com maior número de elementos e subconjuntos.

**Algoritmo Evolutivo (Mutação por troca com 2 pontos de corte):** o foco principal deste algoritmo é a recombinação com dois pontos de troca e de seguida a implementação da mutação por troca. A diferença entre este algoritmo e o anterior é o tipo de mutação que é efetuada. Desse modo, a sua mutação passa a ser realizada por troca, ou seja, troca os valores dos elementos do conjunto por outros do mesmo conjunto, garantindo também que não faz a substituição por valores iguais.

## Algoritmo Evolutivo: Mutação por troca + Recombinação com 2 pontos de corte

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo Base mutação por troca + recombinação com 2 pontos de corte** | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 1000,0 | 992.7 | 5194 | 5061 | 18324 | 17872,7 | 43662 | 42825 | 139947 | 138487.2 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 1000,0 | 998,1 | 5194 | 5084 | 18315 | 17956.9 | 43456 | 42673.1 | 139967 | 138643,7 |
| pm = 0.01 t\_size=5 | pr = 0.7 | 1228,0 | 1228,0 | 1000 | 991.1 | 5194 | 5079.7 | 18295 | 17921.9 | 43005 | 42542.9 | 139223 | 138145.1 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1228,0 | 954 | 869.5 | 4236 | 4079.9 | 15000 | 14435.9 | 35160 | 34381.1 | 118436 | 117335.7 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228,0 | 1000 | 988.2 | 5049 | 4803.2 | 17100 | 16696.4 | 39799 | 38661 | 127913 | 126738.4 |
| t\_size=5 | pm=0.01 | 1228,0 | 1228,0 | 1000 | 991.1 | 5194 | 5079.7 | 18295 | 17921.9 | 43005 | 42542.9 | 139223 | 138145.1 |
| pr = 0.7 | pm = 0.05 | 1228 | 1228 | 1000 | 996.7 | 5194 | 5091.3 | 18460 | 18183.6 | 44807 | 44239.8 | 145665 | 144117.8 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228 | 1228 | 1000 | 993.7 | 5156 | 5029 | 18243 | 17904 | 43298 | 42779.3 | 140138 | 138866.1 |
| pm = melhor valor obtido = 0.01 | pop = 50 (ger = 5K) | 1228 | 1228 | 1000 | 998.93 | 5194 | 5052 | 18127 | 17895.7 | 43288 | 42715.4 | 140321 | 128644.4 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228 | 1228 | 1000 | 995.2 | 5194 | 5060.16 | 18355 | 17849.5 | 43237 | 42580.8 | 140940 | 138344.6 |

### Análise do impacto das componentes do algoritmo:

Através de uma análise da tabela acima referida, pode se verificar que este algoritmo atinge melhores soluções ótimas. Ora, verifica-se que houve uma melhoria considerável, uma vez que a sua mutação passa a ser realizada por troca. Através da tabela de resultados verifica-se este algoritmo apresenta a melhores soluções para todos os ficheiros comparativamente a todos os algoritmos anteriores quando a probabilidade de mutação é mais elevada, concluindo-se assim que uma elevada probabilidade de mutação beneficia a probabilidade de chegar à solução ótima.

Desta forma, podemos concluir que a mutação por troca é bastante superior a mutação binária.

# Algoritmo Híbrido

Os algoritmos híbridos são algoritmos que apresentam a combinação de dois algoritmos, um seguidamente ao outro, comportando-se como algoritmos completos. Desta forma, este algoritmo utiliza o **algoritmo evolutivo (mutação binária + 1 ponto de corte)** numa primeira abordagem e seguidamente utiliza outro **algoritmo local** para refinar a solução.

Contudo, apesar de ser um algoritmo bastante completo que apresente resultados melhores prevê-se um esforço computacional acrescido uma vez que são realizados dois algoritmos.

### Variáveis da experiência em análise:

* Número de iterações 1000;
* Tamanho do torneio 5;
* Número de gerações entre 2500 e 25000;
* Tamanho da população entre 10 e 100;
* Probabilidade de cruzamento entre 0.3 e 0.7;
* Probabilidade de mutação entre 0.0 e 0.05;
* Realização de 30 testes para cada ficheiro.

Para tal, na realização deste algoritmo foram utilizadas as seguintes combinações de algoritmos:

## Algoritmo Híbrido: Mutação binária + Recombinação de 1 ponto de corte + Trepa Colinas

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo Híbrido + Trepa Colinas** | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 1000,0 | 998,3 | 5156 | 4984 | 18074 | 17554 | 41787 | 41357,3 | 141161 | 140389,8 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 1000,0 | 995,2 | 5087 | 4989,3 | 18341 | 18042,1 | 43663 | 43179,7 | 141166 | 140410,4 |
| pm = 0.01 t\_size=5 | pr = 0.7 | 1228,0 | 1228,0 | 1000 | 992 | 5153 | 5055,8 | 18284 | 18064,6 | 43502 | 43036.9 | 140795 | 140127,1 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1228,0 | 1000 | 989 | 5179 | 5050 | 17840 | 17467,5 | 42094 | 41520,5 | 136386 | 134698,5 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228,0 | 1000 | 987 | 5108 | 4975,3 | 18026 | 17670,7 | 42110 | 41815,3 | 137805 | 135712,7 |
| t\_size=5 | pm = 0.01 | 1228,0 | 1228,0 | 1000 | 998 | 5194 | 5093,6 | 18517 | 18296,2 | 44527 | 44258,6 | 144802 | 144333,7 |
| pr = 0.7 | pm = 0.05 | 1228,0 | 1228,0 | 1000 | 992 | 5153 | 5055,8 | 18284 | 18064,6 | 43502 | 43036.9 | 140795 | 140127,1 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228,0 | 1228,0 | 1000 | 983 | 5140 | 5046 | 18368 | 18047,1 | 43788 | 43346,1 | 141994 | 140481 |
| pm = melhor valor obtido = 0.01 | pop = 50 (ger = 5K) | 1228,0 | 1228,0 | 1000 | 995 | 5156 | 5081,9 | 19168 | 18024,9 | 43581 | 43211,3 | 141152 | 140499 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228,0 | 1228,0 | 1000 | 995 | 5194 | 5034,3 | 18219 | 18052,3 | 43320 | 43035 | 141175 | 140009,6 |

### Análise do impacto das componentes do algoritmo:

Através do quadro de resultados verifica-se que o algoritmo híbrido apresenta melhores soluções aquando comparado com o algoritmo evolutivo base e o algoritmo local trepa colinas simples. Ora, neste algoritmo é realizado o algoritmo evolutivo e seguidamente a melhor solução é refinada com o algoritmo local trepa colinas, apresentando como espectável uma melhor solução ótima.

## Algoritmo Híbrido: Mutação binária + Recombinação de 1 ponto de corte + Trepa Colinas

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo Hibrido + Trepa Colinas 2 vizinhanças** | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 1000,0 | 996,8 | 5173 | 5061,8 | 18527 | 18182,8 | 43643 | 43303,2 | 141847 | 141275,3 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 1000,0 | 996,0 | 5194 | 5064,1 | 18410 | 18059 | 43734 | 43343,7 | 143654 | 141544,2 |
| pm = 0.01 t\_size=5 | pr = 0.7 | 1228,0 | 1228,0 | 1000 | 993,6 | 5194 | 5107 | 18128 | 17961,4 | 43745 | 43296 | 141807 | 141185,7 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1228,0 | 1000 | 971 | 5194 | 5080,7 | 18063 | 17726 | 43373 | 42579,8 | 138972 | 138058,2 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228,0 | 1000 | 1000 | 5194 | 5048 | 18288 | 17897 | 43238 | 42570,2 | 140019 | 138774,9 |
| t\_size=5 | pm = 0.01 | 1228,0 | 1228,0 | 1000 | 998,4 | 5136 | 5025,6 | 18525 | 18235 | 44542 | 44257,9 | 145877 | 144868,3 |
| pr = 0.7 | pm = 0.05 | 1228,0 | 1228,0 | 1000 | 993,6 | 5194 | 5107 | 18128 | 17961,4 | 43745 | 43296 | 141807 | 141185,7 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228,0 | 1228,0 | 1000 | 1000 | 5194 | 5055,7 | 18554 | 18344,4 | 44765 | 44477,5 | 143766 | 142847 |
| pm = melhor valor obtido = 0.01 | pop = 50 (ger = 5K) | 1228,0 | 1228,0 | 1000 | 1000 | 5153 | 5037,7 | 18547 | 18252,2 | 44457 | 44282,2 | 144238 | 143383,9 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228,0 | 1228,0 | 1000 | 993,6 | 5194 | 5107 | 18128 | 17961,4 | 43745 | 43296 | 141807 | 141185,7 |

### Análise do impacto das componentes do algoritmo:

Através do quadro de resultados verifica-se que o algoritmo híbrido apresenta melhores soluções aquando comparado com o algoritmo evolutivo base e o algoritmo local trepa colinas 2 vizinhanças. Ora, neste algoritmo é realizado o algoritmo evolutivo e seguidamente a melhor solução é refinada com o algoritmo local trepa colinas com 2 vizinhanças, apresentando como espectável uma melhor solução ótima.

## Algoritmo Híbrido: Mutação binária + Recombinação de 1 ponto de corte + Tabu

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Algoritmo Híbrido + Tabu** | | | | | | | | | | | | |
|  |  | **n010.txt** | | **n012.txt** | | **n030.txt** | | **n060.txt** | | **n120.txt** | | **n240.txt** | |
| Parâmetros Fixos | Parâmetros a variar | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF | Best | MBF |
| ger = 2500 | pr = 0.3 | 1228,0 | 1228,0 | 1000,0 | 992,0 | 5144 | 5012,2 | 18071 | 17736 | 42775 | 42345,6 | 139501 | 138272,7 |
| pop = 100 | pr = 0.5 | 1228,0 | 1228,0 | 1000,0 | 996,0 | 5174 | 5040 | 18015 | 17712,8 | 42484 | 42145,3 | 140172 | 138549,5 |
| pm = 0.01 t\_size=5 | pr = 0.7 | 1228,0 | 1228,0 | 1000 | 998 | 5154 | 5020,7 | 17938 | 17767,9 | 42378 | 42378,4 | 139488 | 138771,8 |
| ger = 2500 | pm = 0.0 | 1228,0 | 1228,0 | 1000 | 998 | 5133 | 5013,4 | 18257 | 17818,7 | 42866 | 42137 | 139586 | 138331,5 |
| pop = 100 | pm = 0.001 | 1228,0 | 1228,0 | 1000 | 995 | 5105 | 5020,1 | 18186 | 17717,8 | 42977 | 42450,2 | 140143 | 138355 |
| t\_size=5 | pm=0.01 | 1228,0 | 1228,0 | 1000 | 998 | 5194 | 5080,7 | 17938 | 17767,9 | 42378 | 42378,4 | 139488 | 138771,8 |
| pr = 0.7 | pm = 0.05 | 1228,0 | 1228,0 | 1000 | 998 | 5147 | 4995,1 | 18156 | 17682 | 43012 | 42289,5 | 139232 | 138593,2 |
| pr = 0.7 | pop = 10 (ger = 25K) | 1228,0 | 1228,0 | 1000 | 988 | 5135 | 4986,6 | 17963 | 17770 | 42773 | 42329,8 | 140287 | 138592 |
| pm = melhor valor obtido = 0.01 | pop = 50 (ger = 5K) | 1228,0 | 1228,0 | 1000 | 989 | 5153 | 5063,1 | 18031 | 17706 | 42657 | 42318,3 | 140294 | 138741,3 |
| t\_size=5 | pop = 100 (ger = 2.5K) | 1228,0 | 1228,0 | 1000 | 995 | 5179 | 5036,1 | 18030 | 17688,4 | 42994 | 42520 | 140432 | 138953,5 |

### Análise do impacto das componentes do algoritmo:

Concluindo, com o quadro acima obtido, conseguimos verificar o mesmo padrão já visto nos algoritmos de pesquisa local. Este algoritmo híbrido acaba por ser bastante semelhante ao algoritmo anterior, híbrido + trepa colinas com 2 vizinhanças, acabando por ser apenas ligeiramente pior, novamente, confirmando os testes anteriormente feitos na pesquisa local.

# Conclusão

Concluindo, através de todos os testes realizados ao longo deste trabalho prática, foi possível analisar o impacto que cada algoritmo tem na procura da melhor solução possível, comprovando esse mesmo impacto com experiências detalhadas para cada um.

Relativamente aos algoritmos de pesquisa local, apesar de serem algoritmos bastante simples e introdutórios, para este tipo de problema revelaram-se bastante eficazes na procura da melhor solução possível despendendo de poucos recursos computacionais para tal.

Por outro lado, os algoritmos evolutivos revelaram-se algoritmos bem mais complexos e com baixo performance a nível de tempo de execução. No entanto, foi possível chegar a soluções semelhantes ao algoritmo de pesquisa local, utilizando a mutação por troca com dois pontos de corte, revelando assim também a eficácia deste tipo de algoritmos na procura do ótimo global.

No entanto, os algoritmos híbridos aplicados neste mesmo trabalho, revelaram ser os mais eficazes na procura do ótimo global. Em contrapartida os recursos computacionais para a execução de um algoritmo desta envergadura será a junção dos outros anteriores, e pode por vezes não se justificar.

Em suma, na realização deste trabalho prático conseguimos corresponder aos objetivos propostos comprovando a eficácia de cada algoritmo com validade estatística.

# Bibliografia

VRAJITORU, Dana. *Large Population or Many Generations for Genetic Algorithms? Im plications in Information Retrieval.* Suiça, 2000.

FROTINA, Mona; GERNOWO, Rahmat; GUNAWAN, Vincencius. *Comparison of Genetic Algorithm and Hill Climbing for Shortest Path Optimization Mapping.* Indonésia, 2017.

STUART, Russel; NORVIG, Peter; *Artificial Intelligence: A Modern Approach (Pearson Series in Artifical Intelligence).* USA, 2020.

RODRIGUEZ, Francisco J.; LOZANO; GARCIA-MARTÍNEZ; GONZÁLEZ-BARRERA, Jonhatam; *An artificial bee colony algorithm for the maximally diverse grouping problema.* Spain, 2012.