Introdução à Inteligência Artificial

TP2 - Problema de Otimização

2021 - 2022

Nuno Santos
Pedro Nogueira

Licenciatura de Engenharia Informática

15 de janeiro de 2022

Índice

1 Introdução **3**

2 Algoritmo de Pesquisa Local (Trepa Colinas) **3**

2.1 Vizinhança 1 4

2.2 Funções mais relevantes 4

2.2.1 Função gera_sol_inicial 4

2.2.2 Função de calcula_fit 4

2.2.3 Função gera_vizinho 5

2.3 Formulação de Hipóteses 5

2.3.1. Hipótese 1 5

2.3.2 Hipótese 2 5

2.3.3 Hipótese 3 5

2.3.4 Hipótese 4 5

2.4 Verificação das Hipóteses (Experiência) 6

2.4.1 Hipótese 1 6

2.4.2 Hipótese 2 6

2.4.3 Hipótese 3 7

2.4.4 Hipótese 4 7

2.5 Conclusão 8

2.5.1 Hipóteses 1 | 2 | 3 | 4 | 5 | 6 8

3 Algoritmo Genético **9**

3.1 Recombinações 9

3.1.2 Mutacões 10

3.1.3 Torneios 10

3.2 Funções mais relevantes 11

3.2.1 Função init_pop 11

3.2.2 Função avaliaIndividual 11

3.3 Formulação das Hipóteses 11

3.3.1 Hipótese 1 | 2 | 3 | 4 | 5 | 6 11

3.4 Verificação das Hipóteses (Experiência) 12

3.4.1 Hipótese 1 12

3.4.2 Hipótese 2 12

3.4.3 Hipótese 3 13

3.4.4 Hipótese 4 13
3.4.5 Hipótese 5 14
3.4.6 Hipótese 6 14
3.5 Conclusão 15
3.51 Hipóteses 1 | 2 | 3 | 4 | 5 | 6 15

4 Algoritmo Híbrido (combinação das duas abordagens anteriores) 16
4.1 Formulação das Hipóteses 16
4.1.1 Hipóteses 1 | 2 | 3 | 4 | 5 | 6 17
4.2 Verificação das Hipóteses (Experiências) 17
4.2.1 Hipótese 1 17
4.2.2 Hipótese 2 18
4.2.3 Hipótese 3 18
4.2.4 Hipótese 4 18
4.2.5 Hipótese 5 19
4.2.6 Hipótese 6 19
4.3 Conclusões 19
4.3.1 Hipóteses 1 | 2 | 3 | 4 | 5 | 6 19

5 Conclusão 20

1 Introdução

Este trabalho consiste em conceber, implementar e testar métodos de otimização que encontrem soluções de boa qualidade para diferentes instâncias do problema a seguir descrito.

Dado um grafo não direcionado G = (V , A), composto por um conjunto V de vértices ligados entre si por arestas A, um subconjunto SV é chamado de conjunto estável quando não há nenhuma aresta entre os vértices de S. O objetivo do problema é encontrar um conjunto estável S tal que a sua cardinalidade (ou seja, o número de vértices que contém) seja máxima.

O objetivo do **problema do conjunto estável máximo** é portanto de maximização.

2 Algoritmo de Pesquisa Local (Trepa Colinas)

O método de pesquisa local implementado foi o Trepa-Colinas (também conhecido por Hill-Climbing ou "Gradient-Descent") .

O nome e ideia base provem de uma analogia com a decisão tomada por um agente que, perdido numa encosta, pretende atingir o topo, deslocar-se-á na direção "em que o caminho sabe".

A implementação deste algoritmo segue a seguinte ordem:

- Parte de um estado inicial dado ou gerado aleatoriamente;
- Gera os estados sucessores do estado atual;
- Através de uma Função de Avaliação, avalia cada estado assim gerado e escolhe o de maior valor;
- Pára quando o estado selecionado tiver um valor inferior ao escolhido na iteração anterior
    - Isto significa que a solução "piorou" e que se "está a descer a colina, em vez de subir".

Alguns problemas associados a este método são:

- Uma máximo local pode ser atingido sem que corresponda ao máximo absoluto (melhor solução)
- Nos "planaltos" é necessário escolher uma direção aleatoriamente
- Um cume pode ter lados tão inclinados que o passo seguinte conduz ao "outro lado do cume" e não ao seu topo. Neste caso a solução poderá "oscilar" nunca atingido o máximo pretendido

Será usada a variante do Trepa-Colinas "First-Choice" pois será o mais adequado ao nosso problema dado que este:

- Visita vizinhos de forma aleatória;
- Aceita um vizinho de melhor qualidade e termina a iteração, este caso é útil quando a vizinhança é grande mas também sabem que é um algoritmo não determinista.

2.1 Vizinhança 1

A primeira vizinhança testada no nosso algoritmo foi uma vizinhança bastante simples, que se baseia em aleatoriedade apenas.

Esta funciona da seguinte forma: escolhe, aleatoriamente, 2 subconjuntos diferentes A e B, e troca, de A para B e de B para A, um elemento, também aleatório.

Nos testes efetuados ao algoritmo fizemos variar o número de vizinhos, fizemos variações entre aceitar soluções de custo igual e não aceitar e também penalização.

Em todos os testes realizados neste trabalho foram usadas 10 repetições de modo a obter equilíbrio estatístico.

Os resultados foram avaliados em comparação com valores base obtidos com os seguintes parâmetros:

- Nvizinhos;
- Nrepetições;
- Aceitar soluções de custo igual;
- Penalização.

2.2 Funções mais relevantes

2.2.1 Função gera_sol_inicial

Usámos um array dinâmico que é responsável por guardar a nossa matriz inicial.
É gerado um número aleatório entre 2 e o número de vértices que irá. É gerado também um número aleatório entre 0 e o número de vértices -1, irá ficar dentro de um ciclo onde só sairá quando a solução for igual a 1.

2.2.2 Função de calcula_fit

Esta função recebe a solução, a matriz e o número de vértices e devolve o custo e o número de ligações (cardinalidade) que existem na solução.

2.2.3 Função gera_vizinho

É gerado um pos aleatório entre 0 e n-1 que é de seguida guardado na variável pos_ant. O pos é continuamente gerado enquanto for igual ao pos_ant garantido que ambos sejam índices diferentes. Quando ambos são diferentes, é feita uma substituição dos valores dos respetivos índices, gerando assim um novo vizinho.

2.3 Formulação de Hipóteses

2.3.1. Hipótese 1

H1 : Trepa-Colinas com vizinhança 1.

2.3.2 Hipótese 2

H2 : Trepa-Colinas com vizinhança 1 e aceitando soluções de custo igual.

2.3.3 Hipótese 3

H3 : Trepa-Colinas com vizinhança 1, aceitando soluções de custo igual e com Reparação.

2.3.4 Hipótese 4

H4 : Trepa-Colinas com vizinhança 1 e com Reparação

2.4 Verificação das Hipóteses (Experiência)

2.4.1 Hipótese 1

Trepa-colinas com vizinhança 1 - 15 rondas						
Ficheiro		**100 iterações**	**1000 iterações**	**2500 iterações**	**5000 itrações**	**10000 iterações**
file1.txt	**Melhor**	5	5	5	5	5
	MBF	3,98	4,004	4,0268	4,0132	4,0087
file2.txt	**Melhor**	7	7	7	7	7
	MBF	6,2	6,476	6,3856	6,3632	6,334
file3.txt	**Melhor**	12	12	12	12	12
	MBF	10,62	10,5	10,424	10,4524	10,4402
file4.txt	**Melhor**	37	38	39	39	39
	MBF	30,2999	30,271	30,3324	30,2896	30,3171
file5.txt	**Melhor**	17	18	18	18	18
	MBF	15,95	16,021	16,035	16,052	16,59
file6.txt	**Melhor**	30	32	32	33	34
	MBF	23,95	24,02	24,04	24,3	24,6

2.4.2 Hipótese 2

Trepa-colinas com vizinhança 1 e aceitando soluções de custo igual - 15 rondas						
Ficheiro		**100 iterações**	**1000 iterações**	**2500 iterações**	**5000 iterações**	**10000 iterações**
file1.txt	**Melhor**	5	5	5	5	5
	MBF	4,07	4,024	4,0208	4,0192	4,0222
file2.txt	**Melhor**	7	7	7	7	7
	MBF	6,32	6,372	6,3584	6,3552	6,3564
file3.txt	**Melhor**	12	12	12	12	12
	MBF	10,38	40,456	10,4992	10,4208	10,4154
file4.txt	**Melhor**	37	38	38	38	39
	MBF	30,43	30,288	30,4388	30,32	30,3248
file5.txt	**Melhor**	18	18	18	18	18
	MBF	16,03	15,985	16,0032	15,958	15,923
file6.txt	**Melhor**	31	34	35	36	36
	MBF	24,01	24,43	24,72	25,02	25,12

2.4.3 Hipótese 3

Trepa-colinas com vizinhança 1, aceitando soluções de custo igual e com Reparação - 15 rondas						
Ficheiro		**100 iterações**	**1000 iterações**	**2500 iterações**	**5000 iterações**	**10000 iterações**
file1.txt	**Melhor**	5	5	5	5	5
	MBF	4,07	4,024	4,0208	4,0192	4,0222
file2.txt	**Melhor**	7	7	7	7	7
	MBF	6,32	6,372	6,3584	6,3552	6,3564
file3.txt	**Melhor**	12	12	12	12	12
	MBF	10,38	40,456	10,4992	10,4208	10,4154
file4.txt	**Melhor**	37	38	38	38	39
	MBF	30,43	30,288	30,4388	30,32	30,3248
file5.txt	**Melhor**	18	18	18	18	18
	MBF	16,03	15,985	16,0032	15,958	15,923
file6.txt	**Melhor**	31	34	35	36	36
	MBF	24,01	24,43	24,72	25,02	25,12

2.4.4 Hipótese 4

Trepa-colinas com vizinhança 1 e com Reparação - 15 rondas						
Ficheiro		**100 iterações**	**1000 iterações**	**2500 iterações**	**5000 itrações**	**10000 iterações**
file1.txt	**Melhor**	5	5	5	5	5
	MBF	3,98	4,004	4,0268	4,0132	4,0087
file2.txt	**Melhor**	7	7	7	7	7
	MBF	6,2	6,476	6,3856	6,3632	6,334
file3.txt	**Melhor**	12	12	12	12	12
	MBF	10,62	10,5	10,424	10,4524	10,4402
file4.txt	**Melhor**	37	38	39	39	39
	MBF	30,2999	30,271	30,3324	30,2896	30,3171
file5.txt	**Melhor**	17	18	18	18	18
	MBF	15,95	16,021	16,035	16,052	16,59
file6.txt	**Melhor**	30	32	32	33	34
	MBF	23,95	24,02	24,04	24,3	24,6

2.5 Conclusão

2.5.1 Hipóteses 1 | 2 | 3 | 4 | 5 | 6

Nos ficheiros mais pequenos (file1.txt e file2.txt) conseguimos perceber que o Trepa-Colinas lidava bastante bem com o problema e em todos os conjuntos de iterações o melhor valor foi sempre atingido.

Nos restantes ficheiros, é percetível que quanto maior for o número de iterações mais próximo o algoritmo fica de chegar ao melhor valor.

Na hipótese 2 fizemos com que o Trepa-Colinas aceitasse soluções de custo igual, de modo a percebermos se isso iria ou não influenciar nos resultados.
Comparativamente às experiências anteriores houve pouca e às vezes nenhuma diferença em relação aos ficheiros file1.txt e file2.txt.
Nas restantes experiências notou-se um aumento pouco significativo, o que faz com que possamos concluir que aceitando soluções de custo igual, não iria alterar em muito os resultados, uma vez que o objetivo é chegar ao melhor valor.

3 Algoritmo Genético

O algoritmo genético é uma técnica para resolução de problemas que necessitem de otimização. É baseado na Teoria de Evolução de Darwin e tem um forte vínculo com conceitos da biologia. Neste algoritmo ao invés de melhorarmos uma única solução, manipulamos uma população. Uma população é um conjunto de indivíduos (soluções) que por sua vez são compostos por cromossomas (no caso do nosso problema, pontos).

Os indivíduos da população vão-se alterando ao longo de várias gerações através de mecanismo genéticos como mutação e crossover. Os melhores indivíduos de cada geração, escolhidos através de torneios, são os que dão origem aos descendentes, ou seja, a próxima geração será composta por variações dos melhores indivíduos da anterior.

Nestas experiências o torneio escolhido foi sempre com o tamanho de 2.

Para fazermos este torneio, selecionamos duas soluções aleatórias e comparamos as suas fitness, a solução que tivesse maior fitness iria ser a solução escolhida para as próximas iterações.

3.1 Recombinações

Na recombinação (crossover) foi usado um ponto de corte aleatório que consiste em dividir duas soluções pai em dois e dividir ema parte para um filho e a outra parte para outro filho como mostra no exemplo em baixo. Depois da recombinação, existia também uma probabilidade de haver uma mutação, que alterava apenas um valor numa solução.

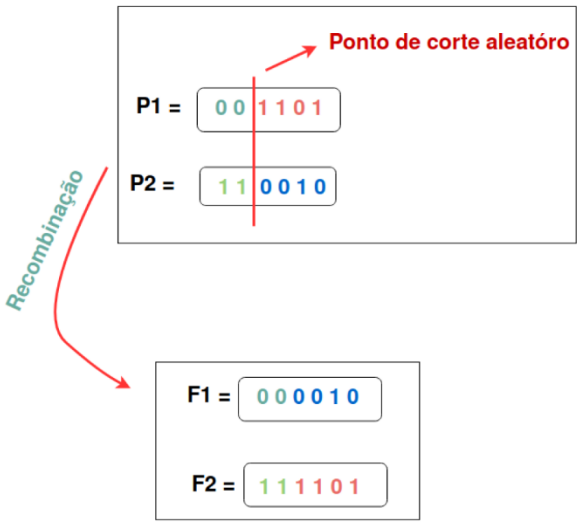

Figura : Exemplo de Recombinação

3.1.2 Mutacões

O processo de mutação consiste em alterar a unidade genética mais básica (neste caso, o cromossoma) do indivíduo. Isto possibilita a mudança subtil dos indivíduos, permitindo desenvolver soluções diferentes (piores ou melhores) que possam ajudar a guiar as próximas gerações a obter a melhor solução possível.

Os resultados foram avaliados em comparação com valores base obtidos com penalização cega, recombinação 1 e os seguintes parâmetros:

- População = 100;
- Gerações = 2500;
- Prob. Recombinação;
- Prob. Mutação.

3.1.3 Torneios

Os torneios servem como método de seleção: são usados para escolher os melhores indivíduos de uma determinada geração e determinar quais ”sobrevivem” para a próxima geração.

Um torneio consiste em escolher k indivíduos. Esses k indivíduos ”competem” entre eles, e o indivíduo com maior qualidade ”ganha” o torneio e é adicionado ao vetor de indivíduos sobreviventes para a próxima geração.

Não há qualquer restrição na repetição de indivíduos: o mesmo indivíduo pode ser adicionado várias vezes ao vetor de sobreviventes, fazendo com que os indivíduos de maior qualidade tenham um peso maior na população da nova geração.

De modo a popular a nova geração com o mesmo número de indivíduos que a anterior, estes torneiros são realizados tantas vezes quanto o tamanho da população, visto que só é escolhido um indiv´ıduo por torneio.

Um torneio onde k = 2 chama-se torneio binário. Este é o tipo de torneio que tem vindo a ser utilizado até agora, e que vai ser alterado e testado para vários valores de k (tsize).

3.2 Funções mais relevantes

3.2.1 Função init_pop

Esta função serve para criar a população inicial e preencher um vetor binário consoante os valores, a função é idêntica à gera_sol_inicial, no entanto cada solução, neste caso é inserida num array que pertence a uma estrutura.

3.2.2 Função avaliaIndividual

Nesta função usámos os mesmos princípios da **calcula_fit**, uma vez que é responsável por calcular a fitness de uma solução numa população.
Como existe a recombinação e a mutação, há uma probabilidade da solução a ser avaliada ser inválida, logo, foi preciso criar uma função chamada **verifica_validade** que tem como objetivo validar uma solução e caso esta seja válida o programa prossegue, caso contrário, é marcada como inválida e sai da **avaliaIndividual**.

3.3 Formulação das Hipóteses

Todas hipóteses abaixo efetuadas serão efetuadas para um algoritmo base sem penalização e outro com penalização.
Os parâmetros a testar serão os mesmos para todas as hipóteses (6), mas para ficheiros diferentes, podendo assim ser possível obter uma conclusão consistente e bem fundamentada.

3.3.1 Hipótese 1 | 2 | 3 | 4 | 5 | 6

- File 1 | File 2 | File 3 | File 4 | File 5 | File 6

		Algorítmo base sem penalização		Algorítmo base com penalização	
		Melhor	MBF	Melhor	MBF
Parâmetros fixos	Parâmetros a variar				
gerações = 2500	prob. recombinação = 0.3				
população = 100	prob. recombinação = 0.5				
prob. mutação = 0.01	prob. recombinação = 0.7				
gerações = 2500	prob. mutação = 0.0				
população = 100	prob. mutação = 0.001				
prob. recombinação = 0.7	prob. mutação = 0.5				
prob. recombinação = 0.7	população = 20 - gerações = 5000				
prob. recombinação = 0.7	população = 50 - gerações = 25000				

3.4 Verificação das Hipóteses (Experiência)

3.4.1 Hipótese 1

		Ficheiro file1.txt			
		Algorítmo base sem penalização		Algorítmo base com penalização	
Parâmetros fixos	Parâmetros a variar	Melhor	MBF	Melhor	MBF
gerações = 2500	prob. recombinação = 0.3	5	4,7	5	4,3
população = 100	prob. recombinação = 0.5	5	4,6	5	4,4
prob. mutação = 0.01	prob. recombinação = 0.7	5	4,7	4	4
gerações = 2500	prob. mutação = 0.0	5	3,3	5	4,7
população = 100	prob. mutação = 0.001	5	4,3	6	4,1
prob. recombinação = 0.7	prob. mutação = 0.5	5	5	4	5,3
prob. recombinação = 0.7	população = 20 - gerações = 5000	5	3	4	3
prob. recombinação = 0.7	população = 50 - gerações = 25000	6	4	5	4,1

3.4.2 Hipótese 2

		Ficheiro file2.txt			
		Algorítmo base sem penalização		Algorítmo base com penalização	
Parâmetros fixos	Parâmetros a variar	Melhor	MBF	Melhor	MBF
gerações = 2500	prob. recombinação = 0.3	7	6,7	7	6,5
população = 100	prob. recombinação = 0.5	7	6,5	6	5
prob. mutação = 0.01	prob. recombinação = 0.7	7	6,4	6	5,2
gerações = 2500	prob. mutação = 0.0	7	5,1	6	5,6
população = 100	prob. mutação = 0.001	7	6,8	7	7
prob. recombinação = 0.7	prob. mutação = 0.5	6	2,7	6	4,2
prob. recombinação = 0.7	população = 20 - gerações = 5000	6	2,3	6	3
prob. recombinação = 0.7	população = 50 - gerações = 7000	7	3	7	4,4

3.4.3 Hipótese 3

		Ficheiro file3.txt			
		Algorítmo base sem penalização		Algorítmo base com penalização	
Parâmetros fixos	Parâmetros a variar	Melhor	MBF	Melhor	MBF
gerações = 2500	prob. recombinação = 0.3	12	7	12	10,2
população = 100	prob. recombinação = 0.5	12	5,5	10	5
prob. mutação = 0.01	prob. recombinação = 0.7	9	5	8	4,6
gerações = 2500	prob. mutação = 0.0	9	5,8	8	6
população = 100	prob. mutação = 0.001	12	7,5	10	8
prob. recombinação = 0.7	prob. mutação = 0.5	4	2,3	5	2,7
prob. recombinação = 0.7	população = 20 - gerações = 5000	6	3,5	6	3,5
prob. recombinação = 0.7	população = 50 - gerações = 7000	8	4,4	7	4,5

3.4.4 Hipótese 4

		Ficheiro file4.txt			
		Algorítmo base sem penalização		Algorítmo base com penalização	
Parâmetros fixos	Parâmetros a variar	Melhor	MBF	Melhor	MBF
gerações = 2500	prob. recombinação = 0.3	38	35,2	37	31,9
população = 100	prob. recombinação = 0.5	38	32,599	36	32
prob. mutação = 0.01	prob. recombinação = 0.7	39	23,4	38	29
gerações = 2500	prob. mutação = 0.0	24	16,7	22	18,6
população = 100	prob. mutação = 0.001	37	28,9	38	30,9
prob. recombinação = 0.7	prob. mutação = 0.5	10	7,2	15	9,6
prob. recombinação = 0.7	população = 20 - gerações = 5000	19	11	19	12,6
prob. recombinação = 0.7	população = 50 - gerações = 7000	18	11,5	23	13,1

3.4.5 Hipótese 5

		Ficheiro file5.txt			
		Algorítmo base sem penalização		Algorítmo base com penalização	
Parâmetros fixos	Parâmetros a variar	Melhor	MBF	Melhor	MBF
gerações = 2500	prob. recombinação = 0.3	10	5,7	17	15,1
população = 100	prob. recombinação = 0.5	11	6,4	16	7,1
prob. mutação = 0.01	prob. recombinação = 0.7	11	6,2	15	6,2
gerações = 2500	prob. mutação = 0.0	10	6	9	6,5
população = 100	prob. mutação = 0.001	7	3,2	10	5,4
prob. recombinação = 0.7	prob. mutação = 0.5	6	3,8	7	4,2
prob. recombinação = 0.7	população = 20 - gerações = 5000	7	1,8	3	0,3
prob. recombinação = 0.7	população = 50 - gerações = 7000	10	3,9	3,8	9

3.4.6 Hipótese 6

		Ficheiro file6.txt			
		Algorítmo base sem penalização		Algorítmo base com penalização	
Parâmetros fixos	Parâmetros a variar	Melhor	MBF	Melhor	MBF
gerações = 2500	prob. recombinação = 0.3	7	2,3	6	2
população = 100	prob. recombinação = 0.5	4	1	6	2,3
prob. mutação = 0.01	prob. recombinação = 0.7	4	3,2	4	2,2
gerações = 2500	prob. mutação = 0.0	4	1,8	5	1,3
população = 100	prob. mutação = 0.001	5	1,1	5	1,7
prob. recombinação = 0.7	prob. mutação = 0.5	3	1,3	4	1,7
prob. recombinação = 0.7	população = 20 - gerações = 5000	4	0,6	4	0,4
prob. recombinação = 0.7	população = 50 - gerações = 7000	5	1,1	4	0,9

3.5 Conclusão

3.51 Hipóteses 1 | 2 | 3 | 4 | 5 | 6

Com o algoritmo base sem penalização, fixando uma população de 100, uma geração de 2500 uma probabilidade de mutação 0.01 variando apenas a probabilidade de recombinação conseguimos perceber que quanto maior é a probabilidade de recombinação mais longe o algoritmo fica de atingir a solução.
Com os mesmos valores anteriores, fixando apenas a probabilidade de recombinação e variando a probabilidade de mutação os resultados tornar-se ainda piores, porque a probabilidade de recombinação fixa é alta e a probabilidade de mutação não favorece os resultados.
Com uma população muito baixa, os resultados foram os piores, mesmo aumentando o número de gerações, o que leva a concluir que uma maior população influencia num bom resultado.

No caso do algoritmo base com penalização, uma vez que este aceita soluções inválidas, havia uma possibilidade de encontrar uma solução ótima, no entanto, isso não aconteceu o que fez com que os resultados não tenham sido muito diferentes comparativamente ao primeiro algoritmo.

4 Algoritmo Híbrido (combinação das duas abordagens anteriores)

Este algoritmo é uma combinação dos algoritmos de pesquisa local (Trepa-Colinas) e do algoritmo genético discutidos anteriormente.

Usamos o algoritmo hibrido de duas maneiras diferentes. A primeira foi usada aquando da criação da população inicial, ou seja, em vez de começarmos com uma população aleatória, melhoramos a população incial com o Trepa-Colinas e esta serviu de ponto de partida para o algoritmo evolutivo.

A segunda maneira foi deixar executar o algoritmo evolutivo primeiro e no final usar o trepa colinas na população final e vendo se era possível ainda melhorar um pouco mais a mesma.

Os resultados obtidos serão comparados com os resultados obtidos para os mesmos parâmetros e métodos no algoritmo genético.

A diferença entre este algoritmo híbrido e o algoritmo genético anteriormente falado é que este invoca, em diferentes etapas da execução, o algoritmo de pesquisa local, de modo a melhorar o indivíduo escolhido.

4.1 Formulação das Hipóteses

Todas hipóteses abaixo efetuadas serão efetuadas para o algoritmo base híbrido i) e algoritmo base híbrido ii).

Os parâmetros a testar serão os mesmos para todas as hipóteses (6), mas para ficheiros diferentes, podendo assim ser possível obter uma conclusão consistente e bem fundamentada.

4.1.1 Hipóteses 1 | 2 | 3 | 4 | 5 | 6

- File 1 | File 2 | File 3 | File 4 | File 5 | File 6

	Ficheiro – 100 iterações			
	Algoritmo base híbrido i)		Algoritmo base híbrido ii)	
Parâmetros	Melhor	MBF	Melhor	MBF
população = 100 - gerações = 2500 prob. mutação = 0 prob. recombinação = 0.3				
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2				

4.2 Verificação das Hipóteses (Experiências)

4.2.1 Hipótese 1

	Ficheiro file1.txt – 100 iterações			
	Algoritmo base híbrido i)		Algoritmo base híbrido ii)	
Parâmetros	Melhor	MBF	Melhor	MBF
população = 100 - gerações = 2500 prob. mutação = 0 prob. recombinação = 0.3	5,0	3,3	5,0	3,4
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2	5,0	3,3	5,0	3,3

4.2.2 Hipótese 2

	Ficheiro file2.txt – 100 iterações			
	Algoritmo base híbrido i)		Algoritmo base híbrido ii)	
Parâmetros	Melhor	MBF	Melhor	MBF
população = 100 - gerações = 2500 prob. mutação = 0 prob. recombinação = 0.3	5,0	2,7	5,0	2,3
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2	5,0	2,3	4,0	2,2

4.2.3 Hipótese 3

	Ficheiro file3.txt – 100 iterações			
	Algoritmo base híbrido i)		Algoritmo base híbrido ii)	
Parâmetros	Melhor	MBF	Melhor	MBF
população = 100 - gerações = 2500 prob. mutação = 0 prob. recombinação = 0.3	6,0	2,5	6,0	2,3
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2	7,0	2,5	5,0	2,4

4.2.4 Hipótese 4

	Ficheiro file4.txt – 100 iterações			
	Algoritmo base híbrido i)		Algoritmo base híbrido ii)	
Parâmetros	Melhor	MBF	Melhor	MBF
população = 100 - gerações = 2500 prob. mutação = 0 prob. recombinação = 0.3	15,0	7,5	15,0	7,8
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2	16,0	7,3	14,0	7,7

4.2.5 Hipótese 5

	Ficheiro file5.txt – 100 iterações			
	Algoritmo base híbrido i)		Algoritmo base híbrido ii)	
Parâmetros	Melhor	MBF	Melhor	MBF
população = 100 - gerações = 2500 prob. mutação = 0 prob. recombinação = 0.3	8,0	3,4	10,0	2,9
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2	11,0	3,9	11,0	3,4

4.2.6 Hipótese 6

	Ficheiro file6.txt – 100 iterações			
	Algoritmo base híbrido i)		Algoritmo base híbrido ii)	
Parâmetros	Melhor	MBF	Melhor	MBF
população = 100 - gerações = 2500 prob. mutação = 0 prob. recombinação = 0.3	6,0	1,7	7,0	1,6
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2	6,0	1,7	5,0	1,6

4.3 Conclusões

4.3.1 Hipóteses 1 | 2 | 3 | 4 | 5 | 6

É possível observar que os resultados obtidos nestas hipóteses melhoram relativamente ao algoritmo genético, era o que se esperava que acontecesse.

Repare que, á medida que o número de vértices/arestas aumenta a diferença de valores obtidos entre o algoritmo evolutivo e o algoritmo híbrido é maior, e faz sentido acontecer, visto que algoritmos híbridos são mais fiáveis em termos de resultados.

Em relação ao algoritmo base híbrido 1 e ao algoritmo base híbrido 2 não se nota resultados muito significativos o que leva a concluir que ambos foram bons comparado com o algoritmo evolutivo.

5 Conclusão

Assim, e concluindo o trabalho efetuado, depois de analisados os resultados, marcamos como principais os seguintes pontos gerais:

- Quando o algoritmo se baseia em lógica altamente aleatória, os resultados não são de todo estáveis e, de certa forma, prejudicam o estudo estatístico dos mesmos;
- Os resultados obtidos dependem largamente da criatividade e qualidade das funções de vizinhança, mutação, recombinação e reparação, sendo que há várias possibilidades para estas (com qualidade semelhante);
- Em geral, um número elevado de iterações (caso da pesquisa local) ou gerações / número de indivíduos da população aumenta a qualidade final dos resultados, mas requer mais tempo e memória para o algoritmo;
- No caso do algoritmo genético, o aumento da probabilidade da recombinação e mutação pode ter ou não efeitos benéficos ao problema, devendo ser estudada a sua variação e escolhido o melhor valor para cada função de mutação / recombinação;
- Um bom algoritmo requer, inevitavelmente, uma maior complexidade temporal para melhores resultados em problemas maiores, sendo que, para alcançar a solução ótima nestes problemas, é necessário uma grande capacidade de computação e tempo;
- Não é necessário um algoritmo com alta complexidade para resolver problemas mais simples, como os dos ficheiros **file1.txt** e **file2.txt**.

Tendo alcançado o objetivo deste trabalho, damos o mesmo por concluído.