Ano Letivo: 2023/2024

Relatório

IIA

Trabalho realizado por:

Martim Alexandre Vieira Antunes

nº: 2022141890 **Curso:** LEI

Pedro Lino Neves Faneca

nº: 2022134142 **Curso:** LEI

1.Introdução

Neste relatório, abordamos o Problema do Subconjunto de custo mínimo, cujo objetivo é encontrar um subconjunto de vértices de tamanho k, tal que os vértices possuam pelos menos uma ligação e o custo das arestas dentro do subconjunto seja mínimo. A representação usada para este problema foi a binária.

2.Desenvolvimento

Nesta seção, apresentamos o código implementado em C para resolver este problema. O programa foi estruturado em vários módulos para melhor organização do código.

2.1 Inicialização da Matriz

A matriz de adjacências é inicializada a partir de um arquivo de entrada, onde são especificados o número de vértices, arestas e o valor de k. A função init_dados realiza a leitura do arquivo, alocando dinamicamente a matriz e preenchendo-a com as informações fornecidas.

2.2 Algoritmo de Pesquisa Local - Trepa Colinas

O algoritmo de Pesquisa Local utilizado neste trabalho é conhecido como Trepa Colinas. A seguir, serão detalhadas algumas das funções mais relevantes que compõem esse algoritmo.

2.2.1 Funções Mais Relevantes

Função geraSolInicial

A função geraSolInicial é responsável por gerar uma solução inicial aleatória para este problema de minimização. A solução é representada por um conjunto de subconjuntos, e a função utiliza dois arrays: um para contagem da inserção de cada subconjunto (auxiliar) e outro para armazenar a solução final (solucao).

1. **Inicialização:** Inicia um array auxiliar de tamanho G para contar a inserção de subconjuntos e um array solucao para representar a solução.
2. **Geração Aleatória:** Gera um número aleatório entre 0 e o número de subconjuntos (G).
3. **Verificação e Inserção:** Verifica se o valor no array auxiliar, na posição do número aleatório gerado, é menor que N. Se verdadeiro, incrementa esse valor no array auxiliar e insere o número aleatório no array de solução.
4. **Condição de Término:** A função continua esse processo até que todas as posições do array auxiliar estejam preenchidas com o valor igual a N.

Função calculaFit

A função calculaFit é responsável por calcular a diversidade de uma solução representada por um conjunto de subconjuntos. Essa diversidade é determinada pela soma das distâncias entre todos os pares de elementos que pertencem ao mesmo subconjunto.

1. **Inicialização:** Inicia um array auxiliar de tamanho N para armazenar os índices das distâncias de cada subconjunto.
2. **Cálculo da Diversidade:** Utiliza o array auxiliar como índice para a matriz de distâncias, somando os valores das distâncias. O array auxiliar é limpo para receber os índices do próximo subconjunto.
3. **Condição de Término:** A função termina quando o cálculo das distâncias de todos os subconjuntos estiver concluído, retornando a variável soma, que representa a diversidade da solução.

Função geraVizinho

A função geraVizinho é responsável por gerar um vizinho a partir de uma solução dada. Esse processo envolve a troca aleatória de elementos entre dois subconjuntos, representados pelos pontos p_ponto e s_ponto.

1. **Geração Aleatória:** Gera aleatoriamente dois pontos (p_ponto e s_ponto) entre 0 e M-1. O s_ponto é continuamente gerado enquanto for igual ao p_ponto, garantindo que ambos sejam índices diferentes.
2. **Substituição:** Realiza a troca dos valores nos índices p_ponto e s_ponto, gerando assim um novo vizinho.

Essa função é crucial para explorar o espaço de soluções vizinhas e buscar soluções que maximizem a diversidade.

2.2.2 Resultados

O algoritmo de Trepa Colinas é executado um número especificado de vezes (30 por padrão), e os resultados são analisados. Para cada execução, são apresentadas a solução inicial, as soluções intermediárias e o custo final. O Melhor Benefício Médio (MBF) é calculado como a média dos custos finais.

Tabela 2.1: Resultados do Trepa Colinas - Vizinhança 1 sem Aceitar Soluções de Custo Igual

N VERTS		100 iterações	1000 iterações	2500 iterações	5000 itrações	10000 iterações
test.txt	Melhor	6	6	6	6	6
	MBF	6.733333	6.133333	6.533333	6.4	6.266667
file1.txt	Melhor	50	45	45	45	48
	MBF	76.133331	60.133335	61.066666	58.166668	57.866665
file2.txt	Melhor	20	15	14	14	8
	MBF	64.300003	25.433332	24.633333	25.9	23
file3.txt	Melhor	446	340	336	336	336
	MBF	497.566681	398.766663	378.733337	376.733337	372.933319
file4.txt	Melhor	45	15	16	9	7
	MBF	4834.833496	4170	4667.899902	4667.299805	4667.299805
file5.txt	Melhor	5000	5000	5000	5000	5000
	MBF	5000	5000	5000	5000	5000

Analisando os resultados para o Trepa Colinas com vizinhança 1 e sem aceitar soluções de custo igual, observamos que:

- Nos ficheiros menores (e.g., test.txt), o algoritmo converge rapidamente, atingindo o melhor valor em todas as iterações.
- Em ficheiros maiores, como file4.txt, percebemos que o algoritmo ainda precisa de mais iterações para se aproximar do ótimo, indicando uma convergência mais lenta.
- A Média do Melhor Fitness (MBF) tende a diminuir à medida que o número de iterações aumenta, indicando uma melhoria na qualidade das soluções encontradas.

Figura 2.2: Resultados do Trepa Colinas - Vizinhança 1 com Aceitação de Soluções de Custo Igual

N VERTS		100 iterações	1000 iterações	2500 iterações	5000 itrações	10000 iterações
test.txt	Melhor	6	6	6	6	6
	MBF	6	6	6	6	6
file1.txt	Melhor	52	45	45	48	45
	MBF	71.466667	60.099998	57.933334	59.099998	53.833332
file2.txt	Melhor	30	13	8	13	14
	MBF	66.23333	25.366667	20.9	26.766666	23.733334
file3.txt	Melhor	402	336	336	336	336
	MBF	483.899994	382.533325	370.233337	370.700012	372.466675
file4.txt	Melhor	49	10	9	8	7
	MBF	2870.633301	197.333328	15.1	11.966666	10.1
file5.txt	Melhor	5000	5000	61	43	16
	MBF	5000	5000	4508.133301	4514.666504	4337.033203

Analisando os resultados para o Trepa Colinas com vizinhança 1 e aceitando soluções de custo igual, observamos que:

- As soluções ótimas são mantidas em todas as iterações para os ficheiros menores (e.g., test.txt), como era esperado.
- A aceitação de soluções de custo igual não afeta significativamente os resultados para ficheiros menores.
- Em ficheiros maiores, como file4.txt, a aceitação de soluções de custo igual parece influenciar positivamente na exploração do espaço de soluções, levando a melhores resultados, especialmente em iterações iniciais.
- A Média do Melhor Fitness (MBF) tende a se manter constante ou melhorar ligeiramente em comparação com a abordagem sem aceitação de custos iguais. Isso sugere que a aceitação de soluções de custo igual pode contribuir para uma maior diversidade de soluções exploradas.

Figura 2.3: Resultados do Trepa Colinas - Vizinhança 2 sem Aceitação de Soluções de Custo Igual

N VERTS		100 iterações	1000 iterações	2500 iterações	5000 itrações	10000 iterações
test.txt	Melhor	6	6	6	6	6
	MBF	7	6.9	7.333333	6.933333	6.566667
file1.txt	Melhor	73	54	45	45	45
	MBF	101.166664	63.366665	56.5	53.700001	52.933334
file2.txt	Melhor	39	21	18	15	15
	MBF	82.300003	44	31.533333	25.9	22.633333
file3.txt	Melhor	456	369	358	354	336
	MBF	525.533325	433.433319	404.366669	396.399994	381.100006
file4.txt	Melhor	81	32	20	17	12
	MBF	4509.033203	2195.733398	3343.399902	2016	2011.199951
file5.txt	Melhor	5000	280	133	101	128
	MBF	5000	4685.533203	4837.766602	4836.700195	4837.600098

Analisando os resultados para o Trepa Colinas com vizinhança 2 e sem aceitação de soluções de custo igual, podemos destacar os seguintes pontos:

- O desempenho em termos de Melhor Fitness tende a melhorar com o aumento do número de iterações para a maioria dos ficheiros.
- Ficheiros maiores, como file4.txt e file5.txt, mostram uma melhoria substancial no Melhor Fitness em iterações posteriores, indicando que a vizinhança 2 é eficaz em explorar soluções melhores com mais iterações.
- A Média do Melhor Fitness (MBF) mostra uma tendência de diminuição ao longo das iterações, indicando uma convergência para soluções melhores.
- Em comparação com a vizinhança 1, a vizinhança 2 parece proporcionar melhorias mais significativas nos ficheiros maiores, explorando soluções mais diversificadas e alcançando melhores resultados.

Figura 2.4: Resultados do Trepa Colinas - Vizinhança 2 com Aceitação de Soluções de Custo Igual

N VERTS		100 iterações	1000 iterações	2500 iterações	5000 itrações	10000 iterações
test.txt	Melhor	6	6	6	6	6
	MBF	6	6	6	6	6
file1.txt	Melhor	62	48	45	45	45
	MBF	93.26667	61.700001	57.266666	51.866665	53.466667
file2.txt	Melhor	34	18	17	10	16
	MBF	86.466667	37.766666	32.866665	26.9	22.666666
file3.txt	Melhor	454	379	356	347	340
	MBF	531.06665	428.266663	404.566681	392.766663	384.933319
file4.txt	Melhor	65	29	24	16	11
	MBF	4184.266602	55.066666	34.799999	25.433332	18.866667
file5.txt	Melhor	5000	104	102	142	80
	MBF	5000	4836.799805	4674.433105	4354.600098	4345.433105

Analisando os resultados para o Trepa Colinas com vizinhança 2 e aceitação de soluções de custo igual, podemos destacar os seguintes pontos:

- Em comparação com a tabela anterior que não aceitava soluções de custo igual, observamos uma ligeira diferença no Melhor Fitness para alguns casos.
- Para o ficheiro file2.txt, a aceitação de soluções de custo igual resulta em uma solução melhor no caso de 1000 iterações.
- No geral, os resultados do Melhor Fitness e Média do Melhor Fitness (MBF) permanecem semelhantes, indicando que a aceitação de soluções de custo igual não teve um impacto significativo nos resultados.
- A vizinhança 2 continua a demonstrar melhorias, especialmente em ficheiros maiores, explorando soluções mais diversificadas.

Os resultados obtidos nas tabelas demonstram que o algoritmo Trepa Colinas, utilizando vizinhança 1 e 2, apresenta melhorias graduais no Melhor Fitness e Média do Melhor Fitness (MBF) à medida que o número de iterações aumenta. A aceitação de soluções de custo igual tem impacto mínimo nos resultados, indicando que o algoritmo é robusto mesmo quando considera soluções com custos iguais.

Conclusões

Podemos concluir que com este algoritmo que obtemos melhores resultados com a vizinhança 1 e aceitando soluções de custo igual, e que este algoritmo tem baixa qualidade em ficheiros mais complexos.

2.3 Algoritmo Evolutivo

O princípio básico do mesmo é, partir com um conjunto de soluções aleatórias para um problema, dando a este conjunto o nome de população. A partir da população selecionamos os melhores que irão ser chamados de progenitores. Estes progenitores depois podem ou não ser recombinados (probabilisticamente) gerando descendentes, descendentes estes que probabilisticamente podem ser mutados. Nestas experiências o torneio escolhido foi sempre com o tamanho de 2. Para fazermos este torneio, selecionamos duas soluções aleatórias e comparavamos as suas fitness, a solução que tivesse menor fitness iria ser a solução escolhida para as próximas iterações. Na recombinação foi usado um ponto de corte aleatório que consistia em dividir duas soluções pai em dois e dividir uma 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.

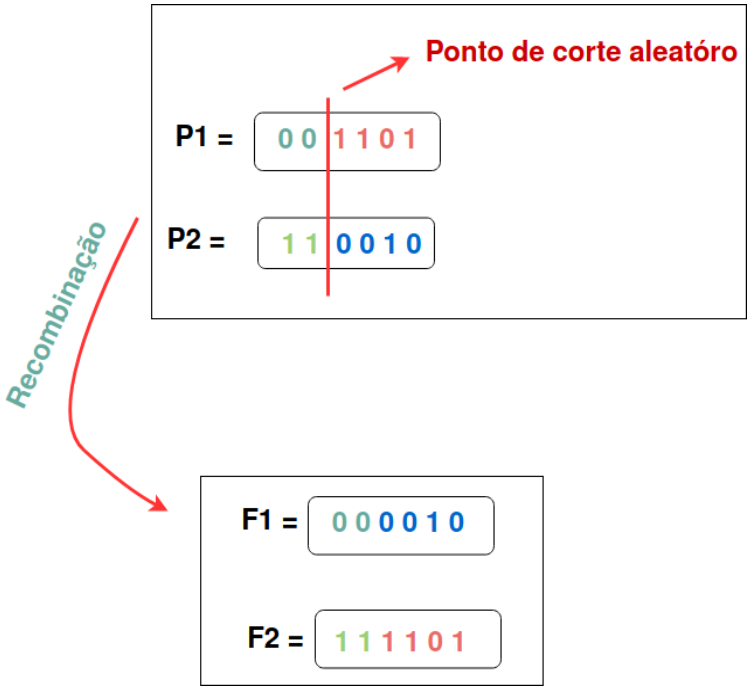

Figura 2.3: Exemplo de Recombinação

2.3.1.Funções mais relevantes

- **initPop**

A função init_pop desempenha um papel crucial em algoritmos evolutivos, sendo responsável por criar e inicializar uma população inicial de soluções. Alocamos dinamicamente memória para armazenar os indivíduos da população e utilizamos um loop para gerar soluções iniciais para cada um deles. O tamanho da população é determinado pela variável popsize e usamos uma função adicional, gera_sol_inicial, para gerar soluções iniciais válidas. O processo de inicialização da população é crucial para garantir uma diversidade inicial de soluções, proporcionando uma base sólida para o algoritmo evolutivo explorar e otimizar ao longo das iterações.

- **tournament**

A função "tournament" implementa um método de seleção chamado torneio, frequentemente utilizado em algoritmos genéticos e algoritmos evolutivos. O principal objetivo do torneio é selecionar indivíduos da população para atuarem como pais na próxima geração do algoritmo. O processo ocorre da seguinte maneira:

Para cada posição na população (popsize vezes), realiza-se um torneio.

Dois indivíduos são escolhidos aleatoriamente (x1 e x2) da população.

Os dois indivíduos participam de um torneio, onde o indivíduo com a menor aptidão (fitness) é escolhido como vencedor.

O vencedor do torneio é adicionado ao conjunto de pais (parents) que será utilizado na reprodução para gerar a próxima geração.

- **tournament_geral**

A função tournament_geral implementa um torneio geral para a seleção de pais. Aqui estão os passos principais da função:

Para cada posição na população (popsize vezes), realiza-se um torneio.

Em cada torneio, um número fixo de participantes (TSIZE) é escolhido aleatoriamente da população.

Os participantes competem entre si, e o indivíduo com a menor aptidão (fitness) é escolhido como vencedor do torneio.

O vencedor de cada torneio é adicionado ao conjunto de pais (parents) que será utilizado na reprodução para gerar a próxima geração.

- **genetic_operators**

A função **genetic_operators** aplica operadores genéticos para gerar a próxima geração de indivíduos a partir dos pais.

- **calcula_fit_reparado1(numero de k)**

A função calcula_fit_reparado1 desempenha um papel crucial na avaliação e possível reparação de soluções em um contexto de otimização. O seu propósito é avaliar a adequação de uma solução para um problema específico, onde o objetivo é encontrar um conjunto específico de vértices num grafo. A função começa a contar o número de vértices ativos na solução e, em seguida, realiza uma reparação caso esse número não corresponda ao valor desejado, ajustando-o para igualar o valor desejado k.

- **calcula_fit_reparado2**

Semelhante à anterior, mas esta faz dois reparos.

Reparo 1: Semelhante à função anterior, ajusta o tamanho do subconjunto para igualar k.

Reparo 2: Além do ajuste de tamanho, realiza uma segunda etapa de reparo. Substitui vértices sem conexões por vértices que tenham alguma ligação com o conjunto, mas que originalmente não fazem parte dele.

2.3.2.Resultados

Para realizar os testes utilizamos um método de penalização, dois métodos de reparação, dois métodos de mutação, dois métodos de recombinação e dois métodos de seleção, com diferentes valores de gerações, população, probabilidade de mutação e recombinação.

- **Test.txt**

Test.txt	
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 = 10 (gerações=25k)
prob. mutacao = 0.5	população = 50 (gerações = 5k)
	população=100 (gerações = 2.5k)

Com base nestes valores, em todos os métodos deu sempre o mesmo resultado (6), assim podemos concluir que não existe melhor solução para este ficheiro.

- **File1.txt**

File1.txt	
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 = 10 (gerações=25k)
prob. mutacao = 0.001	população = 50 (gerações = 5k)
	população=100 (gerações = 2.5k)

Neste ficheiro praticamente todos os métodos testados deram os mesmos valores, podemos concluir que como era fácil chegar à solução ótima, os métodos não mudaram muitos os resultados. Em termos de probabilidades dos operadores, obtivemos melhores resultados com pr=0.7 e pm=0.001.O melhor valor encontrado foi 45.

- **File 2.txt**

File2.txt	
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 = 10 (gerações=25k)
prob. mutacao = 0.001	população = 50 (gerações = 5k)
	população=100 (gerações = 2.5k)

Algorítmo base		Com penalização		Com reparação 1		Com reparação 2	
Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF
44	88,73333	17	29,299999	13	15	8	14,866667
52	102,366669	21	33,866665	13	15,666667	8	15,333333
60	95,76667	16	33,333332	14	17,4	14	17
40	86,833336	49	91,933334	18	35,066666	15	34,599998
28	59,266666	20	55,599998	14	21,1	8	18,433332
53	111,76667	58	114,333336	18	28,633333	18	27,666666
26	73,433334	25	71,199997	14	24,1	11	21,333334
34	70,099998	33	58,599998	14	22,166666	15	26,533333
15	51,966667	28	57,733334	14	22,266666	13	23,700001

Com recombinação 2 pontos		Com recombinação uniforme		Com mutação por troca 1		Com mutação por troca 2		Com seleção 1		Com selecao 2	
Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF
20	26,700001	18	32,366665	22	30,266666	13	23,466667	19	30,966667	13	22,333334
16	28,1	24	39,200001	17	32,633335	15	23,866667	22	33,233334	15	25,033333
19	29,4	17	35,433334	20	34,566666	15	24,966667	23	34,099998	15	22,033333
39	90,433334	35	77,73333	28	78	46	81,933334	43	83,26667	48	107,76667
29	56,200001	19	55,099998	17	37,133335	21	43,166668	23	51,433334	21	54,400002
68	115,166664	65	111,466667	78	112,866669	75	110,2	68	115,76667	45	114,166664
26	60,299999	33	66,73333	21	47	29	58,833332	40	74,833336	13	49,456667
27	58,466667	26	65,066666	21	46,366665	21	41,700001	32	66,300003	21	51,333332
23	53,5	25	58,200001	17	40,033333	20	39,066666	27	54,633335	8	54,266666

Neste ficheiro verificamos que os valores tendem a aumentar quando aumentamos as probabilidades, logo querer dizer que pr e pm quanto mais próxima de 0 melhor. O método de selecao 2 (tournement geral) foi melhor sucedido que o torneio binário.

O melhor valor que encontramos foi 8 quando usámos a reparação 2.

- **File3.txt**

File3.txt	
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 = 10 (gerações=25k)**
prob. mutacao = 0.001	**população = 50 (gerações = 5k)**
	população=100 (gerações = 2.5k)

Algorítmo base		Com penalização		Com reparação 1		Com reparação 2	
Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF
438	495,766663	369	410,899994	336	347,033325	336	347,366669
463	532,700012	379	415,433319	336	350,633331	336	353,566681
523	569,866638	381	415,733337	336	354,700012	336	356
497	575,299988	501	570,766663	343	405,333344	348	409
363	451,233337	388	502,666656	336	366,233337	336	380,366669
522	607,033325	514	596,200012	471	511,399994	436	512,133362
377	462,333344	417	575,966675	336	371,466675	336	372,200012
392	460,033325	426	555,56665	336	375,899994	336	375,299988
408	454,166656	393	509,700012	336	368,5	336	374,833344

Com recombinação 2 pontos		Com recombinação uniforme		Com mutação por troca 1		Com mutação por troca 2		Com seleção 1		Com selecao 2	
Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF
383	416,5	367	406	385	413,5	336	484,466675	380	412,133331	336	365,333344
381	418,299988	371	425,5	383	415,5	344	505,533325	384	414,866669	336	366,899994
376	410,333344	416	472	397	420,600006	360	502,233337	374	414,466675	336	371,666656
526	583,900024	455	536	527	572,333313	505	581,033325	533	581,299988	522	582,766663
394	502	403	472,166656	392	511,033325	413	541,966675	397	485,700012	336	459,700012
562	600,099976	562	603,266663	553	600,733337	438	500,866669	566	603,666687	538	602,333313
412	570,233337	436	585,166687	336	571,633362	442	587	453	563,366638	476	567,465433
398	551,099976	391	526,866638	365	536,93335	432	572,400024	408	569,93335	422	537,466675
395	534	390	462,566681	351	496,966675	397	540,400024	416	515,366638	536	600,456667

Neste ficheiro concluímos que aumentando as probabilidades os valores não são tão piores como no ficheiro anterior. O melhor valor obtido foi 336.

- **File4.txt**

File4.txt	
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 = 10 (gerações=25k)**
prob. mutacao = 0.001	**população = 50 (gerações = 5k)**
	população=100 (gerações = 2.5k)

Algorítmo base		Com penalização		Com reparação 1		Com reparação 2	
Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF
88	3698,633301	90	281,5	31	51	36	49,700001
103	3218	50	247,366669	39	48,666668	28	46,700001
85	2727,93335	72	95,900002	36	50,466667	38	50,833332
86	1418,699951	74	343,866669	25	53,466667	18	52,333332
21	1909,333374	65	283,366669	8	10,066667	8	9,7
85	3212,333252	65	390,866669	37	50,166668	26	48,433334
87	4190,100098	46	471,833344	8	10,166667	7	9,7
24	2884,833252	65	401,399994	8	10,066667	7	10,066667
31	1262,366699	78	298	7	9,966666	7	9,866667

Com recombinação 2 pontos		Com recombinação uniforme		Com mutação por troca 1		Com mutação por troca 2		Com seleção 1		Com selecao 2	
Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF
44	400,766663	60	88,5	94	422	10	382,799988	78	373,600006	8	320,899994
96	362,633331	60	74,73333	74	169,133331	16	363,633331	61	171,699997	8	369,399994
58	260,799988	54	74,566666	56	92,900002	10	319,566681	62	94,833336	7	352,866669
49	383,633331	71	242,733337	69	333,966675	78	293,200012	81	322,166656	95	382,700012
23	358,633331	57	201,96666	87	324,366669	54	286,733337	82	250,566666	22	359,266663
107	439,600006	90	442,566681	109	476,866669	26	48,933334	127	496,933319	111	393
35	442,633331	32	441,066681	30	473,899994	70	483,299988	130	487,666656	13	400,566667
22	403,933319	83	346,533325	79	424,700012	58	422,633331	122	419,266663	15	420,5
14	393,600006	55	184,100006	75	324,799988	63	309,033325	69	361,066681	21	390,299988

Neste ficheiro conseguimos chegar ao ótimo que foi 7 só mesmo, quando mudámos para o método de seleção 2 (“tournament geral”) e também como estávamos à espera com a reparação 2 mas só quando diminuímos também as probabilidades.

- **File5.txt**

File5.txt	
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 = 10 (gerações=25k)**
prob. mutacao = 0.001	**população = 50 (gerações = 5k)**
	população=100 (gerações = 2.5k)

Algorítmo base		Com penalização		Com reparação 1	
Melhor	**MBF**	**Melhor**	**MBF**	**Melhor**	**MBF**
5000	5000	5000	5000	163	448,566681
5000	5000	5000	5000	181	444,799988
5000	5000	5000	5000	157	457,200012
5000	5000	320	494	140	1390,300049
5000	5000	5000	5000	169	608,43335
5000	5000	5000	5003,600006	155	449,899994
5000	5000	5000	500	130	435,5
5000	5000	5000	500	171	426,033325
5000	5000	5000	5000	144	448,966675

Com recombinação 2 pontos		Com recombinação uniforme		Com mutação por troca 1		Com mutação por troca 2		Com seleção 1		Com selecao 2	
Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF	Melhor	MBF
280	498,600006	5000	5003,933319	5000	5002,666656	5000	5000	5000	5002,200012	11	1018,466675
5000	5004,333344	357	5000,433319	5000	5004,200012	5000	5000	5000	5000,400236	11	514,633362
5000	5004,266663	5000	5002,666656	5000	5004,202012	5000	5000	5000	5002,733337	10	348,833344
5000	5000,066681	5000	5006,600006	5000	5000	5000	5000	5000	5000	52	88,466667
5000	5002,266667	5000	5004,244455	5000	5000	5000	5000	52	79,166664	9	182,166672
5000	5004,799988	5000	5000,345567	231	494,100006	310	493,666656	5000	5001,933319	188	459,033325
5000	5000	5000	5000	5000	5000	5000	5000	5000	5000	8	343
5000	5000	5000	5007,66667	5000	5007,013367	5000	5000	5000	5004,933356	10	350,5
5000	5000,600225	183	800,356667	5000	5013,5	5000	5000,023334	76	107,299988	11	682,06665

Neste último ficheiro não conseguimos tão bons resultados como esperávamos, excepto quando usámos o método de seleção 2 (“tournament geral”).

2.4- Método Hibrido

No algoritmo híbrido, aplicou-se a geração de uma população (proveniente do algoritmo genético), e a essa população aplicou-se o trepa colinas, sendo que o resto da implementação foi proveniente do algoritmo genético. Fizemos dois métodos híbridos, um melhorava a população inicial e outro melhorava a população final.

Utilizámos 10000 iterações e colocámos a probabilidade de reprodução a 0.3 e fizemos testes a variar a probabilidade de reprodução entre 0 e 0.001 nos dois métodos híbridos. Usámos o método de seleção geral e a reparação2 porque foram os métodos que proporcionaram melhores resultados no algoritmo evolutivo

2.4.1-Resultados

- **Test.txt**

	Test.txt - 10000 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	6,0	6,0	6.0
prob. mutação = 0.001 prob. recombinação = 0.3 tsize = 2	6,0	6,0	6,0	6,0

Nesta instância, como é um problema com baixa complexidade obteve-se sempre a solução ótima em todos os casos, independentemente dos casos a variar, porque provavelmente já estaria a solução ótima no conjunto de soluções iniciais.

- **File1.txt**

	File1.txt - 10000 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	45,0	55,7	45,0	56,1
prob. mutação = 0.001 prob. recombinação = 0.3 tsize = 2	45,0	57,1	45,0	54,8

Neste ficheiro, também é um problema com baixa complexidade e então obteve-se sempre a solução ótima em todos os casos, independentemente do método escolhido e probabilidades, porque provavelmente já estaria a solução ótima no conjunto de soluções iniciais. Contudo, existem variações do MBF.

- **File2.txt**

	File2.txt - 10000 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	25,5	26,0	63,8
prob. mutação = 0.001 prob. recombinação = 0.3 tsize = 2	8,0	22,6	21,0	57,6

Neste ficheiro, podemos reparar que obtivemos melhores resultados com o método hibrido 1 e com a probabilidade de mutação igual a 0.001.

- **File3.txt**

	File3.txt - 10000 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	336,0	378,0	336,0	373,3
prob. mutação = 0.0001 prob. recombinação = 0.3 tsize = 2	336,0	382,4	336,0	378,4

Neste ficheiro, obteve-se sempre a solução ótima em todos os casos, independentemente do método escolhido e probabilidades, porque provavelmente já estaria a solução ótima no conjunto de soluções iniciais. Contudo, existem variações do MBF.

- **File4.txt**

	File4.txt - 10000 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	9,0	68,3	8,0	11,2
prob. mutação = 0.001 prob. recombinação = 0.3 tsize = 2	11,0	84,8	7,0	11,2

Conseguimos chegar à solução ótima no algoritmo hibrido 2 e com a probabilidade de mutação igual a 0.001.

- **File5.txt**

	File5.txt - 10000 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**	174,0	2359,6	13,0	1756,0
prob. mutação = 0.001 **prob. recombinação = 0.3** **tsize = 2**	210,0	2055,3	16,0	3041,2

Neste último ficheiro notou-se que o hibrido 1 resultou em soluções muito inválidas, enquanto que o hibrido 2 quase chegou à melhor solução.

Nas seis tabelas anteriores, conseguimos perceber que o algoritmo hibrido foi uma mais valia, uma vez que comparativamente ao algoritmo evolutivo registámos uma diminuição significativa o que leva a concluir que o algoritmo hibrido foi o mais eficaz dos três.

Em relação ao algoritmo base hibrido 1 e ao algoritmo base hibrido 2, excepto no file5, não se nota resultados muitos significativos o que leva a concluir que ambos foram bons comparado com o algoritmo evolutivo.

3.Conclusão

Existem algumas conclusões que podemos tirar depois de analisadas as tabelas e os resultados das mesmas.

Uma das conclusões é que o trepa colinas conseguiu apenas os melhores resultados em ficheiros pequenos, quando era deparado com um grande número de subconjuntos a sua eficácia desceu consideravelmente.

Outra conclusão bastante importante e fácil de perceber é que quanto maior era a probabilidade de recombinação, no algoritmo evolutivo, pior eram os resultados uma vez que as soluções iriam ser recombinadas mais vezes logo havia uma maior probabilidade das mesmas serem inválidas.

Reparamos que quanto menor era a população, e mesmo aumentando o número de gerações, o algoritmo era pior, concluindo que o número de populações é um fator importante para atingir soluções ótimas.

Com a mutação não existiu grandes alterações nos resultados, apenas se verificou ocasionalmente algumas descidas da qualidade muito pouco significativas.

Uma notória melhoria nas soluções foi aquando da realização do algoritmo hibrido. Este conseguiu melhorar todas as experiências realizadas anteriormente chegando muito perto dos valores ótimos.

Para que fossem atingidos os valores ótimos, usando o algoritmo hibrido seria necessário aumentar o número de iterações a realizar.