Departamento de Engenharia Informática e de Sistemas

Instituto Superior de Engenharia de Coimbra

Licenciatura em Engenharia Informática

Introdução à Inteligência Artificial 2020/2021

Trabalho Prático Nº2:
Problema de Otimização

Turma Prática Nº 3

André Lopes | 2019139754

Samuel Tavares | 2019126468

Índice

1. Introdução

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

O problema em questão é, a **Diversidade Máxima de Grupos**, cujo objetivo é encontrar uma divisão que **maximize** a diversidade dos elementos pertencentes ao mesmo conjunto.

Para isso, foi pedida a implementação de 3 algoritmos distintos e com complexidades diferentes, a fim de realizar um estudo comparativo aprofundado sobre o desempenho da otimização. Os algoritmos usados para encontrar soluções de boa qualidade para o problema descrito foram: um **algoritmo de pesquisa local**, um ***algoritmo evolutivo*** e um **algoritmo híbrido**. Algoritmos estes abordados nas aulas práticas/teóricas, e que vão ser descritos mais em pormenor ao longo do trabalho.

Por fim, analisamos os resultados obtidos para comparar valores e retirar conclusões relativas ao algoritmo.

2. Implementação

2.1. Problema de Otimização

Este problema tem como objetivo, dividir um conjunto de M elementos em G subconjuntos. Cada um destes subconjuntos deve apresentar o mesmo número de elementos(N), ou seja, **N = M/G** elementos.

O objetivo da otimização é encontrar uma divisão que **maximize** a diversidade dos elementos pertencentes ao mesmo conjunto. A diversidade de um subconjunto G1 com N elementos é igual à soma das distâncias entre todos os pares de elementos que o constituem.

$$div(G_1) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} dist(e_i, e_j)$$

Se um **conjunto G_1** possuir os elementos (**e_1,e_1,e_3**), a sua diversidade vai ser a seguinte:

$$div(G_1) = dist(e_1, e_2) + dist(e_1, e_3) + dist(e_2, e_3)$$

A qualidade de uma solução vai ser igual à soma das diversidades de todos os seus subconjuntos.

$$Qualidade(S) = \sum_{i=1}^{G} div(G_i)$$

Exemplo:

Para a Solução S1:

$div(G_1)$ = dist(3, 4) + dist(3, 5) + dist(4, 5) = 5 + 20 + 10 = 35
$div(G_2)$ = dist(0, 1) + dist(0, 2) + dist(1, 2) = 10 + 12 + 10 = 32
Qualidade(S1) = 35 + 32 = 67

Para a Solução S2:

$div(G_1)$ = dist(0, 2) + dist(0, 4) + dist(2, 4) = 12 + 10 + 4 = 26
$div(G_2)$ = dist(1, 3) + dist(1, 5) + dist(3, 5) = 15 + 5 + 10 = 30
Qualidade(S2) = 26 + 30 = 56

Nesta situação, como se trata de um problema de maximização, a solução **S1** será melhor que a solução **S2**.

2.2. Algoritmos Implementados

No início da execução, o programa vai aceder a um ficheiro de texto com informação sobre o problema a otimizar e usar essa informação. O ficheiro está estruturado com o **número de elementos** e de **subconjuntos**, assim como os **valores das distâncias** entre todos os pares de elementos que fazem parte do problema.

A função **init_dados()** é responsável por ler esses valores e guardá-los numa matriz que vai ser passada como argumento para as funções respetivas a cada algoritmo.

Para o algoritmo Trepa-Colinas, as experiências foram realizadas com 100, 1000, 2500 e 5000 iterações para averiguar os valores obtidos.

2.2.1. Algoritmo de Pesquisa Local

O algoritmo de pesquisa local utilizado na otimização deste problema foi o **Trepa-colinas ou Hill Climbing**.

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node
  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

Este algoritmo é um método probabilístico, pelo que não permite obter resultados estatisticamente válidos com apenas uma execução do algoritmo.

A eficácia do algoritmo apenas é possível com a validação desses resultados e com a repetição das configurações do algoritmo, no mínimo 10 vezes, que foi o número de repetições usado na execução nossos testes.

Ao fazer a comparação de 2 algoritmos probabilísticos, é preciso ter em conta dois resultados fundamentais de desempenho:

- **Melhor solução obtida:** representa a qualidade da melhor solução encontrada, que neste caso vai ser a situação em que se verifica uma maior soma das diversidades de todos subconjuntos de um conjunto G_x;

- **Média da melhor solução (Mean best fitness):** representa a média obtida a partir das soluções encontradas em cada uma das repetições.

Na função **trepa_colinas()**, o vetor sol vai armazena a melhor solução encontrada até ao momento e uma nova solução obtida na vizinhança é armazenada no vetor nova_sol. Caso se verifique que esta é melhor, vai ser efetuada a troca.

Assim sendo, o Trepa Colinas funciona com um gerador de valores vizinhos, compara os valores com o melhor obtido e devolve o melhor resultado para resolver o problema descrito, que neste caso, é o resultado que tiver melhor qualidade.

- **2ª vizinhança:**

```
// Vizinhança 2

// Gera o vizinho2
gera_vizinho(sol, nova_sol2, elem, subconj);

// Avalia o vizinho2
qualidade_viz2 = calcula_fit(nova_sol2, mat, elem, subconj);

if (qualidade_viz2 > qualidade_viz) { //viz2 é melhor
	if (qualidade_viz2 > qualidade) {
		substitui(sol, nova_sol2, elem);
		qualidade = qualidade_viz2;
	}
	else {		//viz é melhor
		if (qualidade_viz > qualidade) {
			substitui(sol, nova_sol, elem);
			qualidade = qualidade_viz;
		}
	}
}
```

Se a qualidade da vizinhança 2 for superior à da vizinhança 1, essa qualidade vai ser substituída no vetor nova_sol. Caso isso não ocorra, significa que a qualidade da vizinhança 1 é melhor e é essa a guardada no vetor.

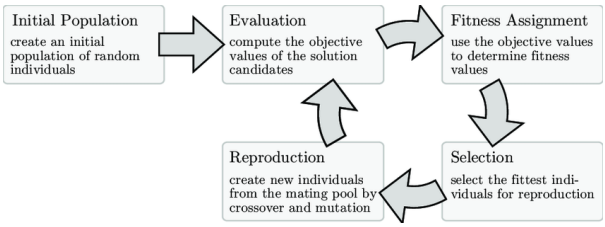
- **Trepa-colinas probabilístico:**

```
// Trepa colinas Probabilistico

if (qualidade_viz >= qualidade){ //Solucao melhor ou igual é sempre aceite
	substitui(sol, nova_sol, elem);
qualidade = qualidade_viz;
} else {	// Solucao pior tambem pode ser aceite
	if (rand_01() < PROB) { //Isto ajuda a fugir a maximos locais
		substitui(sol, nova_sol, elem, subconj);
		qualidade = qualidade_viz;
	}
}
```

O código foi alterado de modo a permitir que o Trepa-colinas **aceite soluções piores**, isto se a qualidade do vizinho for inferior à qualidade atual. Essa aceitação é baseada numa determinada probabilidade fixa onde foram usados os valores **0.01**(1%) e **0.0005**(0.05%), para averiguar alterações nos resultados. Esta função vai também ajudar a fugir a máximos locais.

2.2.2. Algoritmo Evolutivo



O algoritmo de **Computação Evolucionária** é um método probabilístico, pelo que se deve efetuar no mínimo 10 repetições da mesma configuração para poder avaliar a sua eficácia com precisão.

Existem duas medidas de desempenho essenciais:

- **Melhor solução obtida:** representa a qualidade absoluta da melhor solução encontrada;

- **Média da melhor solução (Mean best fitness):** representa a média obtida a partir das melhores soluções encontradas em cada uma das repetições.

- **Operador de Seleção:** O processo de seleção é baseado no princípio da sobrevivência dos indivíduos mais aptos, ou seja, os indivíduos com melhor aptidão possuem uma maior probabilidade de serem selecionados para reprodução;

- **Operador de Mutação:** Tem o objetivo de trazer de volta para a população os genes perdidos durante o processo de seleção de modo a que possam ser testados num novo contexto;

- **Operador de Recombinação:** Produz os dois descendentes ao escolher um ou mais pontos de corte nos cromossomas dos progenitores e depois cria uma combinação diferente das partes resultantes para gerar cada um dos cromossomas dos descendentes.

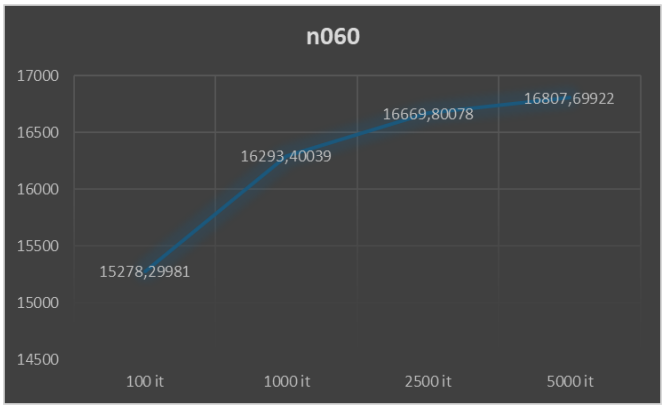
3. Análise de Resultados

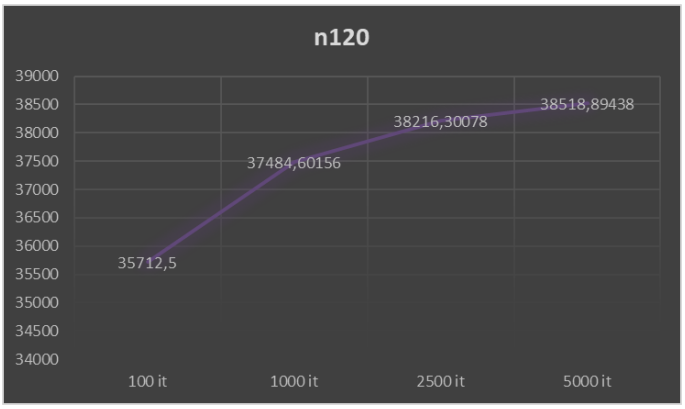
3.1. Algoritmo de Pesquisa Local

- **Trepa-colinas com 1 Vizinhança:**

Número de Elementos		*Número de Iterações*			
		100 it	**1000 it**	**2500 it**	**5000 it**
10: n010.txt	**Melhor**	1228	1228	1228	1228
	MBF	1228	1228	1228	1228
12: n012.txt	**Melhor**	1000	1000	1000	1000
	MBF	920,099976	976,400024	990,799988	992,900024
30: n030.txt	**Melhor**	4547	4787	4963	5048
	MBF	4356,399902	4688,399902	4787,899902	4801,899902
60: n060.txt	**Melhor**	15782	16776	17062	17101
	MBF	15278,29981	16293,40039	16669,80078	16807,69922
120: n120.txt	**Melhor**	36489	38213	39280	39288
	MBF	35712,5	37484,60156	38216,30078	38518,89438
240: n240.txt	**Melhor**	121139	126743	128354	129413
	MBF	119816,6016	125059,8984	126699,2031	127865

Pelos valores obtidos, é possível verificar que existe um crescente aumento nas médias obtidas a partir das soluções encontradas em cada uma das repetições, assim como no valor máximo obtido da melhor qualidade. Esta diferença é mais notável ao ler ficheiros com um maior número de elementos como os que têm 60, 120 e 240 elementos.



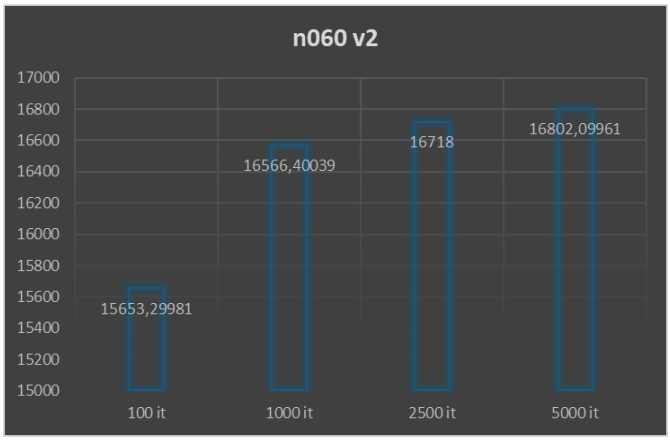


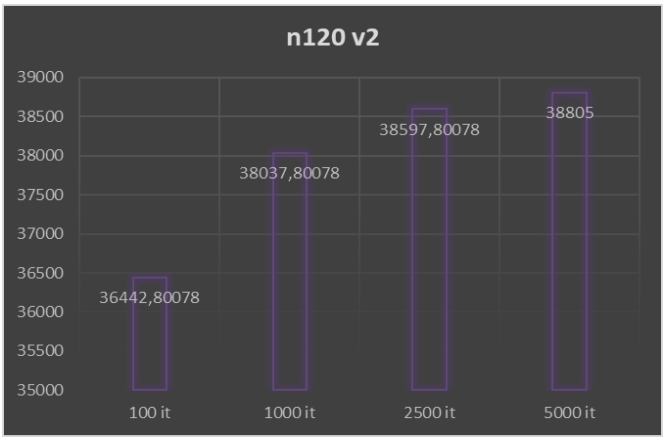
Para além disso, existe uma grande influência por parte do número de iterações no algoritmo Trepa-colinas para cada ficheiro lido. Quanto maior o número de iterações, melhores as soluções obtidas.

- **Trepa-colinas com 2 Vizinhança:**

Número de Elementos		*Número de Iterações*			
		100 it	**1000 it**	**2500 it**	**5000 it**
10: n010.txt	**Melhor**	1228	1228	1228	1228
	MBF	1228	1228	1228	1228
12: n012.txt	**Melhor**	1000	1000	1000	1000
	MBF	982,099976	988,599976	991,5	991,800012
30: n030.txt	**Melhor**	4670	4918	5036	5142
	MBF	4477	4778,899902	4826,899902	4905,5
60: n060.txt	**Melhor**	16127	16995	17248	17243
	MBF	15653,29981	16566,40039	16718	16802,09961
120: n120.txt	**Melhor**	36990	38877	39251	39129
	MBF	36442,80078	38037,80078	38597,80078	38805
240: n240.txt	**Melhor**	123041	127963	128843	130037
	MBF	121902,3984	126566	127767,2969	128517.898438

Pelos valores obtidos, é possível notar um aumento tanto nas médias obtidas como nas qualidades das melhores soluções. Por isso mesmo, é possível concluir que o algoritmo com duas vizinhanças devolve melhores resultados do que com apenas uma.





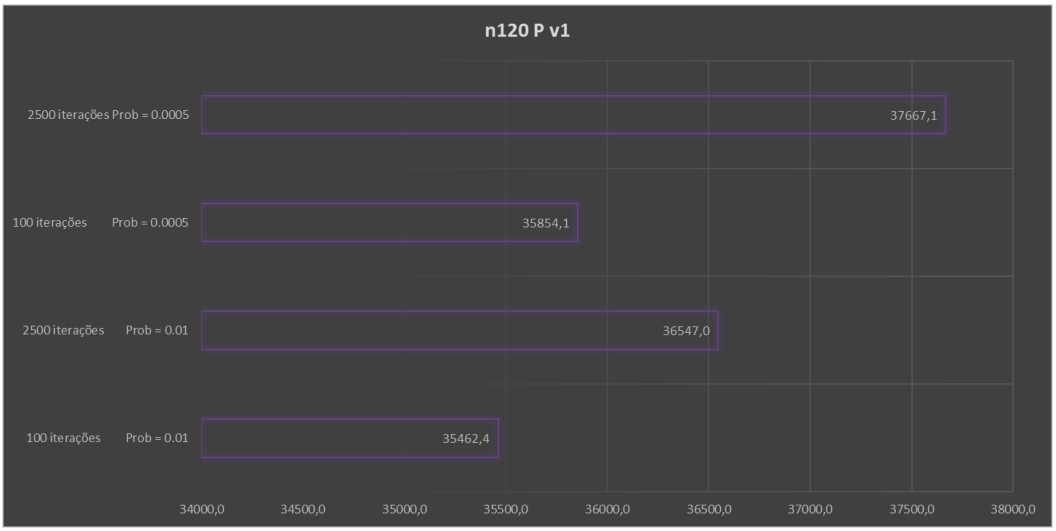
É possível verificar também com os gráficos obtidos, a influência do número de iterações usado neste algoritmo que sofreu alterações.

- **Trepa-colinas Probabilístico:**

Número de Elementos		*100 iterações Prob = 0.01*	*2500 iterações Prob = 0.01*	*100 iterações Prob = 0.0005*	*2500 iterações Prob = 0.0005*
10: n010.txt	**Melhor**	1228	1228	1228	1228
	MBF	1215,099976	1217,099976	1228	1228
12: n012.txt	**Melhor**	961	984	967	1000
	MBF	843,599976	892,599976	895,299988	981
30: n030.txt	**Melhor**	4495	4482	4736	4885
	MBF	4211,7	4240,100098	4407,899902	4720,5
60: n060.txt	**Melhor**	15509,0	16017,0	15862	16656
	MBF	15169,9	15494,2	15295	16137,9
120: n120.txt	**Melhor**	35892,0	37012,0	36357,0	37951,0
	MBF	35462,4	36547,0	35854,1	37667,1
240: n240.txt	**Melhor**	121628	126804	122880	128087
	MBF	119892,2031	123807,6016	120576,7969	126278,6016

Pelos valores obtidos, é possível verificar que quando a probabilidade (de aceitar soluções inferiores) é mais baixa, os resultados são mais próximos do desejado.

No gráfico pode-se observar que na probabilidade 0.01 e 0.0005, a diferença é relativamente considerável!



4. Conclusão

Para concluir, este projeto foi mais difícil do que esperado, apesar da questão a ser desenvolvida não ter sido a mais complexa, utilizar a linguagem C para este tipo de problemas tornou-se um pouco mais confuso para nós, especialmente na parte do código relativo ao algoritmo evolutivo, em que não conseguimos colocar o código funcional de modo a obter os valores esperados, também pelo facto de o algoritmo ser mais complexo.

A implementação do algoritmo Trepa-colinas foi feita com sucesso na nossa opinião, até mesmo pelo resultados obtidos para cada um dos ficheiros lidos. Para além disso, é um algoritmo rápido e que apenas apresenta mais demoras ao fazer um numero maior de iterações ou se o número de elementos for muito alto.

Com a realização deste trabalho, ficamos a conhecer um pouco melhor estes algoritmos e a sua importância na aplicação da otimização de problemas matemáticos no ramo da Inteligência Artificial.

5. Bibliografia

https://files.isec.pt/DOCUMENTOS/SERVICOS/BIBLIO/teses/Tese_Dout_Francisco-Pereira.pdf

https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

https://www.geeksforgeeks.org/genetic-algorithms/

https://www.maxwell.vrac.puc-rio.br/10863/10863_4.PDF

https://www.gsigma.ufsc.br/~popov/aulas/ia/modulo3/index.html

https://www.dei.isep.ipp.pt/~jtavares/PhD_Tese/capitulo_6.pdf

https://www.maxwell.vrac.puc-rio.br/28372/28372_4.PDF