Introdução à Inteligência Artificial
Trabalho Prático nº 2 – Problema de Otimização
Coin Change Problem

Realizado por:

- Rodrigo Bernarda nº 2021136740 email: a2021136740@isec.pt

Coimbra, 15 de dezembro de 2024

Índice

Introdução

Neste relatório, abordarei detalhadamente o desenvolvimento do segundo prático da unidade curricular de “Introdução à Inteligência Artificial”. O objetivo deste projeto consiste em implementar métodos de otimização que ajudem a resolver o *Coin Change Problem*. Para isso, serão utilizados algoritmos de pesquisa local, algoritmos evolutivos e algoritmos híbridos, que combinam as duas abordagens anteriores.

No *Coin Change Problem,* é nos dados um conjunto de N moedas diferentes e um valor alvo a ser atingido, sendo o desafio determinar a combinação que utilize o menor número de moedas possível para atingir esse valor.

Ao longo deste relatório, serão descritas as funcionalidades implementadas, justificadas as escolhas feitas e apresentados os testes efetuados, seguidos de uma análise detalhada dos seus resultados.

Algoritmo de Pesquisa Local

Para o método de pesquisa local, foi escolhido o algoritmo de trepa-colinas. Este algoritmo caracteriza-se por partir de uma solução inicial dada ou gerada aleatoriamente e, a cada passo, move-se para uma solução vizinha que melhore o valor da função objetivo, parando quando não encontrar uma solução vizinha melhor que a atual.

No contexto deste projeto, o algoritmo de trepa-colinas começa com uma solução inicial gerada aleatoriamente. Esta solução é representada por um *array* de inteiros, onde cada posição indica a quantidade de moedas de um determinado tipo a ser utilizada.

A partir dessa solução inicial, será gerada uma solução vizinha, que consiste em trocar o nº de moedas utilizadas entre dois tipos, desde que uma delas seja acima de 0. A melhor solução entre as duas será a que usar menos moedas no total (*fitness*), porém nem todas as soluções chegam ao valor desejado, sendo preciso reparar e penalizar essas soluções.

A forma como as soluções são reparadas seguem a seguinte lógica:

1. O *array* que representa a solução é percorrido em ordem aleatória, removendo o excesso de moedas, caso o valor alcançado ultrapasse o valor alvo.
2. O *array* é percorrido novamente, também em ordem aleatória, adicionando moedas conforme necessário para atingir o valor alvo.

Se a solução reparada for ainda inválida (ou seja, não atingir o valor pretendido), o seu custo (número total de moedas utilizadas) sofrerá uma penalização seguindo a seguinte fórmula:

- Custo Penalizado = custo atual + |valor alcançado – valor desejado| * 1000

Algoritmo Evolutivo

Algoritmos evolutivos são outro tipo de métodos de otimização inspirados no processo de evolução biológica. À semelhança do algoritmo de trepa-colinas, este método de otimização começa com uma população inicial de soluções, e a cada iteração (geração), através de um método de seleção, são selecionadas as melhores soluções para criar soluções através de métodos de recombinação (combinação de características entre duas soluções) e mutações (pequenas alterações aleatórias às soluções). Este processo continua até atingir o número máximo de gerações.

No contexto deste projeto, serão utilizados 2 métodos diferentes de seleção, recombinação e mutação.

1. **Seleção**:
    i. **Torneio:** Realiza competições entre dois indivíduos escolhidos aleatoriamente da população, selecionando o de melhor *fitness* (custo) para os pais da próxima geração.
    ii. **Roleta:** Seleciona pais com base numa probabilidade proporcional ao *fitness* de cada indivíduo, favorecendo os mais aptos, permitindo, porém, a seleção de indivíduos menos aptos.
2. **Recombinação (*crossover*):**
    i. **Uniforme:** Combina dois pais trocando genes a partir de um ponto aleatório, gerando dois descendentes que misturaram características (valores das soluções) dos pais.
    ii. **Dois pontos de corte:** Adiciona um outro ponto de corte à método de *crossover* anterior.
3. **Mutação:**
    i. **Incremento/Decremento:** Modifica os valores dos genes (valor da solução) aumentando ou diminuindo em 1, seguindo uma probabilidade definida.
    ii. **Troca:** Permuta os valores de dois genes aleatórios dentro de um descendente.

As funções para gerar soluções aleatórias e as avaliar serão as mesmas utilizadas no algoritmo de pesquisa local.

Algoritmo Híbrido

Para este último método de pesquisa, foi realizada uma combinação dos dois melhores algoritmos encontrados de pesquisa local e evolutivo. No próximo capítulo, analisarei quais são os melhores parâmetros para esses algoritmos.

Este algoritmo híbrido será usado de duas maneiras diferentes. A primeira será usada aquando da criação da população inicial, ou seja, em vez de começar com uma população totalmente aleatória, melhoramos a população inicial com o algoritmo de pesquisa local, que servirá de ponto de partida para o algoritmo evolutivo. A segunda maneira será igual à primeira, mas o algoritmo de pesquisa local será também usado para tentar melhorar a população final gerada pelo algoritmo evolutivo.

Testes Realizados e Devidas Análises

Neste capítulo serão feitos testes aos respetivos algoritmos, sendo os principais alvos de análise o custo da melhor solução obtida e a média obtida a partir das melhores soluções encontradas em cada uma das repetições (MBF). Para cada teste será formulada uma hipótese que servirá de base para a análise do algoritmo em questão. Essas hipóteses envolvem modificar um parâmetro ou outro que envolve a experiência, como o tipo de vizinhança e nº de iterações a utilizar no algoritmo de pesquisa local ou o método de mutação e a sua probabilidade no algoritmo evolutivo. Cada teste terá 30 repetições e os seus resultados serão devidamente analisados.

Teste nº1: Melhor Tipo de Vizinhança

Neste primeiro teste focado no algoritmo de pesquisa local, serão testados o número de vizinhos gerados a partir de uma solução e o número de iterações que uma experiência terá, que variarão entre 1 e 2 vizinhos, e 100, 1000 e 10000 iterações.

Trepa-Colinas		File 1		File 2		File 3		File 4		File 5	
Nº Vizinhanças	Nº Iterações	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
1	100	2	2.000	20	31.000	152	173.400	132	216.167	4	4.167
	1000	2	2.000	20	30.467	154	176.000	124	207.833	4	4.000
	10000	2	2.000	20	31.300	158	177.967	142	204.667	4	4.000
2	100	2	2.000	20	31.333	132	176.933	112	216.667	4	4.033
	1000	2	2.000	20	29.333	149	180.133	134	219.367	4	4.000
	10000	2	2.000	20	29.900	129	179.067	108	197.433	4	4.000

Fig. 1 – Resultados do Teste nº1

Como se pode observar, os resultados são melhores com 2 vizinhos, alcançando melhores soluções no ficheiro 3 e 4 com 10000 iterações, quando comparado com as soluções com 1 vizinho apenas. Porém, o MBF das experiências com 1 vizinho apenas é ligeiramente melhor.

Teste nº2: Aceitar Soluções Iguais

Para este segundo teste, serão testados os mesmos parâmetros do teste anterior, porém neste teste o algoritmo aceitará como melhores soluções, soluções com custo igual à da melhor solução atual.

Trepa-Colinas (Soluções Iguais)		File 1		File 2		File 3		File 4		File 5	
Nº Vizinhanças	Nº Iterações	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
1	100	2	2.000	23	30.033	107	166.700	130	214.867	4	4.067
	1000	2	2.000	20	32.867	108	171.667	147	211.600	4	4.000
	10000	2	2.000	20	30.633	135	173.767	127	202.067	4	4.000
2	100	2	2.000	20	30.700	120	178.000	153	229.667	4	4.033
	1000	2	2.000	20	30.300	130	177.300	139	202.067	4	4.000
	10000	2	2.000	20	31.367	157	179.500	136	203.967	4	4.000

Fig. 2 – Resultado do Teste nº 2

Neste teste, os resultados foram superiores utilizando apenas 1 vizinhança, tendo o algoritmo encontrado melhores soluções no ficheiro 3, comparado com os resultados do teste anterior.

Teste nº3: Método de Seleção

Para este terceiro teste já será utilizado o algoritmo evolutivo, sendo o objetivo descobrir o melhor método de seleção. Para isso, o método de seleção utilizado variará entre o de torneio e o de roleta. Adicionalmente o número de gerações variará entre 100, 1000 e 1000. Serão utilizados o operador de recombinação uniforme e mutação por incremento/decremento, ambos com probabilidade de 0,1.

Algoritmo Evolutivo		File 1		File 2		File 3		File 4		File 5	
Método de Seleção	Nº Gerações	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
Torneio	100	2	2.000	20	20.000	103	115.767	126	144.067	4	4.000
	1000	2	2.000	20	20.000	102	102.000	108	108.000	4	4.000
	10000	2	2.000	20	20.000	102	102.000	108	108.000	4	4.000
Roleta	100	2	2.000	26	29.767	127	140.300	218	278.733	4	4.000
	1000	2	2.000	23	29.767	130	141.700	213	278.833	4	4.000
	10000	2	2.000	25	29.667	129	140.233	229	284.500	4	4.000

Fig. 3 – Resultados do Teste nº3

Como se pode observar, o método de seleção por torneio apresenta resultados muito melhores quando comparados com os do método de seleção da roleta. Adicionalmente, a diferença dos resultados entre as experiências com 1000 e 10000 gerações são quase negligentes, sendo 1000 gerações suficientes para os próximos testes. De notar que os primeiros resultados do algoritmo evolutivo são muito superiores aos do algoritmo de pesquisa local.

Teste nº4: Operador de Recombinação

Neste quarto teste, serão estudados os dois operadores de recombinação: o uniforme e o com dois pontos de corte. Neste teste será utilizado o método de seleção por torneio e o número de gerações ficará pelos 1000. Adicionalmente, a probabilidade de recombinação variará entre 0.3, 0.5 e 0.7, e a probabilidade de mutação foi baixada para 0.001.

Algoritmo Evolutivo		File 1		File 2		File 3		File 4		File 5	
Operador de Recombinação	P(Recombinação)	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
Uniforme	0.3	2	2.000	22	25.433	102	143.100	132	159.500	4	4.000
	0.5	2	2.000	23	25.133	102	134.467	125	151.333	4	4.000
	0.7	2	2.000	22	24.533	102	113.533	111	137.433	4	4.000
Dois pontos de corte	0.3	2	2.000	20	24.400	102	121.600	116	149.533	4	4.000
	0.5	2	2.000	20	24.867	102	112.933	108	133.600	4	4.000
	0.7	2	2.000	22	25.133	102	109.600	115	128.567	4	4.000

Fig. 4 – Resultados do Teste nº4

Os resultados destes testes mostram que o operador de recombinação com dois pontos de corte tem uma vantagem sobre o operador de recombinação uniforme, sendo ela mais clara no ficheiro 4, onde foi encontrada a melhor solução e o MBF foi geralmente mais baixo que os resultados do operador de recombinação uniforme.

Teste nº5: Operador de Mutação

Para este teste, serão testados os operadores de mutação por incremento/decremento e por troca. O operador de recombinação a utilizar será o com dois pontos de corte. A probabilidade de mutação variará entre 0.005, 0.01 e 0.1, e a probabilidade de recombinação ficará pelos 0.5. Os restantes parâmetros permanecerão iguais.

Algoritmo Evolutivo		File 1		File 2		File 3		File 4		File 5	
Operador de Mutação	P(Mutação)	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
Incremento/Decremento	0.005	2	2.000	20	24.567	102	102.000	108	110.367	4	4.000
	0.01	2	2.000	20	22.500	102	102.000	108	108.167	4	4.000
	0.1	2	2.000	20	20.000	102	102.000	108	108.000	4	4.000
Troca	0.005	2	2.000	20	20.600	102	108.367	108	108.100	4	4.000
	0.01	2	2.000	20	20.600	102	105.033	108	108.067	4	4.000
	0.1	2	2.000	20	20.000	102	102.000	108	108.000	4	4.000

Fig. 5 – Resultados do Teste nº5

Como se pode observar, o operador de mutação por troca apresenta resultados ligeiramente melhores, sendo eles unicamente visíveis no MBF das experiências. O destaque deste teste vai claramente para a probabilidade de mutação, que atingiu resultados perfeitos para ambos operadores, quando situada nos 0.1.

Teste nº6: Estratégias e Populações

Neste último teste do algoritmo evolutivo, o objetivo passa por perceber os resultados que se obtêm ao variar as estratégias para soluções válidas e o tamanho da população do algoritmo. Sendo assim, as estratégias usadas variarão entre a de reparação e penalização, e o tamanho da população variará entre 10, 50 e 150. O operador de mutação a utilizar será o por troca, e a probabilidade de mutação situar-se-á nos 0.1, mantendo-se todos os outros parâmetros. De notar que em todos os testes anteriores foi utilizado a estratégia de reparação para soluções inválidas.

Algoritmo Evolutivo		File 1		File 2		File 3		File 4		File 5	
Estratégia Soluções Inválidas	População	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
Reparação	10	2	2.000	20	20.733	102	107.433	108	108.267	4	4.000
	50	2	2.000	20	20.267	102	102.000	108	108.000	4	4.000
	150	2	2.000	20	20.000	102	102.000	108	108.000	4	4.000
Penalização	10	2	2.800	39	101.300	179	466.967	377	606.567	4	464.067
	50	2	2.000	33	43.800	179	432.300	330	441.000	4	39.833
	150	2	2.000	33	39.733	172	215.00	325	369.233	4	4.000

Fig. 6 – Resultados do Teste nº6

Estes resultados indicam que a estratégia por reparação é bastante superior à de penalização e que o tamanho da população não faz diferença quando utilizada a estratégia da reparação de solução, porém uma população maior melhora os resultados da estratégia de penalização.

Teste nº7: Algoritmo Híbrido 1

Com base nos testes anteriores, cheguei à conclusão que o melhor algoritmo de pesquisa local deve utilizar 1 vizinhança, 10000 iterações e aceitar soluções iguais. O melhor algoritmo evolutivo utiliza o método de seleção por torneio, operador de recombinação com dois pontes de corte e 0.5 de probabilidade, e o operador de mutação por troca com 0.1 de probabilidade. Tendo isto em conta, e utilizando a estratégia de reparação de soluções inválida, será estudado primeiro os resultados obtidos deste algoritmo híbrido, em que o melhor algoritmo de pesquisa local é utilizado na criação da população inicial usada pelo algoritmo evolutivo.

Algoritmo Hibrído i)									
File 1		File 2		File 3		File 4		File 5	
Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
2	2.000	20	20.000	102	102.000	108	110.567	4	4.000

Fig. 7 – Resultados do Teste nº7

Como se pode observar, os resultados obtidos são quase perfeitos, faltando aperfeiçoar o MBF no ficheiro 4.

Teste nº8: Algoritmo Híbrido 2

Neste último teste, experimentarei, à semelhança do teste anterior, utilizar o melhor algoritmo de pesquisa local para refinar também a população final gerada pelo algoritmo evolutivo, para perceber se dessa maneira é possível alcançar os melhores resultados possíveis. Todos os parâmetros utilizados são os mesmos que no teste anterior.

Algoritmo Hibrído ii)									
File 1		File 2		File 3		File 4		File 5	
Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF	Custo	MBF
2	2.000	20	20.000	102	102.000	108	110.333	4	4.000

Fig. 8 – Resultados do Teste nº8

Como se pode observar, embora este algoritmo híbrido, embora consiga encontrar a melhor solução possível, não foi possível obter um MBF perfeito no ficheiro 4.

Conclusão

Existem diversas conclusões que pude tirar depois de analisados os resultados das experiências realizadas.

Uma das conclusões é que o algoritmo de pesquisa local não consegue obter os melhores resultados possíveis em problemas que utilizem um maior número de moedas, como os problemas do ficheiro 3 e 4, sendo apenas capaz de obter resultados satisfatórios quando o tamanho do problema é menor.

Outra das conclusões que tirei é que, no geral, o algoritmo evolutivo obtém resultados consideravelmente melhores que o algoritmo de pesquisa local para problemas mais complexos. O método de seleção por torneio, por não usar nenhum tipo de probabilidade, é capaz de selecionar muito melhores soluções que o método da roleta. Os operadores de recombinação e a sua probabilidade não têm muita influência na obtenção de soluções ótimas, ao contrário da probabilidade de mutação que, quando situada nos 0.1, é capaz finalmente de obter soluções perfeitas, independentemente do operador utilizado. Foi possível chegar à conclusão que o tamanho da população tem mais influência na obtenção de melhores soluções, quando utilizada a estratégia de penalização de soluções inválidas, que, no geral, tem mais dificuldades a atingir soluções ótimas.

Combinando os melhores algoritmos encontrados nos testes aos métodos de pesquisa local e evolutivo, foi possível alcançar as melhores soluções possíveis, faltando, em ambas abordagens híbridas, obter um MBF perfeito no ficheiro 4.