Licenciatura em Engenharia Informática

Regime Pós-Laboral

Introdução à Inteligência Artificial

Problema da Diversidade Máxima de Grupo

2020/2021

Liliana Faustino 2017012944

Índice

Introdução

Este relatório faz parte do segundo trabalho prático de Introdução à Inteligência Artificial e pretende ajudar na interpretação dos resultados expostos no ficheiro Excel que anexa a este relatório.

O trabalho foi dividido em três etapas: na primeira foi utilizado um algoritmo de pesquisa local para avaliar esta questão. Neste trabalho, o algoritmo escolhido foi o trepa-colinas.

Na segunda etapa do trabalho, foram aplicados algoritmos genéticos e na terceira foram combinados os algoritmos desenvolvidos nas fases anteriores.

Com a recolha e análise destes dados, foi elaborado este relatório que visa apresentar as melhores conclusões sobre este problema.

Algoritmos de Pesquisa Local

Um algoritmo de pesquisa local define, para cada solução, uma vizinhança com características muito semelhantes. Para problemas de maximização, como o que temos em mãos, uma forma de implementar um algoritmo de pesquisa local é percorrer a vizinhança da solução “em mãos” à procura de uma solução com uma melhor qualidade. Se a dita solução vizinha for encontrada, torna-se a solução “em mãos” e repetimos o processo.

Algoritmo Escolhido: Trepa Colinas

No trabalho, o algoritmo de pesquisa local escolhido foi o Trepa Colinas e este algoritmo comporta-se como se estivesse a escalar uma montanha: olha à sua volta e percebe se existe um melhor caminho (uma solução que apresente uma qualidade maior). Se existir, toma esse caminho como seu e continua a executar o mesmo pensamento.

Não existem algoritmos perfeitos, o Trepa Colinas não é excepção: tem problemas associados como máximos locais, planaltos e ainda cristas. Existem algumas estratégias que ajudam a evitar estes problemas que vamos explorar mais tarde.

Neste trabalho foram exploradas 4 modificações do Trepa Colinas: o first choice, o first choice mas aceitando soluções com a mesma qualidade, trepa colinas gerando quatro vizinhanças e aceitando soluções de qualidade igual e, por fim um trepa colinas probabilístico . O objetivo destas modificações foi sempre encontrar uma melhor solução para o problema em mãos e, de alguma forma, conseguir corrigir os principais problemas do algoritmo escolhido.

Trepa Colinas First Choice

O trepa colinas first choice é um método probabilístico pelo que precisamos de várias execuções do algoritmo para obtermos resultados válidos. Para esta fase foram realizadas sempre 100 repetições para conseguir obter resultados com alguma validade estatística.
Segue um quadro explicativo do algoritmo utilizado e aplicado aos diversos dados.

```
function trepa-colinas (problema) retorna um Estado que é um máximo
local
      inputs: problema
      variáveis locais: solucao_atual
                        solucao_vizinha

      solucao_atual <- Gera-Solucao(Estado-Inicial[problema])
      loop do
      solucao_vizinha <- um sucessor com uma maior qualidade do que a
solucao_atual
      if qualidade[solucao_vizinha] <= qualidade[solucao_atual] the
return Estado[solucao_atual]
      solucao_atual <- solucao_vizinha
```

A condição de saída escolhida para terminar o ciclo foi o número de iterações. Enquanto medidas de desempenho foram escolhidas duas: a melhor solução obtida - no caso o máximo de qualidade encontrada - e a média da melhor avaliação (MBF).

Análise de Resultados

Para avaliar a qualidade deste algoritmo em termos de produção de resultados foram utilizadas várias fontes de dados e alterando o número de iterações (condição de saída) - representados nas colunas - para perceber como é que esta mudança afetava os resultados produzidos. Em todas as análises, utilizamos como método de comparação os valores fornecidos pelos docentes (coluna Best).

Ficheiro n010

	100 it	1000 it	5000 it	10000 it	Best
Melhor	1228	1228	1228	1228	1228
MBF	1226.390015	1225.255981	1224.671631	1224.651855	

Utilizando este pequeno conjunto de dados para perceber a eficácia do algoritmo percebemos que ele é extremamente eficaz. Mesmo com apenas 100 iterações foi capaz de encontrar a melhor solução. Contudo, não podemos deixar de reparar que os valores de MBF vão diminuindo à medida que aumentamos as iterações. Neste caso, com diferenças tão pequenas, não é possível ser conclusivo mas é uma nota necessária.

Ficheiro n012

	100 it	1000 it	5000 it	10000 it	Best
Melhor	956	967	1000	1000	1000
MBF	690.849976	697.567993	700.497009	697.136230	

Tal como desconfiado, quando a quantidade de dados aumenta, começamos a perceber as debilidades deste algoritmo. No caso desta fonte de dados, só conseguimos encontrar a solução às 5000 iterações. No caso dos valores do MBF não conseguimos estabelecer um padrão com os dados anteriores pelo que é melhor aguardar mais dados.

Ficheiro n30

	100 it	1000 it	5000 it	10000 it	Best
Melhor	4234	4463	4537	4558	5194
MBF	3800.340088	3811.019043	3806.526855	3804.536865	

Tal como antecipado, já não conseguimos encontrar a solução ótima mesmo quando utilizamos 10.000 iterações. Conseguimos observar uma melhoria da qualidade da solução à medida que aumentamos as iterações o que nos faz concluir que talvez conseguíssemos atingir a melhor solução se aumentássemos as iterações. Relativamente à média, continuamos sem um padrão quando comparamos com os resultados anteriores.

Ficheiro n60

	100 it	1000 it	5000 it	10000 it	Best
Melhor	14731	15472	15675	15494	18714
MBF	13937.019531	13945.640625	13968.024414	13958.055664	

Neste conjunto de dados aconteceu algo curioso: a qualidade da solução gerada nas 10.000 iterações é inferior à qualidade das soluções gerada nas 5.000 iterações. Isto confirma que o trepa-colinas first choice não é fiável sozinho. Também a média da qualidade nas 10.000 iterações é inferior, contudo, uma nota importante: as médias estão todas muito próximas independente do número de iterações.

Estamos muito, muito longe da melhor solução. A melhor qualidade encontrada pelo algoritmo está 16% abaixo da qualidade esperada.

Ficheiro n120

	100 it	1000 it	5000 it	10000 it	Best
Melhor	35041	35450	35774	35907	46647
MBF	33589.199219	33517.625000	33543.703125	33533.519531	

Na análise deste ficheiro, os resultados já condizem com o esperado: a qualidade da melhor solução encontrada vai aumentando consoante o número de iterações aumenta. Mais uma vez, as médias estão muito próximas sendo que existe um acontecimento curioso: a melhor média é a de com menos iterações. Estamos ainda mais longe da melhor qualidade: agora temos uma solução 23% inferior à ideal.

Ficheiro n240

	100 it	1000 it	5000 it	10000 it	Best
Melhor	118987	119849	120212	121134	155536
MBF	116015.992188	115699.648438	115748.914062	115768.820312	

Neste ficheiro, as melhores soluções começaram a ser algo melhoradas consoante o número de iterações, mas nunca foi suficiente para sequer se aproximar da melhor solução conhecida. Contudo, a melhor solução encontrada é de 22% inferior à ideal, o que significa uma melhoria em relação ao ficheiro anterior.

Conclusões gerais

		100 it	1000 it	5000 it	10000 it	Best
n010.txt	Melhor	1228	1228	1228	1228	1228
	MBF	1226.390015	1225.255981	1224.671631	1224.651855	
n012.txt	Melhor	956	967	1000	1000	1000
	MBF	690.849976	697.567993	700.497009	697.136230	
n030.txt	Melhor	4234	4463	4537	4558	5194
	MBF	3800.340088	3811.019043	3806.526855	3804.536865	
n060.txt	Melhor	14731	15472	15675	15494	18714
	MBF	13937.019531	13945.640625	13968.024414	13958.055664	
n0120.txt	Melhor	35041	35450	35774	35907	46647
	MBF	33589.199219	33517.625000	33543.703125	33533.519531	
n0240.txt	Melhor	118987	119849	120212	121134	155536
	MBF	116015.992188	115699.648438	115748.914062	115768.820312	

Com o trepa colinas first choice apenas conseguimos descobrir a melhor solução em duas fontes de dados e sendo que numa esta descoberta só foi possível após as 5.000 iterações. Desta forma, a conclusão só pode ser uma: este algoritmo não é bom o suficiente para ser eficaz em grande quantidade de dados.

Ainda assim, é quase sempre notório que o aumento de iterações melhora significativamente a qualidade das soluções o que pode representar que talvez fosse possível atingir a melhor solução no ficheiro de n030.

Para grandes quantidades de dados, talvez seja melhor ideia melhorar este algoritmo para algo mais composto.

Trepa Colinas aceita soluções com qualidade igual

Fazendo uma pequena alteração do código (apresentado abaixo e sinalizado a laranja) será que conseguimos melhorar a qualidade das soluções? A análise abaixo foi realizada exatamente com os mesmos dados e com as mesmas condições. Esta alteração foi uma forma de tentar que o algoritmo fosse capaz de sair de planaltos.

```
function trepa-colinas (problema) retorna um Estado que é um máximo
local
      inputs: problema
      variáveis locais: solucao_atual
                        solucao_vizinha

      solucao_atual <- Gera-Solucao(Estado-Inicial[problema])
      loop do
      solucao_vizinha <- um sucessor da solucao_atual
      if qualidade[solucao_vizinha] < qualidade[solucao_atual] the
return Estado[solucao_atual]
      solucao_atual <- solucao_vizinha
```

Análise de Resultados

Por ser uma alteração pequena e apenas com a pretensão de perceber o impacto que uma pequena alteração pode causar, esta análise será uma comparação direta de todos os resultados e não analisados caso a caso.

Análise geral

		100 it		1000 it		5000 it		10000 it		Best
n010.txt	Melhor	1228	1228	1228	1228	1228	1228	1228	1228	1228
	MBF	1226.390015	1221.939941	1225.255981	1225.036987	1224.671631	1225.327637	1224.651855	1224.971680	
n012.txt	Melhor	956	912	967	957	1000	984	1000	1000	1000
	MBF	690.849976	689.570007	697.567993	694.851990	700.497009	697.407776	697.136230	697.913879	
n030.txt	Melhor	4234	4357	4463	4447	4537	4576	4558	4562	5194
	MBF	3800.340088	3774.520020	3811.019043	3804.635986	3806.526855	3802.951660	3804.536865	3802.208496	
n060.txt	Melhor	14731	14893	15472	15211	15675	15471	15494	15496	18714
	MBF	13937.019531	13988.750000	13945.640625	13960.677734	13968.024414	13958.405273	13958.055664	13952.568359	
n0120.txt	Melhor	35041	35288	35450	35520	35774	36058	35907	36224	46647
	MBF	33589.199219	33498.878906	33517.625000	33524.121094	33543.703125	33524.453125	33533.519531	33529.511719	
n0240.txt	Melhor	118987	118635	119849	118932	120212	120193	121134	120422	155536
	MBF	116015.992188	115914.14062	115699.648438	115835.187500	115748.914062	115818.562500	115768.820312	115789.148438	

Esta alteração no algoritmo foi uma desilusão no que toca aos resultados. No caso do documento n010, já tínhamos atingido a melhor solução no algoritmo anterior pelo que neste não foi surpresa também o atingir. Contudo, no grupo da 100 e das 1000 iterações a média da qualidade das soluções diminui e nos outros grupos aumentou (apenas ligeiramente).

No caso do ficheiro n012 os resultados anteriores foram todos melhores. Ou seja, aceitar soluções com qualidade igual não foi nada eficaz e não melhorou em nada os resultados. As médias mantiveram-se muito próximas, mas mais uma vez, nada que seja de valorizar.
No ficheiro n030 começamos a perceber que talvez exista alguma eficácia nesta alteração. Em 3 dos 4 grupos de iterações encontrámos melhores soluções. Será que conseguimos concluir que aceitar soluções com qualidade igual pode melhorar o nosso algoritmo?
No ficheiro n060 não conseguimos corroborar a teoria anterior. Mais uma vez, tivemos sucesso em apenas 50% dos grupos de iterações, o que não nos valida a teoria.
No ficheiro n120 melhoramos todos os resultados! Aceitar soluções com qualidade igual à atual fez com que melhorassem 0,08% a proximidade à solução ideal. Podemos portanto concluir, usando os dois resultados dos ficheiros anteriores, que aceitar estas soluções nos permitiu atingir melhores resultados.

Infelizmente, o ficheiro n240 veio desmentir a teoria e neste caso pioramos todos os resultados. Contudo, a média melhorou bastante, o que é um sinal muito bom.
Em jeito de resumo geral deste algoritmo, com baixa quantidade de dados não existe um benefício grande, contudo, quando a quantidade de dados aumenta, conseguimos obter mais e melhores resultados. Vamos portanto manter esta alteração para a próxima análise.

Trepa Colinas com 3 vizinhanças e aceita soluções de qualidade igual

Numa tentativa de tornar o algoritmo mais robusto, foi tomada a decisão de cada vez que se executava o ciclo eram geradas 3 soluções vizinhas (em vez de apenas uma) e guardava-se a melhor. Com esta alteração no algoritmo pretende-se uma melhoria significativa na qualidade das soluções apresentadas pois as hipóteses de encontrar uma solução melhor a cada iteração quadruplicaram.

```
function trepa-colinas_3_vizinhanca (problema) retorna um Estado que é
um máximo local
      inputs: problema
      variáveis locais: solucao_atual
                        solucao_vizinha1
                        solucao_vizinha2
                        solucao_vizinha3
                        solucao_vizinha

      solucao_atual <- Gera-Solucao(Estado-Inicial[problema])
      loop do
      solucao_vizinha1 <- um sucessor da solucao_atual
      solucao_vizinha2 <- um sucessor da solucao_atual
      solucao_vizinha3 <- um sucessor da solucao_atual

      solucao_vizinha <- max(solucao_vizinha1, solucao_vizinha2,
solucao_vizinha3)

      if qualidade[solucao_vizinha] < qualidade[solucao_atual] then
return Estado[solucao_atual]
      solucao_atual <- solucao_vizinha
```

Análise de Resultados

Daqui para a frente, todas as análises irão utilizar o melhor resultado obtido em comparação com o novo. A cor das células representa o algoritmo onde foi encontrado: laranja para o Trepa Colinas First Choice, azul para o Trepas Colinas First Choice aceitando soluções com qualidade igual. No caso da qualidade da solução ser igual, o critério de desempate é a melhor média que vai ser escolhida.

Desta forma, vamos obter uma comparação real capaz de avaliar a eficácia deste novo algoritmo representado nas tabelas a roxo.

Ficheiro n010

	100 it		1000 it		5000 it		10000 it		Best
Melhor	1228	1228	1228	1228	1228	1228	1228	1228	1228
MBF	1226.390015	1228	1225.255981	1228	1225.327637	1228	1224.971680	1228	

Sem muita surpresa, conseguimos atingir a qualidade máxima em todos os grupos de iterações. Porém, conseguimos perceber que a média foi sempre - sem excepção - a qualidade máxima. Com isto, é de fácil conclusão que o algoritmo conseguiu melhorar bastante com a geração de mais vizinhança.

Ficheiro n012

	100 it		1000 it		5000 it		10000 it		Best
Melhor	956	1000	967	1000	1000	1000	1000	1000	1000
MBF	690.849976	987,57	697.567993	992,95	700.497009	989,63	697.913879	991,450012	

Mesmo com o aumento dos dados, a melhor qualidade foi sempre atingida. Porém, a média já não foi 100% a melhor qualidade. Contudo, não existe um padrão com o número de iterações neste caso. Embora se note uma ligeiramente melhoria das 5.000 iterações para as 10.000 a verdade é que o melhor resultado atingido foi nas 1000. É um pouco estranho, mas pode ser que os próximos dados ajudem a tirar conclusões.

Ficheiro n030

	100 it		1000 it		5000 it		10000 it		Best
Melhor	**4357**	5092	**4463**	5194	**4576**	5194	**4562**	5194	5194
MBF	3774.520020	4847,689941	3811.019043	5059,779785	3802.951660	5067,319824	3802.208496	5067,350098	

Infelizmente, com este conjunto de dados e 100 iterações não conseguimos atingir o melhor resultado. Contudo, nos outros grupos de iterações conseguimos sempre atingir o objetivo. Como previsto na análise anterior, a verdade é que a média vai sempre aumentando, ainda que ligeiramente. A diferença entre uma vizinhança e 3 é absurda neste conjunto de dados, tanto em termos de qualidade como na alteração da média.

Ficheiro n60

	100 it		1000 it		5000 it		10000 it		Best
Melhor	**14893**	17311	**15472**	18367	**15675**	18545	**15496**	18630	18714
MBF	13988.750000	16671,18945	13945.640625	17965,91992	13968.024414	18233,53906	13952.568359	18242,74023	

Quando o tamanho do range de dados vai aumentando, o algoritmo começa a mostrar as suas debilidades. Neste caso, já não conseguimos atingir a melhor qualidade nem mesmo no máximo de iterações. A diferença é de menos de 1% mas ainda assim, não foi o resultado esperado. A média, mais uma vez, aumentou drasticamente e neste conjunto de dados é notório o quanto o número de iterações afeta a qualidade das soluções apresentadas assim como a sua média.

Ficheiro n120

	100 it		1000 it		5000 it		10000 it		Best
Melhor	**35288**	39906	35450	43855	**36058**	45174	**36224**	45584	46647
MBF	33498.878906	38904,82813	33517.625000	43073,28125	33524.453125	44549,03125	33529.511719	44920,14063	

Agora a surpresa é menor pois já não havia esperança de atingir a melhor solução. Entre o algoritmo anterior e este houve realmente uma melhoria enorme mas claramente esta alteração não foi suficiente para atingir o objetivo. Mesmo no caso das 10.000 iterações estamos cerca de 2% abaixo do valor que gostaríamos.

Confirma-se que o número de iterações tem influência na qualidade das soluções.

Ficheiro n240

	100 it		1000 it		5000 it		10000 it		Best
Melhor	118635	128743	**119849**	141247	**120212**	147300	**121134**	148938	155536
MBF	115914.14062	126110,6016	115699.648438	139809,5781	115748.914062	145803,0156	115768.820312	147518,375	

Com um conjunto tão grande de dados, o algoritmo é bastante ineficiente. Na melhor das hipóteses, ficou 4% atrás do que seria esperado. Ainda assim, ajudou a confirmar as teorias obtidas ao longo desta análise.

Conclusões Gerais

		100 it		1000 it		5000 it		10000 it		Best
n010.txt	Melhor	1228	1228	1228	1228	1228	1228	1228	1228	1228
	MBF	1226.390015	1228	1225.255981	1228	1225.327637	1228	1224.971680	1228	
n012.txt	Melhor	956	1000	967	1000	1000	1000	1000	1000	1000
	MBF	690.849976	987,57	697.567993	992,95	700.497009	989,63	697.913879	991,450012	
n030.txt	Melhor	4357	5092	4463	5194	4576	5194	4562	5194	5194
	MBF	3774.520020	4847,689941	3811.019043	5059,779785	3802.951660	5067,319824	3802.208496	5067,350098	
n060.txt	Melhor	14893	17311	15472	18367	15675	18545	15496	18630	18714
	MBF	13988.750000	16671,18945	13945.640625	17965,91992	13968.024414	18233,53906	13952.568359	18242,74023	
n0120.txt	Melhor	35288	39906	35450	43855	36058	45174	36224	45584	46647
	MBF	33498.878906	38904,82813	33517.625000	43073,28125	33524.453125	44549,03125	33529.511719	44920,14063	
n0240.txt	Melhor	118635	128743	119849	141247	120212	147300	121134	148938	155536
	MBF	115914.14062	126110,6016	115699.648438	139809,5781	115748.914062	145803,0156	115768.820312	147518,375	

Com a geração de 3 soluções vizinhas, o algoritmo tornou-se muito mais eficaz para analisar conjuntos de dados um bocadinho maiores. Até ao ficheiro de n030, o algoritmo foi eficaz a partir das 1000 iterações.

Contudo, a partir daí as soluções nunca mais conseguiram atingir a qualidade pretendida e, à medida que o número de dados para serem analisados aumentaram, cada vez mais se foi afastando daquilo que seria o ideal. Um ponto a registar é que o aumento de iterações melhora e muito tanto a média como a qualidade da solução.

Se fosse possível (em termos de tempo e capacidade de computação) o ideal seria aumentar para a ordem dos milhões de alterações para perceber se seria mais eficaz com grandes quantidades de dados.

Trepa Colinas com 3 vizinhanças probabilístico

Para tentar contornar o aparecimento de cristas, uma das soluções é - por vezes - aceitar uma solução com uma qualidade inferior. Foram feitos alguns testes utilizando sempre o mesmo número de iterações e utilizando probabilidades diferentes para ser possível efetuar uma melhor análise.

```
function trepa-colinas_3_vizinhanca_probabilistico (problema) retorna um
Estado que é um máximo local
      inputs: problema
      constants: prob
      variáveis locais: solucao_atual
                        solucao_vizinha1
                        solucao_vizinha2
                        solucao_vizinha3
                        solucao_vizinha

      solucao_atual <- Gera-Solucao(Estado-Inicial[problema])
      loop do
      solucao_vizinha1 <- um sucessor da solucao_atual
      solucao_vizinha2 <- um sucessor da solucao_atual
      solucao_vizinha3 <- um sucessor da solucao_atual

      solucao_vizinha <- max(solucao_vizinha1, solucao_vizinha2,
solucao_vizinha3)

      if qualidade[solucao_vizinha] < qualidade[solucao_atual] or
         random() < prob then
         return Estado[solucao_atual]
      solucao_atual <- solucao_vizinha
```

Análise de Resultados

Para uma melhor comparação foram feitos dois testes distintos: um com apenas 100 iterações e testando diferentes probabilidades e outro com 5000 iterações e testando as mesmas probabilidades do teste anterior. Segue abaixo a análise destes resultados.

100 Iterações - principais conclusões

Ficheiro		100 It	100 iterações Prob = 0.01	100 iterações Prob = 0.0005	Best
n010.txt	Melhor	1228	1228	1228	1228
	MBF	1228	1221,060059	1228	
n012.txt	Melhor	1000	1000	1000	1000
	MBF	987,57	982,98999	985,059998	
n030.txt	Melhor	5092	5001	5043	5194
	MBF	4847,689941	4783,830078	4812,720215	
n060.txt	Melhor	17311	17155	17416	18714
	MBF	16671,18945	16627,78906	16700,21094	
n0120.txt	Melhor	39906	39698	39725	46647
	MBF	38904,82813	38776,85156	38826,55859	
n0240.txt	Melhor	128743	128186	127899	155536
	MBF	126110,6016	126144,7734	126105,75	

Probabilidade de 10%

Neste teste, quando utilizada a probabilidade de 10% ser aceite uma solução com qualidade inferior, os resultados foram na sua generalidade bastante piores do que não aceitar essa dita solução.

No caso de termos poucos dados (ficheiros n010 e n012) a verdade é que continuamos a conseguir atingir a qualidade máxima mas a média da qualidade das soluções baixou.

Em todos os outros ficheiros, a qualidade da melhor solução é inferior à que tinhamos anteriormente, assim como a sua média (com excepção do ficheiro n240).

Ou seja, com uma probabilidade tão alta e tão poucas iterações, este método revela-se pouco eficiente para encontrar uma melhor solução.

Probabilidade de 0,05%

Mais uma vez, nos conjuntos de dados mais pequenos, a melhor solução possível foi encontrada, contudo, no caso do n012, a média baixou em relação à solução a roxo.

No ficheiro n030, a solução continua a ser mais baixa assim como a sua média. Porém, no caso do n060 a verdade é que a melhor solução encontrada e a média aumentam - ainda que ligeiramente. Talvez este ficheiro contenha alguns picos e esta estratégia possibilitou a procura de outras soluções.

Infelizmente, no ficheiro n120 e no n240 isto não se verificou e voltamos a perder qualidade.

5000 Iterações - Principais Conclusões

Ficheiro		5000 It	5000 iterações Prob = 0.01	5000 iterações Prob = 0.0005	Best
n010.txt	Melhor	1228	1228	1228	1228
	MBF	1228	1218,890015	1227,930054	
n012.txt	Melhor	1000	1000	1000	1000
	MBF	989,63	980,289978	992,950012	
n030.txt	Melhor	5194	5035	5194	5194
	MBF	5067,319824	4794,049805	5101,22998	
n060.txt	Melhor	18545	17216	18668	18714
	MBF	18233,53906	16651,18945	18246,11914	
n0120.txt	Melhor	45174	39677	45061	46647
	MBF	44549,03125	38787,05078	44446,05859	
n0240.txt	Melhor	147300	129032	146792	155536
	MBF	145803,0156	126118,8203	145585,875	

Probabilidade de 10%

Mais uma vez, encontramos a melhor solução, contudo a nossa média volta a descer. No ficheiro n030 não conseguimos atingir a melhor solução (ao contrário do que tinha acontecido no algoritmo anterior), no n060 a queda da qualidade da solução e da média é muito acentuada o que põe em causa a teoria levantada no ponto anterior.
Esta queda acentuada verifica-se em todos os outros ficheiros o que me leva a concluir que aceitar 10% das vezes uma solução de qualidade inferior não é uma boa estratégia para chegar à melhor solução possível.

Probabilidade de 0,05%

Nos conjuntos de dados mais pequenos, conseguimos sempre atingir a melhor solução e mesmo a média melhora em todos os conjuntos (menos no primeiro ficheiro, o que é totalmente normal devido ao pouco volume de dados.

No ficheiro n060 conseguimos até uma melhor solução do que a anterior o que prova que esta estratégia pode ser útil em determinados casos. Infelizmente, nos ficheiros seguintes isto não se confirma e os dados voltam a piorar. Talvez estes ficheiros tenham menos picos como achámos anteriormente?

Conclusões Gerais

Ficheiro		100 It	100 iterações Prob = 0.01	5000 iterações Prob = 0.01	5000 It	100 iterações Prob = 0.0005	5000 iterações Prob = 0.0005	Best
n010.txt	Melhor	**1228**	1228	1228	**1228**	1228	1228	**1228**
	MBF	**1228**	1221,060059	1218,890015	**1228**	1228	1227,930054	
n012.txt	Melhor	**1000**	1000	1000	1000	1000	**1000**	**1000**
	MBF	**987,57**	982,98999	980,289978	989,63	985,059998	**992,950012**	
n030.txt	Melhor	**5092**	5001	5035	5194	5043	**5194**	**5194**
	MBF	**4847,689941**	4783,830078	4794,049805	5067,319824	4812,720215	**5101,22998**	
n060.txt	Melhor	**17311**	17155	17216	18545	17416	**18668**	**18714**
	MBF	**16671,18945**	16627,78906	16651,18945	18233,53906	16700,21094	**18246,11914**	
n0120.txt	Melhor	**39906**	39698	39677	**45174**	39725	45061	**46647**
	MBF	**38904,82813**	38776,85156	38787,05078	**44549,03125**	38826,55859	44446,05859	
n0240.txt	Melhor	128743	128186	**129032**	**147300**	127899	146792	**155536**
	MBF	126110,6016	126144,7734	**126118,8203**	**145803,0156**	126105,75	145585,875	

De uma forma geral, podemos concluir que utilizar uma percentagem de 10% é demasiado violento para a eficácia do algoritmo. Em todos os casos, menos no maior range de dados, os dados foram bastante inferiores ao que já tínhamos, mesmo com o maior número de iterações esta estratégia não foi bem conseguida.

O mesmo não podemos, felizmente, concluir da probabilidade de 0,05%! Até ao ficheiro de n060 os resultados foram sempre melhores assim como as médias. Uma estratégia que com as iterações anteriores tinha sido ineficaz revela-se agora bastante melhor neste conjunto de dados.

Infelizmente, nos ficheiros com maiores dados o mesmo não se verifica o que nos pode levar a concluir que talvez não tenham sido feitas iterações suficientes para esta estratégia ser mais eficaz.

Conclusão dos Algoritmos de Pesquisa Local

A principal conclusão que podemos tirar de toda esta análise é que o algoritmo consegue ser eficaz em pequenas quantidades de dados. Quando começamos a aumentar muito o range, é notório que fica aquém do que gostaríamos.

As estratégias utilizadas para evitar os problemas conhecidos deste tipo de algoritmos foram quase sempre bem sucedidas: talvez aceitar soluções de qualidade igual não tenha sido muito eficaz numa fase tão precoce do algoritmo mas com toda a certeza fez diferente ao longo do resto da análise.

Para melhorar o algoritmo atual, poderia ser boa ideia aumentar o número de vizinhanças geradas e realizar mais iterações. Com certeza seriam geradas melhores soluções para o problema em causa.

Algoritmos Genéticos

Os algoritmos genéticos são um tipo de algoritmos evolutivos que têm como inspiração a evolução natural. A principal ideia é tentar imitar etapas do processo da evolução natural das espécies e tentando incorporá-lo num algoritmo computacional.
Em jeito de resumo, deve ser gerada a população inicial e novas populações a partir da mesma mas com propriedades genéticas superiores.
Neste trabalho, a população inicial foi gerada de forma a evitar soluções inválidas.
A nossa população depende de qual o ficheiro que estamos a analisar. Os critérios de análise usados foram vários tais como a taxa de mutação, a taxa de recombinação e foi mantido constante o tamanho do torneio.
Foram também testados vários operadores genéticos para que as conclusões fossem o mais abrangente possível.

Algoritmo base

Para o desenvolvimento do algoritmo base foi escolhida uma geração da população inicial sem soluções inválidas. Após encontrada a população inicial, é feito um torneio binário de forma a gerar o pai da próxima geração. Depois é aplicado um crossover uma mutação por troca aleatória.

```
function crossover (problema) retorna os descendentes
      inputs: parents
              tamanho_populacao
              numGenes
              parametro_PR
      variáveis locais: ponto_corte
                        probabilidade_gerada
      loop for i = 0 until tamanho_populacao incrementando 2
            if probabilidade_gerada < parametro_PR then
               ponto_corte = random(0, numGenes-1)
               loop for j = 0 until ponto_corte incrementando 1
                  descendente[i].posicao[j] <- parents[i].posicao[j]
                  descendente[i+1].posicao[j] <- parents[i+1].posicao[j]
```

```
            loop for j = ponto_corte until numGenes incrementando 1
               descendente[i].posicao[j] <- parents[i+1].posicao[j]
               descendente[i+1].posicao[j] <- parents[i].posicao[j]
         else
               descendente[i] <- parents[i]
               offspdescendentering[i+1] <- parents[i+1]
```

```
function mutacao_por_troca(problema) retorna os descendentes trocados
      inputs: descendentes
              tamanho_populacao
              numGenes
              parametro_PM
      variáveis locais: posicao_1
                        posicao_2
                        probabilidade_gerada
      loop for i = 0 until tamanho_populacao incrementando 1
            if probabilidade_gerada < parametro_PM then
               posicao_1 = random(0, numGenes-1)
               loop do
                   posicao_2 = random(0, numGenes-1)
               until posicao_2 != posicao_1
               descendente[i].posicao[posicao_1] < -
descendente[i].posicao[posicao_2]
               descendente[i].posicao[posicao_1] < -
descendente[i].posicao[posicao_2]
```

Ficheiro n010

		Ficheiro n010.txt	
		Algoritmo base	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	1228,0	1228,0
pm = 0.01	pr = 0.5	1228,0	1228,0
tsize = 2	pr = 0.7	1228,0	1228,0
pop = 100 (ger = 2500)	pm = 0.0	1228,0	1227,2
pop = 100	pm = 0.001	1228,0	1227,9
pr = 0.7	pm = 0.01	1228,0	1228,0
tsize = 2	pm = 0.05	1228,0	1228,0
pr = 0.7	pop = 10 (ger = 25K)	1228,0	1228,0
pm = 0.05	pop = 50 (ger = 5K)	1228,0	1228,0
tsize = 2	pop = 100 (ger = 2.5K)	1228,0	1228,0

Neste ficheiro, as conclusões que podemos tirar não são muito úteis devido a termos a solução máxima em todas, e mesmo na média apenas existem valores residuais que não foram o máximo. Este algoritmo é overkill para conjuntos de dados tão pequenos.

Ficheiro n012

		Ficheiro n012.txt	
		Algoritmo base	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	1000,0	993,2
pm = 0.01	pr = 0.5	1000,0	996,5
tsize = 2	pr = 0.7	1000,0	991,9
pop = 100 (ger = 2500)	pm = 0.0	1000,0	870,8
pop = 100	pm = 0.001	1000,0	976,3
pr = 0.7	pm = 0.01	1000,0	993,2
tsize = 2	pm = 0.05	1000,0	995,6
pr = 0.7	pop = 10 (ger = 25K)	1000,0	989,3
pm = 0.05	pop = 50 (ger = 5K)	1000,0	992,4
tsize = 2	pop = 100 (ger = 2.5K)	1000,0	995,3

Mais uma vez, atingimos a melhor solução em todas as tentativas, pelo que ainda continuamos a acreditar que este algoritmo é demasiado overkill para um conjunto de dados tão pequeno.

Ainda assim, ao olhar para as médias podemos concluir que quanto maior a probabilidade de mutação, melhor é a nossa média e que quanto maior é a nossa população melhor qualidade temos na nossa solução (mesmo que existam menos gerações).

Ficheiro n030

		Ficheiro n030.txt	
		Algoritmo base	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	5194,0	5066,2
pm = 0.01	pr = 0.5	5194,0	5047,1
tsize = 2	pr = 0.7	5194,0	5035,4
pop = 100 (ger = 2500)	pm = 0.0	4452,0	4050,4
pop = 100	pm = 0.001	4864,0	4546,9
pr = 0.7	pm = 0.01	5194,0	4870,4
tsize = 2	pm = 0.05	5194,0	5013,8
pr = 0.7	pop = 10 (ger = 25K)	5194,0	5018,6
pm = 0.05	pop = 50 (ger = 5K)	5194,0	4943,5
tsize = 2	pop = 100 (ger = 2.5K)	5194,0	4980,2

Com este conjunto de dados, as conclusões começam a surgir nomeadamente:

- Quanto maior a probabilidade de recombinação, menor a média da qualidade;
- A probabilidade de mutação é extremamente importante, quando a mesma está próxima de zero não conseguimos atingir a melhor solução.
- Ao contrário do que tinha acontecido no conjunto de dados, a média das soluções são afetadas pelo número de gerações: à medida que as gerações diminuem as médias também.

Para tentar perceber se o tamanho do torneio ia colmatar a probabilidade de mutação foram feitos com o tamanho de torneio 10 e 50:

		Ficheiro n030.txt	
		Algoritmo base	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100	tsize = 10 pm = 0.0	4386,0	4072,5
ger = 2500	tsize = 10 pm = 0.001	5071,0	4847,1
pr = 0.7	tsize = 50 pm = 0.0	4303,0	4050,8
	tsize = 50 pm = 0.001	5063,0	4827,6

Infelizmente, esta tentativa não teve sucesso. Apesar de ter tentando compensar a falta de mutação com o tamanho do torneio, não teve um efeito que justificasse manter esta estratégia.

Com estes dados, foi decidido que não se utilizam mais as probabilidades 0 e 0,1% para a mutação.

Ficheiro n060

		Ficheiro n060.txt	
		Algoritmo base	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	18260,0	17879,0
pm = 0.01	pr = 0.5	18214,0	17603,9
tsize = 2	pr = 0.7	17948,0	16040,0
pop = 100 (ger = 2500)	pm = 0.01	17821,0	15656,7
pr = 0.7	pm = 0.05	18489,0	15447,1
tsize = 2	pm = 0.1	18465,0	15901,5
pr = 0.7 tsize = 2	pop = 50 (ger = 5K)	18522,0	16964,8
pm = 0.05	pop = 100 (ger = 2.5K)	18273,0	15737,5
pr = 0.7 tsize = 10 pm = 0.05	pop = 100 (ger = 2.5K)	18458,0	18146,4

Pela primeira vez desde que estamos nos algoritmos genéticos, não conseguimos atingir a melhor solução possível

Continuamos a verificar que o aumento da probabilidade de recombinação é prejudicial ao encontro da solução ótima e que o aumento da probabilidade de mutação ajuda-nos a encontrar melhores soluções.

Mais uma vez, percebemos que se aumentarmos o número de gerações os resultados são melhores. Infelizmente, o mesmo não se verificou quanto passamos a ter 10 torneios.

Ficheiro n120

		Ficheiro n120.txt	
		Algoritmo base	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	41760,0	41116,1
pm = 0.01	pr = 0.5	42024,0	40994,0
tsize = 2	pr = 0.7	41499,0	37548,8
pr = 0.5 pop = 100	pm = 0.1	44767,0	44310,4
tsize = 2	pm = 0.05	44582,0	43802,0
pm = 0.05 pr = 0.5	pop = 50 (ger = 5K)	44199,0	38138,7
tsize = 2	pop = 100 (ger = 2.5K)	44041,0	36418,5
Melhores Indicadores	tsize= 10	45496,0	44796,0

Com este ficheiro foi decidido fazer uma escolha diferente. Na primeira fase, foram selecionadas 3 probabilidade de recombinação com o resto dos parâmetros fixos.

Utilizando a probabilidade de recombinação que deu melhor resultado absoluto (0.5), foram escolhidas duas probabilidades de mutação para perceber a evolução.

Utilizando exatamente os mesmo parâmetros, foram feitas simulações com mais gerações para perceber o impacto e, por fim, foi utilizado um torneio melhor com todos os indicadores que produziram os melhores resultados.

As conclusões tiradas foram as seguintes:

- Quanto maior a probabilidade de recombinação, pior é a qualidade das soluções (apesar da melhor ser superior, a média é inferior);
- A probabilidade de mutação afeta positivamente a qualidade das soluções;
- Aumentar o número de gerações resulta em soluções de maior qualidade quando comparadas com uma de população maior mas menos gerações;
- Finalmente, temos um exemplo onde o número de torneiros afetou positivamente a solução, tendo atingindo os melhores resultados até agora neste ficheiro.

Algoritmo Recombinação de 2 Pontos de Corte + Mutação por Troca

Para esta análise foi utilizado um novo operador genético: em vez de um crossover simples é feita uma recombinação com dois pontos. É expectável que a qualidade das soluções cresça.

```
function recombinacao_2_pontos_corte(problema) retorna os descendentes
      inputs: parents
              tamanho_populacao
              numGenes
              parametro_PR
      variáveis locais: ponto_corte_1
                        ponto_corte_2
                        probabilidade_gerada
      loop for i <- 0 until tamanho_populacao incrementando 2
            if probabilidade_gerada < parametro_PR then
               loop do
                        ponto_corte_1 <- random(0, numGenes-1)
                        ponto_corte_2 <- random(0, numGenes-1)
               until ponto_corte_1 != ponto_corte_2
               loop for j <- 0 until ponto_corte_1 incrementando 1
                  descendente[i].posicao[j] <- parents[i].posicao[j]
                  descendente[i+1].posicao[j] <- parents[i+1].posicao[j]


               loop for j <- ponto_corte_1 until ponto_corte_2
incrementando 1
                  descendente[i].posicao[j] <- parents[i+1].posicao[j]
                  descendente[i+1].posicao[j] <- parents[i].posicao[j]
               loop for j <- ponto_corte_2 until numGenes incrementando 1
                  descendente[i].posicao[j] <- parents[i].posicao[j]
                  descendente[i+1].posicao[j] <- parents[i+1].posicao[j]

            else
                  descendente[i] <- parents[i]
                  offspdescendentering[i+1] <- parents[i+1]
```

Ficheiro n010

Ficheiro n010.txt			
		Recombinação de 2 ponto de corte + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	1228,0	1228,0
pm = 0.01	pr = 0.5	1228,0	1228,0
tsize = 2	pr = 0.7	1228,0	1228,0
pop = 100 (ger = 2500)	pm = 0.0	1228,0	1228,0
pop = 100	pm = 0.001	1228,0	1228,0
pr = 0.7	pm = 0.01	1228,0	1228,0
tsize = 2	pm = 0.05	1228,0	1228,0
pr = 0.7	pop = 10 (ger = 25K)	1228,0	1228,0
pm = 0.05	pop = 50 (ger = 5K)	1228,0	1228,0
tsize = 2	pop = 100 (ger = 2.5K)	1228,0	1228,0

Este conjunto de dados continua a atingir todos os resultados com o valor máximo pelo que não se consegue tirar nenhum tipo de conclusão que nos ajude a perceber a evolução.

Ficheiro n012

Ficheiro n012.txt			
		Recombinação de 2 ponto de corte + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	1000,0	992,0
pm = 0.01	pr = 0.5	1000,0	993,2
tsize = 2	pr = 0.7	1000,0	993,9
pop = 100 (ger = 2500)	pm = 0.0	1000,0	883,0
pop = 100	pm = 0.001	1000,0	981,4
pr = 0.7	pm = 0.01	1000,0	994,0
tsize = 2	pm = 0.05	1000,0	995,3
pr = 0.7	pop = 10 (ger = 25K)	1000,0	984,3
pm = 0.05	pop = 50 (ger = 5K)	1000,0	990,7
tsize = 2	pop = 100 (ger = 2.5K)	1000,0	989,6

Com este conjunto de dados, conseguimos mais uma vez atingir com sucesso a melhor solução. Contudo existem algumas notas nas médias que valem a pena ser referidas:

- Ao contrário do que tem sido habitual, à medida que a probabilidade de recombinação aumenta, a média das soluções vai aumentando;
- Sem surpresas, a probabilidade de mutação é extremamente influente na qualidade das soluções;
- Por alguma razão, a média dos 25 ks de gerações é inferior o que é estranho.

Ficheiro n30

Ficheiro n030.txt			
		Recombinação de 2 ponto de corte + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	5194,0	5058,3
pm = 0.01	pr = 0.5	5194,0	5017,5
tsize = 2	pr = 0.7	5194,0	4936,3
pr = 0.7	pm = 0.1	5194,0	4958,9
tsize = 2	pm = 0.05	5194,0	4905,6
pr = 0.7	pop = 10 (ger = 25K)	5194,0	5108,8
pm = 0.05	pop = 50 (ger = 5K)	5194,0	4953,8
tsize = 2	pop = 100 (ger = 2.5K)	5194,0	4823,7
Melhores Indicadores	tsize= 10	5194,0	5061,0

Neste conjunto de dados conseguimos mais uma vez atingir a melhor solução possível. E com várias conclusões que podem ser apontadas:

- Como anteriormente visto, a probabilidade de recombinação diminui a média da qualidade das soluções;
- A probabilidade de mutação - neste caso - não melhorou a média das soluções;
- O melhor resultado é atingido utilizando mais gerações (25k).
- O tamanho do torneio teve resultados muito bons, tendo apenas sido ultrapassados pelo resultado com o máximo de gerações utilizado para este estudo.

Ficheiro n060

Ficheiro n060.txt			
		Recombinação de 2 ponto de corte + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	18351,0	17822,9
pm = 0.01	pr = 0.5	18134,0	17779,0
tsize = 2	pr = 0.7	18031,0	15325,2
pop = 100 (ger = 2500)	pm = 0.01	18003,0	15495,4
pr = 0.7	pm = 0.05	18549,0	15820,2
tsize = 2	pm = 0.1	18449,0	14512,1
pr = 0.7 tsize = 2	pop = 50 (ger = 5K)	18580,0	16046,9
pm = 0.05	pop = 100 (ger = 2.5K)	18412,0	14974,2
pr = 0.3 tsize = 10 pm = 0.05	pop = 100 (ger = 2.5K)	18528,0	18213,7

Neste ficheiro, mantivemos as mesmas condições do algoritmo base e tentámos utilizar as características que foram melhores no algoritmo anterior para agora conseguirmos comparar a melhoria do algoritmo.

As principais conclusões foram:

- Desta vez, e pela primeira vez que estamos neste algoritmo, a solução ótima não foi encontrada;
- A influência da probabilidade de recombinação está confirmada assim como a da probabilidade de mutação;
- Quando foram utilizadas mais gerações (5000) os resultados foram visivelmente melhores;
- O tamanho do torneio neste ficheiro não foi tão eficaz quanto nos outros.

Ficheiro n120

Ficheiro n120.txt			
		Recombinação de 2 ponto de corte + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	42883,0	42302,1
pm = 0.01	pr = 0.5	42762,0	41794,7
tsize = 2	pr = 0.7	42448,0	35418,1
pr = 0.5 pop = 100	pm = 0.1	44843,0	43933,1
tsize = 2	pm = 0.05	44466,0	43893,7
pm = 0.05 pr = 0.5	pop = 50 (ger = 5K)	44560,0	44056,8
Melhores Indicadores	tsize= 10	44998,0	44365,9

Nesta nova versão do algoritmo é clara a melhoria: em praticamente todos os casos os valores subiram. As principais conclusões que conseguimos tirar são:

- A probabilidade de recombinação não deve ser tão grande, a qualidade das soluções cai a pique cada vez que aumentamos em 20%;
- A probabilidade de mutação de 10% ajuda bastante a qualidade das soluções apresentadas;
- Sem dúvida, o que mais influencia neste caso, são o número de gerações. Apesar de não ser a solução ótima, é sem dúvida a melhor solução até agora;
- Mais uma vez, o aumento do torneio tornou as soluções mais aceitáveis. Talvez se fosse combinado 5000 de gerações e um torneiro de 50 fosse ainda melhor.

Algoritmo Recombinação Uniforme + Mutação por Troca

Para esta análise foi utilizado um novo operador genético: em vez de uma recombinação com dois pontos é feita uma recombinação totalmente uniforme. Não temos qualquer tipo de previsão de como serão os resultados desta tentativa.

```
function recombinacao_uniforme(problema) retorna os descendentes
      inputs: parents
              tamanho_populacao
              numGenes
              parametro_PR
      variáveis locais: probabilidade_gerada

      loop for i <- 0 until tamanho_populacao incrementando 2
            if probabilidade_gerada < parametro_PR then

               loop for j <- 0 until numGenes incrementando 1
                    if random (0, 1) == 1 then
                        descendente[i].posicao[j] <- parents[i].posicao[j]
                        descendente[i+1].posicao[j] <- parents[i+1].posicao[j]

                    else
                        descendente[i].posicao[j] <- parents[i+1].posicao[j]
                        descendente[i+1].posicao[j] <- parents[i].posicao[j]


            else
                  descendente[i] <- parents[i]
                  offspdescendentering[i+1] <- parents[i+1]
```

Ficheiro n010

Ficheiro n010.txt			
		Recombinação uniforme + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	1228,0	1228,0
pm = 0.01	pr = 0.5	1228,0	1228,0
tsize = 2	pr = 0.7	1228,0	1228,0
pop = 100 (ger = 2500)	pm = 0.0	1228,0	1228,0
pop = 100	pm = 0.001	1228,0	1228,0
pr = 0.7	pm = 0.01	1228,0	1228,0
tsize = 2	pm = 0.05	1228,0	1228,0
pr = 0.7	pop = 10 (ger = 25K)	1228,0	1228,0
pm = 0.05	pop = 50 (ger = 5K)	1228,0	1228,0
tsize = 2	pop = 100 (ger = 2.5K)	1228,0	1228,0

Como temos vindo a concluir, este conjunto de dados não nos traz qualquer segurança em termos de conclusões.

Ficheiro n012

Ficheiro n012.txt			
		Recombinação uniforme + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	1000,0	992,0
pm = 0.01	pr = 0.5	1000,0	991,3
tsize = 2	pr = 0.7	1000,0	996,3
pop = 100 (ger = 2500)	pm = 0.0	1000,0	882,5
pop = 100	pm = 0.001	1000,0	977,9
pr = 0.7	pm = 0.01	1000,0	991,4
tsize = 2	pm = 0.05	1000,0	993,7
pr = 0.7	pop = 10 (ger = 25K)	1000,0	978,1
pm = 0.05	pop = 50 (ger = 5K)	1000,0	989,5
tsize = 2	pop = 100 (ger = 2.5K)	1000,0	996,8

Com a melhor solução possível encontrada em todas as combinações temos as seguintes considerações:

- Confirma-se que neste tipo de recombinação o aumento da probabilidade de recombinação melhora os resultados;
- A probabilidade de mutação é mesmo muito importante;
- “Não importa” o número de gerações, o tamanho da população fornece melhores resoluções.

Ficheiro n030

Ficheiro n030.txt			
		Recombinação uniforme + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	5194,0	5044,4
pm = 0.01	pr = 0.5	5194,0	5043,1
tsize = 2	pr = 0.7	5194,0	4663,7
pr = 0.7	pm = 0.1	5194,0	4696,9
tsize = 2	pm = 0.05	5194,0	4467,4
pr = 0.7	pop = 10 (ger = 25K)	5194,0	5102,8
pm = 0.05	pop = 50 (ger = 5K)	5194,0	4536,3
tsize = 2	pop = 100 (ger = 2.5K)	5194,0	4384,8
Melhores Indicadores	tsize= 10	5194,0	5069,1

Mais uma vez, sem qualquer surpresa, são atingidas as melhores soluções possíveis em todas as combinações.

As principais conclusões foram:

- Continuamos a ter uma influência negativa do aumenta da probabilidade de recombinação;
- A probabilidade de mutação elevada produz soluções com uma maior qualidade;
- O aumento das gerações geraram resultados muito positivos em relação ao que já tinha sido encontrado;
- O tamanho do torneio teve a melhor solução a seguir às 25k de gerações.

Ficheiro n060

Ficheiro n060.txt			
		Recombinação uniforme + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	18276,0	17763,1
pm = 0.01	pr = 0.5	18123,0	17575,1
tsize = 2	pr = 0.7	17605,0	14560,2
pr = 0.7	pm = 0.05	14726,0	14366,9
tsize = 2	pm = 0.1	14926,0	14346,2
pr = 0.7 tsize = 2	pop = 50 (ger = 5K)	18338,0	14554,8
pm = 0.05	pop = 100 (ger = 2.5K)	14971,0	14382,9
pr = 0.3 tsize = 10 pm = 0.05	pop = 100 (ger = 2.5K)	18533,0	18234,1

Neste ficheiro, foram utilizados em quase todos os testes uma probabilidade de recombinação de 70% e o efeito foi que quase todas as soluções foram de qualidade bastante inferiores ao esperado. Valeu a solução das 5000 de gerações e a do torneio com tamanho 10.

Ao que parece, a quantidade de gerações conseguem compensar as probabilidades de recombinação altíssimas.

Ficheiro n120

Ficheiro n120.txt			
		Recombinação uniforme + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	43037,0	42142,9
pm = 0.01	pr = 0.5	42825,0	41729,5
tsize = 2	pr = 0.7	42551,0	35876,0
pr = 0.5 pop = 100	pm = 0.1	44499,0	41761,8
tsize = 2	pm = 0.05	44369,0	42784,6
pm = 0.05 pr = 0.5	pop = 50 (ger = 5K)	44400,0	42546,0
Melhores Indicadores	tsize= 10	45464,0	44729,3

Utilizando exatamente os mesmos parâmetros que nos algoritmos base e recombinação de 2 pontos de corte, foi aplicado o algoritmo de recombinação uniforme e as melhorias foram bastante visíveis, sendo que os principais a registar são:

- A probabilidade de recombinação continua a fazer danos às médias;
- A probabilidade de mutação, apesar de ter gerado uma melhor solução, baixou a qualidade das médias geradas em ambos os testes;
- Desta vez, as gerações não ajudaram a encontrar uma melhor solução;
- O tamanho do torneio gerou uma boa solução, mas o melhor de tudo foram as médias atingidas. Será que com mais torneios a solução ótima seria encontrada?

Ficheiro n240

Ficheiro n240.txt			
		Recombinação uniforme + Mutação por troca	
Parâmetros Fixos	Parâmetros a variar	Best	MBF
pop = 100 (ger = 2500)	pr = 0.3	139654,0	136682,0
pm = 0.01	pr = 0.5	137256,0	134492,9
tsize = 2	pr = 0.7	119011,0	117369,7
pr = 0.5 pop = 100	pm = 0.1	144951,0	137995,8
tsize = 2	pm = 0.05	142697,0	139028,0
pm = 0.05 PR = 0.7	pop = 50 (ger = 5K)	141579,0	120266,4
tsize = 2	pop = 100 (ger = 2.5K)	118902,0	117311,5
Melhores Indicadores	tsize= 10	147833,0	146867,0

Por último, as conclusões não podem ser muito diferentes do que já tínhamos visto anteriormente:

- Probabilidade de recombinação é importante mas não deve ser demasiado alta;

- A probabilidade de mutação nem sempre deve ser muito alta pois por vezes prejudica a qualidade das soluções;
- O aumento das gerações nem sempre melhora a qualidade;
- O aumento dos torneiro provoca uma subida da qualidade enorme nos resultados apresentados;

Infelizmente, nenhum destes algoritmos e combinações conseguiu encontrar a solução ótima deste ficheiro.

Algoritmos Híbridos

Os algoritmos híbridos são os que, como o próprio nome indica, combinam vários tipos de algoritmos para chegar a uma melhor conclusão.

Neste trabalho, fizemos 3 abordagens diferentes utilizando o trepa colinas desenvolvido no capítulo dos algoritmos de pesquisa locais:

- Utilizámos o trepa colinas para refinar a população inicial;
- Utilizámos o trepa colinas para refinar a população final;
- Utilizámos o trepa colinas para refinar a população dentro do ciclo de gerações.

Para todos os algoritmos que se seguem foram utilizados os seguintes parâmetros fixos:

- Probabilidade de recombinação: 50%;
- Probabilidade de mutação: 5%;
- Gerações: 2500;
- População: 100;
- Tamanho do torneio: 2

Trepa colinas na População Inicial

		População Inicial
n010.txt	Melhor	1228
	MBF	1228
n012.txt	Melhor	1000
	MBF	1000
n030.txt	Melhor	5194
	MBF	5090,899
n060.txt	Melhor	18364
	MBF	18116,4
n0120.txt	Melhor	44519
	MBF	42681,23
n0240.txt	Melhor	142583
	MBF	127993,84

Infelizmente, colocar o trepa colinas a refinar a população inicial não foi muito produtivo. Apesar dos resultados serem relativamente melhores que quando comparados com algoritmos bases com alta probabilidade de recombinação, a verdade é que existiram muitos outros melhores e com médias igualmente melhores.

Trepa colinas na População Final

		População Final
n010.txt	Melhor	1228
	MBF	1228
n012.txt	Melhor	1000
	MBF	1000
n030.txt	Melhor	5194
	MBF	5083,1665
n060.txt	Melhor	18447
	MBF	17523,533
n0120.txt	Melhor	44512
	MBF	43208
n0240.txt	Melhor	142668
	MBF	138118,1719

Até ao momento, este foi o algoritmo que produziu melhores resultados. Ainda que os melhores não tenham sido atingidos, as médias mantiveram-se ao nível de quando existem muitas gerações.

Na verdade, no ficheiro n060 ficámos o mais perto de sempre que conseguimos atingir com torneios de 2, o que nos faz crer que esta abordagem produz resultados muito bons e que deveria ser utilizada no futuro.

Trepa colinas no Ciclo de Gerações

		Ciclo de Gerações	Best
n010.txt	Melhor	1228	1228
	MBF	1228	
n012.txt	Melhor	1000	1000
	MBF	1000	
n030.txt	Melhor	5194	5194
	MBF	5159,899	
n060.txt	Melhor	18482	18714
	MBF	17824,779	
n0120.txt	Melhor		46647
	MBF		
n0240.txt	Melhor		155536
	MBF		

Esta abordagem é demasiado pesada para o computador disponível, pelo que apenas foi feita até ao ficheiro n060 (e mesmo assim, demorou cerca de 4 horas a terminar).
Contudo, e tal como se previa, esta é a melhor combinação dos 3 híbridos. A média do n030 subiu para valores nunca vistos e o resultado do n060 foi o melhor de sempre em torneios de 2. O ideal seria subir o número de torneios e de gerações para atingir a tão ambicionada melhor média mas, como referido, o tempo necessário de processamento é demasiado grande.

Conclusão

As principais conclusões deste trabalho foram tiradas ao longo da análise dos resultados, portanto é aconselhada a leitura dos mesmos para percepção total do que foi analisado.
Em termos académicos, este trabalho serviu para se perceber a dificuldade que muitas vezes estamos sujeitos na análise de dados e na falta de capacidade de processamento que temos nos nossos computadores domesticos.
Foi, sem dúvida, um dos trabalhos mais interessantes que já realizei em ambiente académico mas também um dos mais desafiantes.
Fica a sugestão de ser dado mais tempo para que os testes e os algoritmos explorados sejam outros, de forma a conseguirmos “sair fora da caixa” num trabalho tão interessante quanto este.