· U  C ·

Computação Evolucionária

2014/2015

Projectos

Ernesto Costa

1 Introdução

Este texto destina-se a apresentar os vários projectos práticos, a realizar fora das aulas, no âmbito da avaliação da disciplina de Computação Evolucionária do Mestrado em Engenharia Informática. Os trabalhos são de grupo, sendo que cada um é formado por **duas** pessoas. Os grupos devem indicar, por ordem de preferência, **três** projectos a que se candidatam, indiciando de modo claro o **código** do trabalho e o **título**. A data limite para a escolha é **12 de Abril de 2015**, finda a qual a decisão de afectação dos projectos será tomada pelo docente, que procurará respeitar o ordenamento indicado. Os desempates serão feitos por escolha aleatória. O resultado será divulgado até **19 de Abril**. Cada grupo fará o seu trabalho de modo **autónomo**, devendo a entrega ser feita até **05 de Junho de 2014**, através de correio electrónico para ernesto@dei.uc.pt[1]. A qualidade do trabalho será avaliada em função do valor intrínseco do que foi realizado, isto é, da metodologia, dos resultados a que se chegou e da avaliação crítica desses resultados. Também terá em linha de conta o valor do documento descritivo do trabalho e a sua eventual defesa pública. O documento pode ser escrito em português ou em inglês.

O documento tem uma dimensão **máxima** de 15 páginas, podendo no entanto em casos excepcionais e justificados (por exemplo, um número elevado de tabelas e gráficos) ter este limite ultrapassado. Será facultado um modelo para o documento que ajudará a perceber o que se pretende. Tenha em atenção que se trata apenas de um modelo e não de um colete de forças que está obrigado a seguir. Para além do documento, deve ainda ser entregue o programa que foi utilizado, com a indicação clara do modo como pode ser **utilizado**, e ainda os **ficheiros com os dados** que foram usados para obter os resultados estatísticos apresentados de modo a ser possível reproduzir os resultados.

Caso se justifique haverá lugar a uma apresentação/defesa do trabalho, com a presença dos elementos do grupo, em data a combinar, mas nunca depois da data do exame normal. Sendo um trabalho de grupo, admite-se uma divisão de tarefas, devendo no entanto ser claro a referência a quem fez o quê. A nota final não é para o grupo mas para cada aluno(a) individualmente.

Existem dois tipos de trabalhos. Um primeiro, que se destina a comparar diferentes algoritmos evolucionários, procurando determinar, com base num

[1] Uma versão em papel do trabalho pode, **em complemento**, ser entregue na Secretaria do DEI

estudo estatístico, qual a melhor solução de entre várias variantes. Neste tipo de trabalho a escolha dos problemas de teste é secundário. Um segundo, em que o objectivo é chegar a um algoritmo evolucionário competente para resolver um dado problema específico. Aqui, será dada importância à solução proposta. Será valorizado a pesquisa por soluções já existentes e sua crítica. Em qualquer dos casos, não é determinante para a qualidade do trabalho a produção de código próprio. O que se exige é que todas as fontes sejam referenciadas.

2 Experimentação

Para a realização do trabalho o aluno pode consultar todas as fontes que julgar pertinentes, devendo no entanto indicar sempre a sua origem, sendo motivo de anulação da prova e reprovação o recurso a plágio. Este trabalho **não** se destina a avaliar competências no domínio da programação, pelo que o estudante é livre de utilizar código fornecido pelo docente, ou código feito por si ou mesmo código de outros autores. Uma vez mais apenas se exige a indicação da origem.

Para a realização do seu trabalho deve ter em atenção um conjunto de aspectos importante, que podem ter mais ou menos importância em função do tipo de trabalho escolhido:

- descrever a arquitectura genérica do(s) algoritmo(s) usado(s);
- descrever a solução a testar em termos de representação, operadores de variação e função de mérito;
- descrever de modo claro a experiência, indicando numa tabela os parâmetros usados;
- em função do problema que lhe é colocado, deve determinar sobre o que incide a análise estatística: qualidade do resultado, eficiência, eficácia, medida de diversidade ou outro aspecto;
- o número **mínimo** de execuções por cada situação é de 30;
- na execução de ordem n todas as alternativas devem usar a mesma população inicial;

- deve ter o cuidado de promover uma comparação justa, por exemplo, garantindo que as versões em análise fazem o mesmo número de avaliações dos indivíduos. Dito de outro modo, o produto entre a dimensão da população e o número de gerações deve ser sempre o mesmo;
- não é obrigado a usar os problemas de teste que vêm na parte final deste texto. Nesse caso deve indicar as razões porque optou por outro problema;

Para além de tudo o que foi dito acima, não deve esquecer que o fundamental do trabalho é: (1) fazer uma análise estatística correcta (que inclui a escolha do método apropriado), (2) discutir de modo informado os resultados obtidos e (3) evidenciar as vantagens da solução escolhida.

3 Problemas

Segue-se uma descrição dos diferentes problemas. Mais uma vez: Não se esqueça de indicar o código e o nome das suas preferências.

3.1 TP1- Operadores de Variação Variáveis

Descrição Sumária

Desde sempre foi objecto de discussão a importância relativa dos operadores de variação, em particular a mutação e a recombinação. Normalmente, escolhidos os valores das probabilidades de recombinação e de mutação, estas são mantidas constantes ao longo da experiência. No entanto há quem defenda que a recombinação é mais importante no início e a mutação mais importante no final. Vamos procurar tirar isso a limpo. Use um dos problemas de referência para fazer o estudo, justificando a opção que efectuou.

Objectivos

Faça uns testes preliminares que lhe permita definir valores aceitáveis para os diferentes parâmetros do seu AG. Testes três versões do algoritmo genético para este problema. Uma, com probabilidades fixas, por exemplo a probabilidade de cruzamento igual a 0.9 e a de mutação igual a 0.01. Outra, com uma probabilidade fixa de cruzamento de 0.8 durante as primeiras 70% de gerações, zero nas seguintes, enquanto a probabilidade de mutação é zero durante as primeiras 70% gerações, passando para 0.05% nas restantes. A terceira versão, com probabilidades variáveis, com a probabilidade de recombinação a descer de 0.9 para 0.7, 0.5 e 0.3, ao mesmo tempo que a de mutação

vai subindo de 0.01 para 0.05, 0.1 e 0.2. Neste último caso temos então quatro pares de valores *(prob. cruzamento, prob. mutação)*. Defina o modo como esta variação é feita (por exemplo, após um certo número de gerações).

Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado e pela rapidez com que foi encontrado o melhor resultado. Analise estatisticamente os resultados e tire conclusões.

3.2 TP2- Selecção dos Pais

Descrição Sumária

Muitos defendem que manter a diversidade da população ao longo do processo evolutivo é fundamental para evitar o problema da convergência prematura. Nas aulas discutimos como os mecanismos de **selecção dos pais** podem favorecer a diversidade e, por isso, a qualidade do resultado final. Use um dos problemas de referência para fazer o estudo, justificando a opção que efectuou.

Objectivos

Faça uns testes preliminares que lhe permita definir valores aceitáveis para os diferentes parâmetros do seu AG. Use três versões do algoritmo genético. Uma, com seleção por roleta, outra, com seleção por torneio, e uma terceira com selecção por amostragem estocástica.
Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Recolha também os valores da diversidade ao longo das gerações. Não se esqueça de indicar como calculou este valor. Analise estatisticamente os resultados e tire conclusões.

3.3 TP3- Selecção dos Sobreviventes

Descrição Sumária

Muitos defendem que manter a diversidade da população ao longo do processo evolutivo é fundamental para evitar o problema da convergência prematura. Vamos estudar em que medida os mecanismos de **selecção dos sobreviventes** podem influenciar a diversidade e, como consequência, a qualidade do desempenho do algoritmo genético. Vamos procurar tirar isso a limpo.

Use um dos problemas de referência para fazer o estudo, justificando a opção que efectuou.

Objectivos

Faça uns testes preliminares que lhe permita definir valores aceitáveis para os diferentes parâmetros do seu AG. Use três versões do algoritmo genético. Uma, com seleção geracional, outra, com seleção de estado estável, e uma terceira com selecção com elitismo. Determine empiricamente valores razoáveis para os progenitores que são mantidos intocáveis.
Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Recolha também os valores da diversidade ao longo das gerações. Analise estatisticamente os resultados e tire conclusões.

3.4 TP4- Operadores de Recombinação

Descrição Sumária

Muitos defendem que manter a diversidade da população ao longo do processo evolutivo é fundamental para evitar o problema da convergência prematura. Vamos estudar em que medida os **operadores de recombinação** podem influenciar a diversidade e, como consequência, a qualidade do desempenho do algoritmo genético. Vamos procurar tirar isso a limpo. Use um dos problemas de referência para fazer o estudo, justificando a opção que efectuou.

Objectivos

Faça uns testes preliminares que lhe permita definir valores aceitáveis para os diferentes parâmetros do seu AG. Use três versões do algoritmo genético. Uma, recombinação de um ponto, outra, com recombinação de dois pontos, e uma terceira com recombinação uniforme.
Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Recolha também os valores da diversidade ao longo das gerações. Analise estatisticamente os resultados e tire conclusões.

3.5 TP5- Operadores de Mutação em Programação Genética

Descrição Sumária

Pretende-se estudar em que medida os **operadores de mutação** podem influenciar a qualidade do desempenho do algoritmo de **programação genética**. Vamos procurar tirar isso a limpo. Use um problema de referência adequado.

Objectivos

Faça uns testes preliminares que lhe permita definir valores aceitáveis para os diferentes parâmetros do seu algoritmo GP. Use duas versões do algoritmo: uma, com mutação de sub-árvore, e, a outra, com mutação por nó da árvore. Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.6 TP6- Auto-Adaptação em Estratégias Evolutivas

Descrição Sumária

Fazer variar os parâmetros de uma experiência ao longo da simulação parece dar bons resultados. Por exemplo, o tamanho da população ou as probabilidades dos operadores de variação. Podemos incluir na nossa implementação o modo como isso é feito. Mas melhor seria deixar o processo evolutivo tratar desse assunto. É isso que vamos estudar, no contexto das **estratégias evolutivas**, variante dos algoritmos evolucionários.

Objectivos

Como sabe, as EE permitem incluir na representação das soluções os parâmetros estratégicos, como os valores dos desvios padrão da distribuição normal que usamos para efectuar a mutação de cada gene. Escolha uma implementação em Python das estratégias evolutivas, do tipo $(\mu + \lambda)$. Use como problema de referência para fazer o seu estudo, um dos problemas de optimização de funções. Ver descrição dos problemas possíveis na secção **??**. Use duas versões do algoritmo, uma, sem e outra com auto-adaptação.
Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.7 TP7- *Bloat* em Programação Genética

Descrição Sumária

Um dos grandes problemas da **programação genética** consiste no crescimento sem controlo dos indivíduos da população sem que isso se traduza por melhoria do desempenho. Chama-se a esse fenómeno *bloat*. Têm vindo a ser propostos diversos mecanismos para lidar com esta questão. Vamos estudar alguns envolvendo apenas a profundidade da árvore.

Objectivos

Faça uns testes preliminares que lhe permita definir valores aceitáveis para os diferentes parâmetros do seu AG. Use três versões do algoritmo de programação genética. Uma, sem limite de profundidade, outra com limite de profundidade máxima fixo e, a terceira, com profundidade limitada mas permitindo excepções no caso de os filhos gerados serem melhores do que os pais. Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Recolha também os valores da profundidade média dos indivíduos ao longo das gerações. Analise estatisticamente os resultados e tire conclusões.

3.8 TP8- População Inicial

Descrição Sumária

Quando falámos de métodos que usam apenas um indivíduo, como o trepa colonas, referimos a questão da escolha do ponto inicial de procura. No caso dos métodos baseados numa população de soluções candidatas temos optado por criar uma população aleatoriamente com base numa distribuição uniforme. Mas será que esta é mesmo a melhor maneira? Vamos tirar isso a limpo.

Objectivos

Escolha um problema que envolva a procura do óptimo de uma função. Vários exemplos são dados na secção sobre problemas de teste. De seguida compare dois métodos de criação da população inicial. O primeiro, o clássico acima definido: gerar aleatoriamente usando uma distribuição uniforme. O segundo, baseia-se na divisão do espaço de procura em hiper volumes semelhantes e em escolher um número igual de indivíduos em cada um dos hiper volumes.

Para cada um dos dois métodos faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado e pela eficiência da procura, isto é, pelo momento em que a solução óptima é encontrada. Analise estatisticamente os resultados e tire conclusões.

3.9 TP9- Indivíduos inviáveis

Descrição Sumária

Muitos problemas de optimização estão sujeitos a restrições. Para uma dada representação a escolha dos operadores de variação pode ser tal que seja possível gerar indivíduos inviáveis. Neste caso existem pelo menos duas grandes alternativas para lidar com o problema. Uma, consiste em penalizar os indivíduos inviáveis por forma a que a sua qualidade seja baixa. A outra. baseia-se num mecanismo de reparação que torne o indivíduo viável. Qual destas duas alternativas será a melhor? Vamos ver.

Objectivos

Escolha um problema com restrições. Por exemplo, o problema da mochila em que a capacidade do saco é a restrição. Implemente duas variantes de um algoritmo evolucionário, uma para cada um dos métodos indicados. Para cada algoritmo faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.10 TP10- Operadores de mutação para permutações

Descrição Sumária

Existem problemas, como é o caso das N-rainhas e do Caixeiro Viajante, em que a representação natural é uma permutação de inteiros. O operador de mutação mais comum é o que troca dois elementos. Mas existem outros, como são o operador de inversão e o de deslocamento. O primeiro, seleciona dois pontos aleatoriamente no indivíduo e inverte a ordem dos elementos ente esses dois pontos. O segundo, escolhe um sub-caminho e desloca-o para outra posição. Será que a escolha do operador de mutação é relevante? Vamos tirar isso a limpo.

Objectivos

Escolhido o problema, implemente três versões de um algoritmo evolucionário genérico em que o operador de mutação é cada um dos três acima referido. Para cada um dos três métodos faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.11 TP11- Operadores de mutação para reais

Descrição Sumária

O operador de mutação mais frequente para representações reais é o que modifica um gene juntando-lhe um valor amostrado de uma distribuição gaussiana. Existem outras alternativas, como por exemplo escolher um valor completamente novo de entre a gama de valores possível por recurso a uma distribuição uniforme, ou, ainda, usar um operador de mutação não uniforme que a seguir se define. Admitamos que vai haver lugar a mutar o gene x_k. Então, o novo valor x'_k é dado por:

$$x'_k = \begin{cases} x_k + \Delta(t, S_k - x_k) & \text{se} \quad choice(0,1) = 0 \\ x_k - \Delta(t, x_k - I_k) & \text{se} \quad choice(0,1) = 1 \end{cases} \tag{1}$$

onde S_k é o limite superior e I_k o limite inferior de x_k e $\Delta(t, y)$ uma função que produz um valor no intervalo $[0, y]$ de modo não uniforme, sendo que a probabilidade de o valor devolvido ser próximo de zero aumenta à medida que t aumenta. t é o contador de gerações. Esta função é dada por:

$$\Delta(t, y) = y \times r \times (1 - \frac{t}{T})^b$$

Agora, r é um número aleatório entre 0 e 1, T o número máximo de gerações, e b um parâmetro do sistema que determina o grau de não uniformidade. Optar por um ou por outro será relevante? Vamos analisar.

Objectivos

Escolha uma das funções dadas na secção **??**. Implemente as duas versões do algoritmo, uma para cada operador de mutação. Para cada um dos dois métodos faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.12 TP12- Operadores de cruzamento para reais (I)

Descrição Sumária

Ao longo do tempo foram aparecendo propostas alternativas aos operadores de cruzamento básico, para representações com vectores de reais, clamando pela sua superioridade. É isso que vamos averiguar. Um dos operadores propostos designa-se por **cruzamento aritmético** e pode ser descrito assim. Suponha dois progenitores (isto é , vectores de reais), x_1 e x_2 que pretendemos cruzar para obter dois filhos. Eles são criados da seguinte maneira:

$$\begin{cases} x'_1 = a \times x_1 + (1-a) \times x_2 \\ x'_2 = a \times x_2 + (1-a) \times x_1 \end{cases} \tag{2}$$

com a um número real no intervalo [0..1]. A operação é feita, obviamente, componente a componente.

Objectivos

Escolha uma das funções dadas na secção **??**. Implemente as duas versões do algoritmo, uma para cada operador de cruzamento. Para cada um dos dois métodos faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.13 TP13- Operadores de cruzamento para reais (II)

Ao longo do tempo foram aparecendo propostas alternativas aos operadores de cruzamento básico, para representações com vectores de reais, clamando pela sua superioridade. É isso que vamos averiguar. Um dos operadores propostos designa-se por **cruzamento heurístico** e pode ser descrito assim. Suponha dois progenitores (isto é , vectores de reais), x_1 e x_2 que pretendemos cruzar para obter **um** filho. Ele é criado da seguinte maneira:

$$x_f = r \times (x_2 - x_1) + x_2$$

com r um número real aleatório no intervalo [0..1]. A operação é feita, obviamente, componente a componente. Este operador introduz um enviesamento no sentido do pais de melhor qualidade, pois obriga a que a qualidade de x_2 não seja inferior à de x_1, num problema de maximização, ou o inverso num problema de minimização.. Deve ter ainda em atenção a possibilidade de gerar indivíduos inviáveis. Caso o descendente seja inviável gera-se novo

valor de r. A operação repete-se um número pré determinado de vezes. Ultrapassado esse valor, **nenhum** descendente é gerado.

Objectivos

Escolha uma das funções dadas na secção **??**. Implemente as duas versões do algoritmo, uma para cada operador de cruzamento. Para cada um dos dois métodos faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.14 TP14- Operadores de cruzamento para permutações

Descrição Sumária

Ao longo do tempo foram aparecendo propostas alternativas aos operadores de cruzamento básico, para representações baseadas em permutações de inteiros, clamando pela sua superioridade. É isso que vamos averiguar. Vamos considerar dois exemplos: PMX e OX. Ambos funcionam começando por escolher aleatoriamente dois pontos de corte. Trocamos entre ambos a parte do meio, gerada pelos dois pontos de corte. No caso do PMX, procura-se preservar o máximo de valores de origem sem que se viole a permutação. Os restantes valores são obtidos pela relação existente entre os elementos de cada indivíduo na parte do meio. Exemplo:

$$\begin{cases} p_1 = (123|4567|89) & \Rightarrow o_1 = (423|1876|59) \\ p_2 = (452|1876|93) & \Rightarrow o_2 = (182|4567|93) \end{cases} \tag{3}$$

No caso do OX começa-se a partir do segundo ponto de corte a tentar manter a ordem relativa dos elementos. Exemplo.

$$\begin{cases} p_1 = (123|4567|89) & \Rightarrow o_1 = (345|1876|92) \\ p_2 = (452|1876|93) & \Rightarrow o_2 = (218|4567|93) \end{cases} \tag{4}$$

Objectivos

Escolha uma das funções dadas na secção **??**. Implemente as duas versões do algoritmo, uma para cada operador de cruzamento. Para cada um dos dois métodos faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Analise estatisticamente os resultados e tire conclusões.

3.15 TP15- Ambientes Dinâmicos

Descrição Sumária

Os problemas do mundo real são muitas vezes dinâmicos. Isto faz com que o óptimo num dado instante não seja necessariamente o mesmo no instante seguinte. Ao longo do tempo têm sido propostos vários métodos para lidar com o problema da adaptação do algoritmo face à mudança. Vamos estudar dois desses mecanismos e analisar o seu comportamento. Vamos usar a variante dinâmica do problema da mochila (ver descrição na secção **??**).

Objectivos

Escolha uma implementação de um algoritmo genético em Python, adequado a este tipo de problema. Pode usar o que foi utilizado nas aulas, mas neste caso vai ter que o alterar por forma a ter duas versões que permitam o seguinte. Num primeiro caso, sempre que o óptimo mudar a probabilidade de mutação deve ser fortemente aumentada durante algumas gerações, e assim permanecer até voltar ao seu valor normal. Chama-se a este método **hipermutação**. Num segundo caso, deve prever um mecanismo de **memória**, onde são guardados os melhores elementos obtidos para uma dada situação. Sempre que o ambiente mude deve ir à memória buscar um elemento adequado à nova situação e guardar nesta elementos que eram bons para a situação anterior.

Para cada um dos dois algoritmos faça 30 testes (*runs*). Recolha os dados sobre desempenho de cada algoritmo, medido pela qualidade do resultado. Dada a natureza do problema é importante que avalie sobretudo a capacidade de recuperação de cada um dos algoritmos face à mudança. Analise estatisticamente os resultados e tire conclusões.

3.16 TP16- Inserção de Emigrantes

Descrição Sumária

Muitos defendem que manter a diversidade da população ao longo do processo evolutivo é fundamental para evitar o problema da **convergência prematura** e melhorar a qualidade do resultado final. Vamos estudar em que medida a inclusão de **novos indivíduos** ao longo da execução pode influenciar a diversidade e, como consequência, a qualidade do resultado final. Vamos tirar isso a limpo. Use um dos problemas de referência para fazer o estudo, justificando a opção que efectuou.

Objectivos

Faça uns testes preliminares que lhe permita definir valores aceitáveis para os diferentes parâmetros dos seu algoritmo evolucionário. Use três versões do algoritmo. Uma primeira, sem alterações do algoritmo de base, uma segunda, em que a cada iteração uma percentagem reduzida de elementos novos gerados aleatoriamente é introduzida na população de sobreviventes substituindo os de menor qualidade, e uma terceira, em que os elementos a introduzir são mutações simples do melhor elemento da população corrente, substituindo de novo os de pior qualidade. Recolha os dados sobre o desempenho, medida pela qualidade final. Analise estatisticamente os resultados e tire conclusões.

3.17 TP17- Dois melhor do que um: múltiplas populações

Descrição Sumária

Existe quem defenda que é melhor usar múltiplas populações num algoritmo evolucionários para resolver um dado problema. Aqui pretendemos fazer um pouco diferente. Começando com uma mesma população, usamos dois algoritmos evolucionários diferentes para resolver o mesmo problema. De tempos a tempos promovemos a migração de soluções entre os algoritmos. Queremos saber se o resultado é melhor do que usar apenas um algoritmo no qual, com a mesma frequência com que há migração no outro caso, se introduzem elementos aleatórios na população.

Objectivos

Escolha uma implementação em Python de um algoritmo evolucionário genérico e produza duas variantes: uma, com probabilidade de cruzamento normal e probabilidade de mutação muito baixa; outra, com probabilidade de cruzamento muito baixa e probabilidade de mutação normal. Escolha um problema e corra em paralelo os dois algoritmos. De tempos a temos promove a migração de elementos entre populações. Vai analisar a importância da frequência com que se efectua a migração e a quantidade de indivíduos que migram, para a qualidade do resultado. Pode assumir que se trata de um processo de troca simples. Finalmente, compara a melhor solução com o resultado de usar um algoritmo evolucionário *normal*, ou seja, em que usa as duas probabilidades de cruzamento e de mutação convencionais.

3.18 TP18- Atribuição de projectos a alunos(as)

Descrição Sumária

Na aula foi referido de forma breve a questão de atribuir projectos a alunos que têm que indicar várias opções (e.g., três) de forma ordenada. A ideia é que todos os estudantes fiquem satisfeitos com o resultado da atribuição por esta respeitar ao máximo as suas preferências. Vamos efectuar um estudo que permita determinar que soluções já existem propostas e como é que um algoritmo evolucionário pode resolver este problema.

Objectivos

Defina uma arquitectura evolucionária para procurar a solução. Descreva o problema de forma rigorosa, incluindo a representação, os operadores de variação e a função de qualidade. Admita que o número de projectos é **maior ou igual** ao número de estudantes. Faça experiências que lhe permitam escolher, de forma justificada, a melhor alternativa para os valores dos parâmetros a usar.

3.19 TP19- Descoberta de Leis Físicas

Descrição Sumária

As leis de Kepler descrevem o movimento dos planetas em torno do Sol. A terceira lei diz que o quadrado do período é proporcional ao cudo da distância de metade do semi-eixo maior da órbita do planeta:

$$P^2 = kD^3$$

Pretende-se mostrar como um algoritmo evolucionário pode ser usado para re-descobrir esta lei (ou outra do mesmo tipo...)

Objectivos

A tabela abaixo (ver **??**) mostra os valores que relacionam o período (em anos) com a distância (em unidades de $10^{10}m$).

Escolha a melhor alternativa de algoritmo evolucionário que lhe permita encontrar a terceira lei de Kepler, incluindo o valor da constante k. Faça uma pesquisa sobre solucções para este tipo de problemas e discuta as suas vantagens e inconvenientes.

Planeta	Distância	Período
Mercúrio	5.79	0.241
Vénus	10.8	0.615
Terra	15.0	1.0
Marte	22.8	1.88
Júpiter	77.8	11.9
Saturno	143	29.5
Urano	297	84
Neptuno	450	165

Tabela 1: Planetas: distâncias e períodos

3.20 TP20- Sudoku: uma solução evolucionária

Descrição Sumária

O problema do Sudoku é suficientemente conhecido. Em que medida pode ser resolvido por técnicas evolucionárias? Vamos tirar isso a limpo.

Objectivos

Pretende-se fazer uma levantamento das soluções evolucionárias para o problema. Implemente uma solução e estude o seu desempenho. Será valorizado o seu contributo pessoal para a solução.

3.21 TP21- Doenças do coração

Descrição Sumária

Existem um conjunto de indicadores que permitem determinar se uma dada pessoa tem ou não um problema cardíaco. Vamos tentar descobrir um classificador binário que nos permita classificar casos concretos de pacientes. O classificador vai ser evoluído por um algoritmo evolucionário.

Objectivos

Foram colocados na InforEstudante um conjunto de exemplos de pessoas que em função de 13 indicadores precisos determinam se a pessoa tem ou não um problema cardíaco. Desenvolva uma arquitectura evolucionária que lhe permita construir o classificador binário a partir dos exemplos fornecidos. Analise a precisão do classificador.

4 Problemas de Teste

Apresentamos alguns problemas de teste clássicos (*benchmark problems*), para algoritmos estocásticos de procura global. Podem por isso ser usados para os diferentes tipos de algoritmos evolucionários, objecto deste trabalho, mas também para outro tipo de metaheurísticas, como PSO, ACO e afins que a seu tempo serão dados nas aulas. Este tipo de problemas permite testar a qualidade dos algoritmos que usamos e, ao mesmo tempo, facilitam a comparação de resultados entre diferentes técnicas. Como é evidente os problemas aqui presentes não esgotam as possibilidades. Para cada exemplo procura-se dar a sua caracterização, sendo que estão agrupados segundo critérios simples. Separados por técnicas, optimização de funções mono e multiobjectivo e por programação genética.

Se o enunciado do trabalho não fizer menção explícita a nenhum deles, é livre de escolher o que achar mais adequado ao trabalho, ou outro que encontre nas suas pesquisas. Neste último caso, deve no entanto obter previamente a autorização do docente para o seu uso.

4.1 *Trap*

Existem um conjunto de funções que foram concebidas para tornar a vida difícil aos algoritmos evolucionários. Um caso clássico é o da função cujo valor associado é dado pelo número de zeros no vector binário a menos que não exista nenhum, caso em que o mérito é máximo e igual ao comprimento do vector mais um. Pode ser descrito pela equação **??**.

$$f(\langle x_1, \ldots, x_n \rangle) = (n - \sum_{i=1}^{n} x_i) + (n+1) \prod_{i=1}^{n} x_i \qquad x_i \in \{0, 1\} \qquad (5)$$

4.2 SAT: Satisfiabilidade de uma fórmula booleana

Este problema, conhecido na literatura por SAT, consiste em determinar qual o valor de verdade de cada uma das variáveis de uma fórmula booleana que resulte no valor de verdade global de *True*. Sem perda de generalidade a fórmula pode ser considerada na forma normal conjuntiva. Abaixo mostra-se um exemplo de uma fórmula na FNC com quatro variáveis booleanas.

$$(x_1 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$$

Como se pode ver por este exemplo, pode existir mais do que uma solução. De um modo geral, trata-se de problema muito complexo e as soluções muito poucas, quando o número de variáveis é muito grande. Basta pensar que para 100 variáveis o nosso de possibilidades de atribuir valores de verdade às variáveis é de 2^{100}!

4.3 Números de João Brandão

Considere a sequência de inteiros entre 1 e um dado n (por exemplo, 100). Dizemos que três números estão em conflito se um deles for a média da soma dos outros dois. Os números João Brandão para um dado n, é o maior conjunto de inteiros, retirados da sequência de 1 a n, para a qual não existe nenhum trio de números em conflito.

4.4 Caixeiro Viajante - *Travelling Salesman Problem*

Este problema já foi discutido nas aulas. Um vendedor pretende visitar de modo óptimo várias cidades. À excepção da cidade de onde parte e a onde tem que regressar, as restantes cidades só podem ser visitadas uma vez. Existem várias soluções evolucionárias para o problema. Para o problema ser interessante é preciso usar um número razoável de cidades.

4.5 Mochila - *Knapsack Problem*

Outro problema clássico. Na sua formulação mais simples, existe uma mochila com uma dada capacidade, e um conjunto de n objectos x_i com um dado peso w_i e um certo valor v_i. Queremos maximizar a soma do valor dos objectos que colocamos na mochila, não podendo no entanto violar a capacidade da mochila C. Mai formalmente temos um problema de optimização definido por:

$$max(\sum_{i=1}^{n} v_i \times x_i)$$

sujeito a

$$\sum_{i=1}^{n} w_i \times x_i \leq C$$

com

$$x_i \in \{0, 1\}$$

Na variante **dinâmica**, a capacidade da mochila vai variando ao longo das gerações, seja de modo periódico, seja de modo não periódico. A escolha dos itens, dos seus pesos e restantes características deve ser feita de modo criterioso. Existem basicamente três modos de gerar os pesos e os valores.

Não correlacionados

Os pesos w_i, e os respectivos valores v_i de cada item são obtidos a partir de uma distribuição uniforme no intervalo $[1..v]$, isto é:

w_i = uniform([1..v])
v_i= uniform([1..v])

Fracamente correlacionados

w_i = uniform([1..v])
$v_i = w_i$ + uniform([-r..r])

Nota: não são permitidos valores negativos, pelo que a geração deve ser repetida se tal acontecer até que o valor $w_i > 0$.

Fortemente correlacionados

w_i = uniform([1..v])
$v_i = w_i$ + r

Quanto mais correlacionados forem os dados mais difícil é o problema!

4.5.1 Parâmetros

Usar como parâmetros:

- $v = 10$
- $r = 5$
- Número de itens $n \in \{100, 250, 500\}$

A capacidade da mochila pode ser calculada basicamente de dois modos distintos:
$C = 2 \times v$
$C = 0.5 \times \sum_{i=1}^{n} w_i$

4.6 Soma de Subconjuntos de Inteiros

Este problema é um caso particular do problema da mochila. Dado um conjunto de n inteiros distintos, $a_1, a_2, \ldots, a_n$, não necessariamente consecutivos, encontrar um subconjunto desses inteiros, logo sem repetições, cuja soma é igual a um dado X. Admita que pretende a solução de menor cardinalidade. Por exemplo, se tivermos o conjunto $\{5, 8, 4, 11, 6, 12\}$ e $X = 20$ então temos as soluções $\{8, 12\}$ e $\{4, 5, 11\}$, sendo a primeira a que nos interessa.

4.7 Réguas de Golomb

Informalmente as réguas de Golomb caracterizam-se por podermos medir mais comprimentos discretos (inteiros) do que as marcas que contém. A particularidade das RG é a de que as diferenças entre todos os pares de marcas são únicas. Uma régua de Golomb diz-se **óptima**, e designa-se por OGR, se o seu comprimento for o menor para um dado número de marcas. Podem existir várias OGR para um determinado número de marcas. Uma OGR diz-se **perfeita** se for possível medir todos os comprimentos inteiros de 1 até ao comprimento da régua. A figura **??** mostra uma RG perfeita com 4 marcas e comprimento 6.

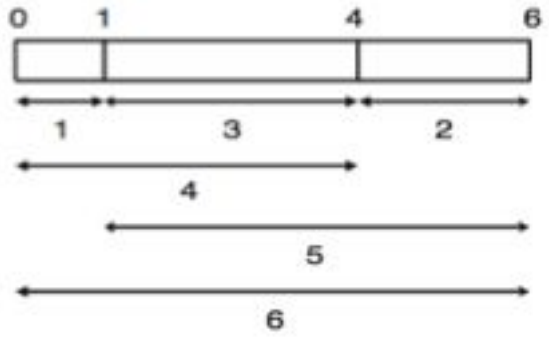

Figura 1: Régua de Golomb perfeita

As réguas de Golomb podem ser representadas de diferentes maneiras. Em abstracto, são um vector $< a_1, a_2, \ldots, a_n >$, com a_i a representar uma marca. No caso da figura **??** teríamos $< 0, 1, 4, 6 >$. Por convenção a_1 é sempre colocada na posição 0, enquanto que a_n representa o comprimento da régua.

4.8 Regressão Simbólica

Exemplo típico da programação genética. Queremos aproximar um polinómio de um certo graus, partindo de um dado conjunto de treino, baseado em pares /(entrada,saída). São exemplo de polinómios:

$$x^4 + x^3 + x^2 + x \tag{6}$$
$$x^2 + x + 1 \tag{7}$$

Num contexto de programação genética as condições da experiência são as dadas na tabela **??**.

Característica	Descrição
Objectivo	A melhor aproximação de $x^4 + x^3 + x^2 + x$ para $x \in [-1, 1]$
Conjunto de Funções	$+, -, *$ e divisão protegida
Conjunto de Terminais	variável x (não há constantes)
Mérito	Erro quadrático
Conjunto de teste	21 casos no intervalo $[-1, 1]$

Tabela 2: Regressão Simbólica: parâmetros

4.9 Detector de Paridade Par - *5- Even Bit Parity*

Mais um exemplo usado em programação genética. Pretende-se evoluir a solução para o problema de detectar se o número de uns numa cadeia binária, com 5 bits, é par ou não.

Característica	Descrição
Objectivo	Classificação correcta para os 2^5 casos
Conjunto de Funções	$and, or, nand, nor$
Conjunto de Terminais	variáveis x_1, x_2, x_3, x_4, x_5 (não há constantes)
Mérito	32 - número de casos bem classificados
Conjunto de teste	subconjunto dos 2^5 casos possíveis

Tabela 3: Detector de Paridade: parâmetros

4.10 Formiga Artificial

Ainda um exemplo usado em programação genética. Oobjectivo é evoluir uma estratégia para uma formiga seguir um trilho de comida. a comida encontra-se numa grelha toroidal de 32×32. O trilho não é contínuo existindo por 89 pedaços de comida. A formiga é colocada no início na posição superior esquerda e voltada para este. Tem um máximo de 400 acções para realizar o seu objectivo. A imagem **??** procura ilustrar a situação.

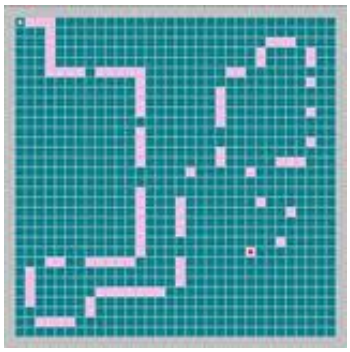

Figura 2: **Trilho de Santa Fé**

Característica	Descrição
Objectivo	Comer todos os pedaços de comida no máximo de 400 operações
Conjunto de Funções	$\{if\_food\_ahead, progn2, progn3\}$
Conjunto de Terminais	$\{left, right, move\}$ (não há varáveis)
Mérito	Número de pedaços de comida por comer

Tabela 4: Trilho de Santa Fé: parâmetros

4.11 Funções

Nesta secção apresentamos um número significativo de funções de testes. Existem muitas mais que pode procurar. Se quiser usar uma diferente das apresentadas deve questionar o docente.

Sphere (De Jong's F1 function)

A função *Sphere* faz parte do grupo de funções de teste conhecidas por funções de De Jong. É a função **F1**. Trata-se de uma função contínua, convexa,

unimodal. Tem o mínimo na origem (e máximos nas zonas extremas do domínio).

$$Min(F_1) = F_1(0, \ldots, 0) = 0$$

A versão 2D pode ser visualizada na figura **??**.

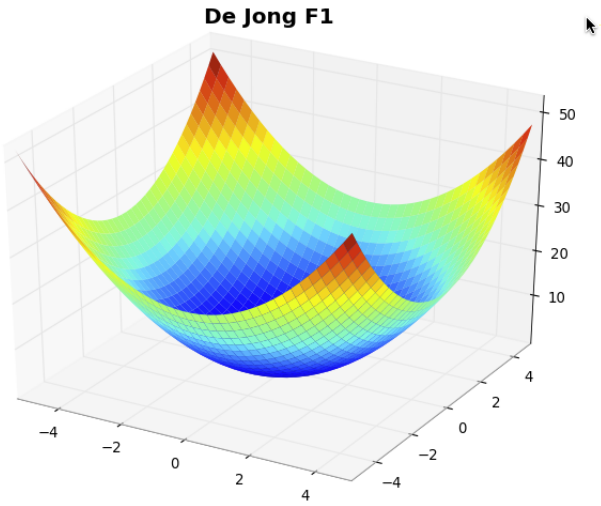

Figura 3: **De Jong F1**

A função é dada pela equação **??**.

$$f(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n} x_i^2 \qquad x_i \in [-5.12, 5.12] \tag{8}$$

4.11.1 *Rosenbrock*

Conhecida também por De Jong F2. Trata-se de uma função contínua, não convexa e unimodal. A versão 2D pode ser visualizada na figura **??**.

$$Min(F_2) = F_2(1, \ldots, 1) = 0$$

A função é dada pela equação **??**.

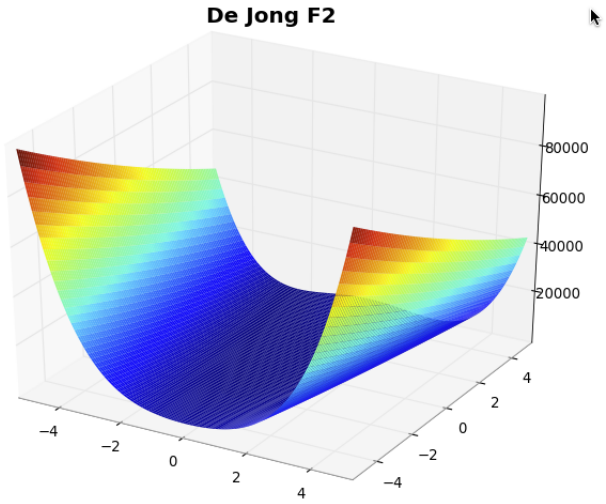

Figura 4: **De Jong F2**

$$f(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n-1} (1-x_i)^2 + 100 \times (x_{i+1} - x_i^2)^2 \qquad x_i \in [-2.048, 2.048] \quad (9)$$

4.11.2 *Step*

Conhecida também por De Jong F3. Trata-se de uma função não contínua, não convexa e unimodal. A versão 2D pode ser visualizada na figura **??**.

$$Min(F_3) = F_3(-5.12, \ldots, -5.12) = 0$$

A função é dada pela equação **??**.

$$f(\langle x_1, \ldots, x_n \rangle) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor \qquad x_i \in [-5.12, 5.12] \qquad (10)$$

4.11.3 *Quartic*

Conhecida também por De Jong F4.É uma função contínua, convexa e unimodal. A versão 2D pode ser visualizada na figura **??**.

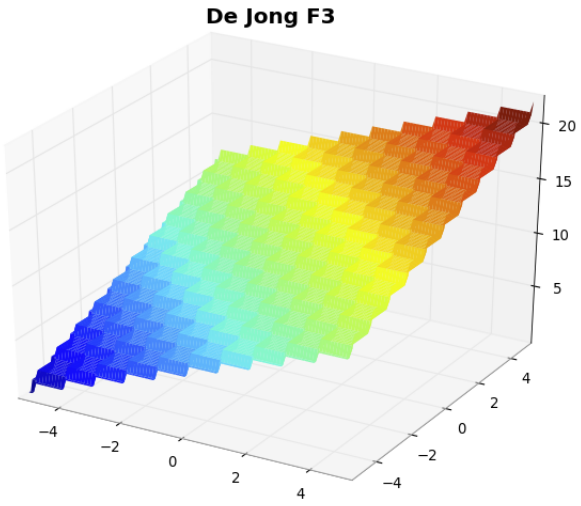

Figura 5: **De Jong F3**

$$Min(F_4) = F_4(0, \ldots, 0) = 0$$

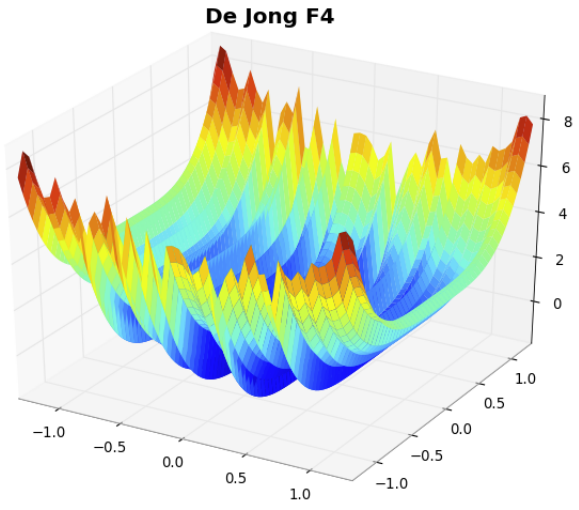

Figura 6: **De Jong F4**

A função é dada pela equação **??**.

$$f(\langle x_1, \ldots, x_n \rangle) = (\sum_{i=1}^{n} i \times x_i^4) + \mathcal{N}(0,1) \qquad x_i \in [-1.28, 1.28] \qquad (11)$$

com $\mathcal{N}(0,1)$ um ruído gaussiano de média zero e desvio padrão 1.

4.11.4 *Rastrigin*

Trata-se de uma função altamente multimodal, tendo por isso muitos mínimos locais. A versão 2D pode ser visualizada na figura **??**.

$$Min(Rastrigin) = Rastrigin(0, \ldots, 0) = 0$$

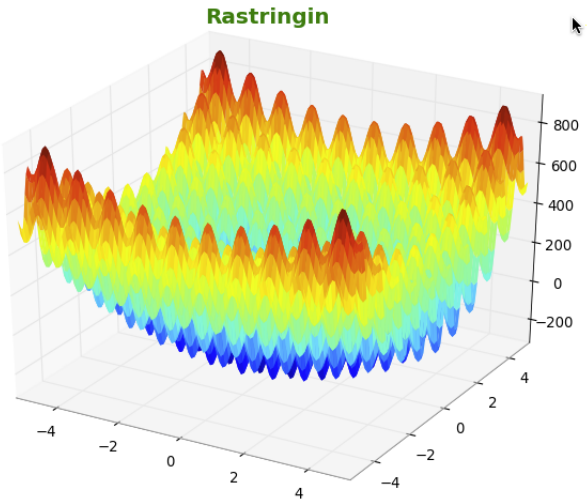

Figura 7: **Rastrigin**

A função é dada pela equação **??**.

$$f(\langle x_1, \ldots, x_n \rangle) = A \times n + (\sum_{i=1}^{n} x_i^2 - A \times cos(2\pi x_i) \qquad x_i \in [-5.12, 5.12] \quad (12)$$

com A a assumir tipicamente o valor de 10.

4.11.5 *Schwefel*

Mais um exemplo de função multimodal, com vários mínimos globais. A versão 2D pode ser visualizada na figura **??**.

$$Min(Schwefel) = Schwefel(420.9687, \ldots, 420.9687) = n \times 418.9829$$

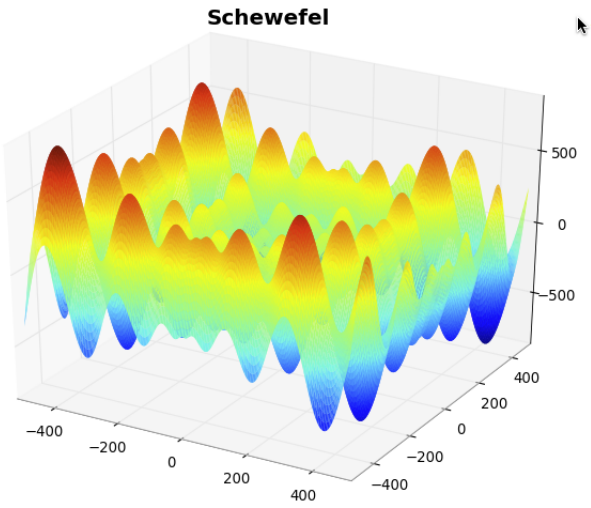

Figura 8: **Schwefel**

A função é dada pela equação **??**.

$$f(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n} -x_i \times sin(\sqrt{|x_i|}) \qquad x_i \in [-500, 500] \qquad (13)$$

4.11.6 *Griewangk*

Outro exemplo de função multimodal, semelhante à função de Rastrigin. A versão 2D pode ser visualizada na figura **??**.

$$Min(Griewangk) = Griewangk(0, \ldots, 0) = 0$$

A função é dada pela equação **??**.

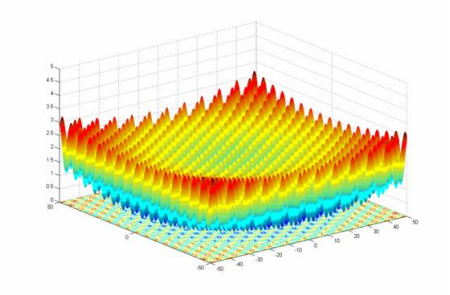

Figura 9: **Griewangk**

$$f(\langle x_1, \ldots, x_n \rangle) = 1 + \frac{1}{4000}(\sum_{i=1}^{n} x_i^2) + \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{(i)}}) \qquad x_i \in [-600, 600] \quad (14)$$

5 Conclusão

Apresentados os temas, alguns comentários finais muito curtos. Desde logo, reafirmar que os trabalhos irão sendo acompanhados ao longo das aulas (T e PL) e, ainda, durante o horário de atendimento. Em segundo lugar, dizer que procurámos que os trabalhos não necessitassem de conceitos ainda não dados, ou, no caso em que tal acontece, que o seu tratamento em aula seja feito logo a seguir à interrupção da Páscoa. Em terceiro lugar, ter o cuidado de dar trabalhos de complexidade equivalente. Em quarto lugar, dimensionar o trabalho para o tornar compatível com o seu peso na nota final.
As questões metodológicas, mais concretamente as questões que envolvem as noções de estatística relevantes para o trabalho, têm vindo a ser tratadas nas aulas. A ferramenta estatística a utilizar é da sua escolha. Finalmente, deixar claro que sendo este um trabalho baseado em simulações, exige no entanto conhecimentos da teoria, pelo que aconselhamos a leitura de alguns textos, nomeadamente [**?**, **?**, **?**].
Boa sorte!

Referências

[1] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[2] Sean Luke. *Essentials of Metaheuristics*. Lulu Press, 2011.

[3] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.