Computação Natural - TP1
Programação Genética

Guilherme Torres
Departamento de Ciência da Computação - UFMG

1 Introdução

O objetivo deste trabalho foi formular e escolher os melhores parâmetros para um algoritmo de aprendizado para regressão simbólica. Dada uma função $f : \mathbb{R}^n \to \mathbb{R}$, a regressão simbólica consiste em tentar aproximar essa função usando uma árvore constituída por terminais nas folhas (constantes ou variáveis) e operadores.

A heurística usada foi a programação genética, que usa de conceitos evolucionários para evoluir as soluções para o problema e obter indivíduos mais adaptados. É um algoritmo estocástico, ou seja, com vários componentes de aleatoriedade como geração inicial dos indivíduos e mutações, o que requer que muitos experimentos sejam realizados para tirar as conclusões sobre o trabalho, que serão discutidos nesse documento.

O algoritmo foi implementado em Python 3, na pasta src/. Instruções para executar o teste estão explicitadas no arquivo README.md. O programa também pode ser encontrado no repositório:
`https://github.com/torresguilherme/cn-tp1-gp`.

2 Implementação

2.1 Fundamentação

O funcionamento do algoritmo pode ser descrito, no plano geral, pelos seguintes passos:

1. Geração aleatória dos primeiros indivíduos;
2. Mostra a melhor, pior e média das fitness na população, usando os dados de treino;

3. Adiciona o grupo elitista à nova população;
4. Repete até que o número de indivíduos na população possa ser reposto:
   (a) Seleciona dois pais através do torneio;
   (b) Aplica a probabilidade de cruzamento;
   (c) Ao se obter um cruzamento, aplica a chance de mutação;
   (d) Adiciona os filhos gerados à nova população;
5. Repete desde o passo 2, até que o número máximo de gerações seja alcançado;
6. Avalia as soluções encontradas nos dados de teste e retorna o melhor valor.

Como é sugerido, há muitas variáveis envolvidas e deve-se encontrar o valor ótimo para elas por meio de experimentos.

2.2 Decisões de implementação

Indivíduo

Um indivíduo é representado por uma árvore, que tem um valor de uma função. Cada uma das folhas da árvore faz o papel de uma constante ou uma variável. As constantes são geradas aleatoriamente com números de ponto flutuante entre -10 e 10 (distribuição uniforme).

Os operadores escolhidos foram '+', '*' e '/', respectivamente, para adição, multiplicação e divisão. Como a subtração é praticamente uma adição por um número negativo, não foi necessário adicionar um operador '-'.

Geração aleatória de indivíduos

Cada indivíduo começa com um operador a priori, e sua árvore é estendida até que todos os caminhos acabem em um terminal. Como a profundidade máxima do indivídio é 7, apenas terminais serão escolhidos aleatoriamente quando algum nó chegar na profundidade 6.

Cálculo da Fitness

A fitness de um indivíduo é dada pelo seu erro quadrático em relação ao resultado dado para um conjunto de variáveis da função que se quer aprender.

O erro de um indivíduo é obtido por

$$\sqrt{1/N \sum_{i=0}^{n-1} (Ev_i - y_i)^2}$$

sendo N o número de avaliações dadas no arquivo de teste, Ev_i a avaliação de um indivíduo para cada conjunto de variáveis, y_i o valor real dado no arquivo para aquele conjunto.

Como a fitness é uma medida de erro, quanto menor o erro, melhor é o indivíduo.

Tamanho da população e número de gerações

O tamanho da população para um algoritmo de programação genética é tipicamente menor do que para um algoritmo genético, onde o objetivo é achar uma solução para um problema em vez de desenvolver um método para ele.

É razoável que a população varie, neste caso, entre 50 e 500. Por motivos de limitações em questão de tempo e recursos computacionais, o tamanho da população foi fixado em 50, assim como o número de gerações. Sabendo disso, é provável que seja necessária uma taxa de mutações e cruzamento maior para obter-se mais diversidade na população reduzida.

Seleção natural

A seleção é realizada por torneiro. Dado um tamanho de torneio T, uma amostra de tamanho T é selecionada também de forma aleatória entre os indivíduos e o vencedor é o que tiver a melhor fitness, no caso, o menor erro quadrático.

A variável T é o principal fator que define a intensidade da pressão seletiva na população. Quanto maior esse número é, mais difícil é a permanência de indivíduos com fitness pior do que a média na população. Por isso, o valor adequado para esse parâmetro também foi definido através de experimentos, entre 2 e 5.

Cruzamento

O cruzamento também faz parte da seleção natural. Ele ocorre para definir a nova geração a ser avaliada no GP. Dados dois indivíduo que venceram seus respectivos torneios, a probabilidade de ocorrer um cruzamento entre eles subsequentemente é de p_c (entre 60% e 90%).

Uma das principais questões do cruzamento é a possibilidade de se manter a localidade ao cruzar duas árvores (se uma grande parte das árvores for cruzada, a variabilidade gerada pode levar a um indivíduo completamente diferente dos dois pais, o que aumenta drasticamente a aleatoriedade da busca).

Para evitar que um crossover provoque uma variedade maior do que deveria e destrua bons indivíduos, o modelo de crossover adotado é um método mais simples, porém ainda amplamente usado [1], que é a troca de duas folhas aleatoriamente escolhidas entre os pais.

Mutação

Como o cruzamento é escolhido de forma bem conservadora, é papel da mutação provocar uma variabiidade mais drástica e promover a diversidade entre a população. A estratégia usada consiste em escolher um nó aleatório da árvore, podar os seus filhos e re-extender a árvore aleatoriamente a partir desse ponto [1].

Para cada indivíduo que surge de um cruzamento, a chance de ele sofrer uma mutação é p_m (entre 5% e 30%).

Elitismo

Passar um pequeno grupo dos melhores indivíduos da população atual para a próxima geração sem submetê-los ao processo de seleção é um processo muito utilizado para fazer algoritmos convergir mais rápido e preservar bons genótipos. O elitismo é adotado neste projeto também.

Como o número de indivíduos da população é relativamente pequeno, julgou-se razoável manter o tamanho do grupo elitista em apenas um, para que a pressão seletiva não fosse desproporcional e assim, evitando que todos os indivíduos ficassem iguais antes do número de iterações desejado.

3 Experimentos

Os experimentos foram realizados em um desktop com sistema operacional Debian 9, Python 3.5.3, processador Pentium Dual Core (3.3 GHz) e 8GB de memória RAM. Foram usadas três bases de dados diferentes (house, keijzer-10 e keijzer-7, cada um com um arquivo de treino e teste, todos disponíveis na pasta *datasets/*) Cada experimento foi repetido 30 vezes.

Os atributos do algoritmo foram testados na seguinte ordem:

1. Variação de p_m de 0.05 a 0.3
2. Variação de p_c de 0.9 a 0.6
3. Variação de T de 2 a 5

Os gráficos plotados a seguir incluem a média aritmética das melhores soluções obtidas para cada configuração, no *arquivo de treino*, a cada geração. As tabelas em *test-logs/best/* apresentam a média dos melhores valores obtidos para a função, para cada um dos experimentos, levando em consideração o *arquivo de teste*.

Mais detalhes sobre os experimentos podem ser encontrados nos logs na pasta *test-logs/average/*. Lá estão tabelados, para cada variação, na média dos 30 experimentos, a fitness do melhor indivíduo, média das fitness de todos os indivíduos e a fitness do pior, respectivamente. Como a ordem de grandeza desses valores tem uma grande variação, principalmente no caso de teste *house*, que possui erros maiores, não foi possível plotar esses dados de forma satisfatória.

3.1 Mutação

As figuras de 1 a 3 demonstram a média do erro das melhores soluções para os arquivos de teste nos experimentos realizados. No momento, os testes estão sendo realizados com $p_m = 0.05$ e $p_c = 0.9$. Os plots azul, vermelho, marrom e preto correspondem, respectivamente, às taxas de mutação de 0.05, 0.1, 0.15 e 0.2.

Os dados plotados dão uma boa ideia de como a solução converge. Podemos reparar que, como era esperado, valores maiores de mutação são os que promovem a diversidade necessária para que a convergência a longo prazo seja mais favorável. Este parece ser o caso para todos os conjuntos de dados.

Usar um valor de mutação mais baixo, por sua vez, promove menos diversidade, porém facilita a convergência. Pode-se perceber que o menor valor de mutação (linha azul) no geral converge relativamente rapidamente, porém nos dois últimos testes (que são bases de dados menores e com menos variáveis), ele fica estagnado. Isso pode ser um sinal de que os indivíduos da população ficam iguais em certo ponto, e a chance de se conseguir sair dessa situação é baixa.

Como há uma tendência a achar soluções mais bem adaptadas quanto maior é o valor da mutação (e essa tendência é confirmada também pelas tabelas com as melhores fitness de cada taxa de mutação), o valor da mutação

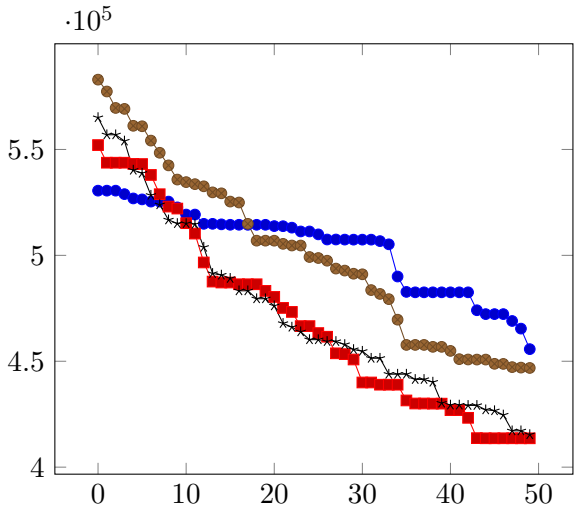

Figure 1: Testes de mutação: melhores soluções, house-train.csv

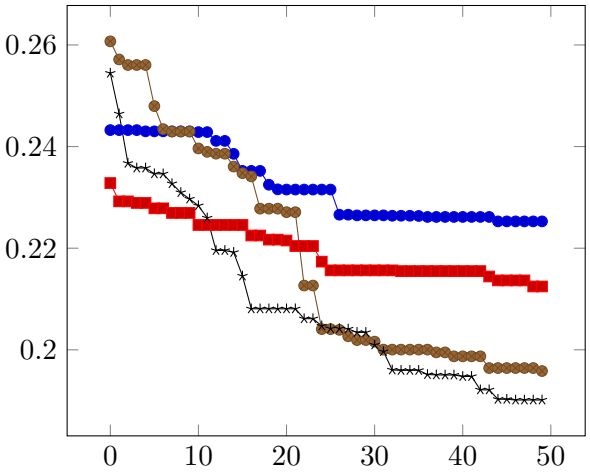

Figure 2: Testes de mutação: melhores soluções, keijzer-10-train.csv

para os próximos experimentos foi fixado no maior possível dentro do limite estipulado, que é 0.3.

3.2 Cruzamento

Nos gráficos para os testes de cruzamento, as cores azul, vermelho, marrom e preto corresponder respectivamente às chances de cruzamento de 0.9, 0.8,

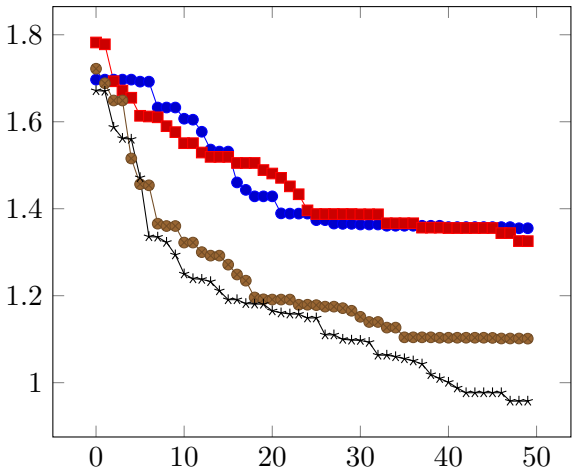

Figure 3: Testes de mutação: melhores soluções, keijzer-7-train.csv

0.7 e 0.6. Sabemos que o cruzamento visa conseguir alguma variabilidade genética na população, porém não de forma radical como a mutação. Como a mutação está bem alta, é de se esperar que achemos um limite superior para a probabilidade de crossover durante essa série de experimentos. Os resultados estão à mostra nas figuras 4, 5 e 6.

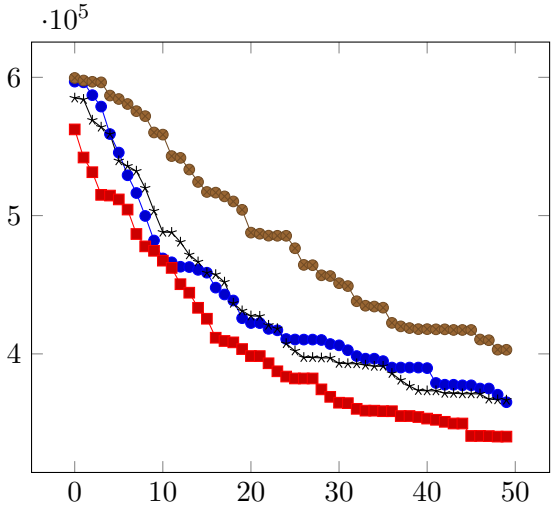

Figure 4: Testes de cruzamento: melhores soluções, house-train.csv

Nos deparamos com um resutado curioso: diminuir o crossover para 0.8

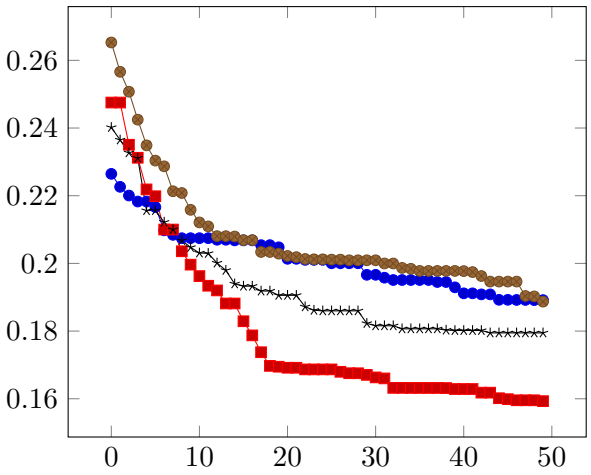

Figure 5: Testes de cruzamento: melhores soluções, keijzer-10-train.csv

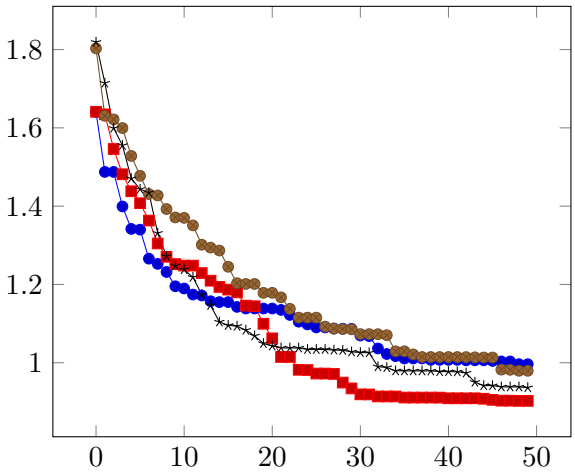

Figure 6: Testes de cruzamento: melhores soluções, keijzer-7-train.csv

melhora as qualidades das soluções consistentemente. Um aumento para 0.7 leva a resultados piores que os dois anteriores no geral, e espera-se que isso seja porque falta variabilidade na população, porém houve uma melhora a partir da redução para 0.6. Imagina-se que a menor taxa de crossover levou o algoritmo a convergir melhor em direção aos bons indivíduos, o que funciona bem a curto prazo.

Mesmo levando em conta esses resultados, a configuração que gerou os

melhores resultados no geral foi $p_c = 0.8$, e portanto, ela foi mantida para os experimentos seguintes.

3.3 Torneio

Como explicitado antes, o torneio é a principal medida que regula o quão intensa é a pressão seletiva do GP.A mesma sequência de cores foi usada novamente para os gráficos, atribuindo azul a $T = 2$, vermelho a 3, marrom a 4 e preto a 5.

Na análise desses dados, expressos nos gráficos das figuras 7 a 9, não se deve se ater apenas às qualidades das soluções encontradas, mas também aos aspectos dos gráficos. Percebemos na figura 8 que o menor valor para o torneio tem a tendência de fazer com que as melhores soluções tenham uma convergência estável: a cada passo, a média diminui uma quantidade razoável. Porém, o maior valor para o torneio faz com que a solução tenda a convergir em "picos": o erro diminui muito em um passo específico, e permanece quase constante durante outros.

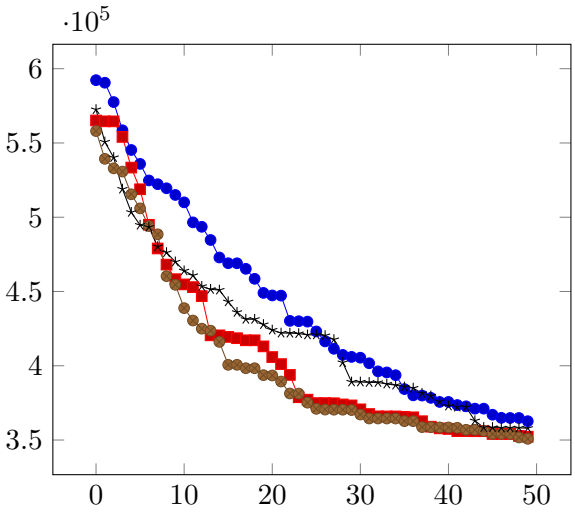

Figure 7: Testes de torneio: melhores soluções, house-train.csv

Porém, analisando a qualidade das soluções finais para cada um dos gráficos, percebe-se que as maiores pressões seletivas prevalecem. Porém, o primeiro lugar parece compartilhado entre os valores 5 e 4, na keijzer-10 e keijzer-7, respectivamente.

Ao analisar as funções em si, vemos que os erros para a primeira são uma ordem de grandeza maior do que a segunda, apesar de seus valores serem

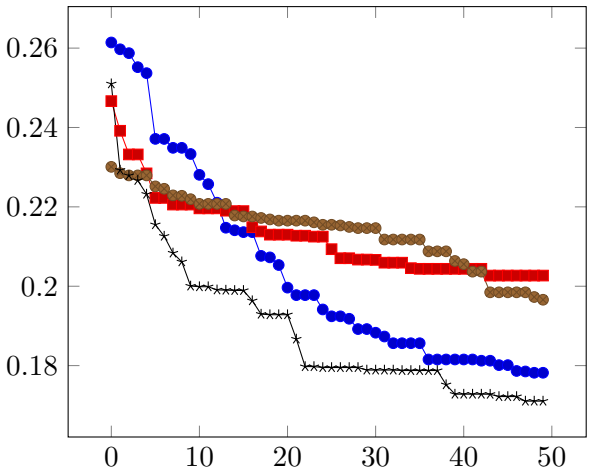

Figure 8: Testes de torneio: melhores soluções, keijzer-10-train.csv

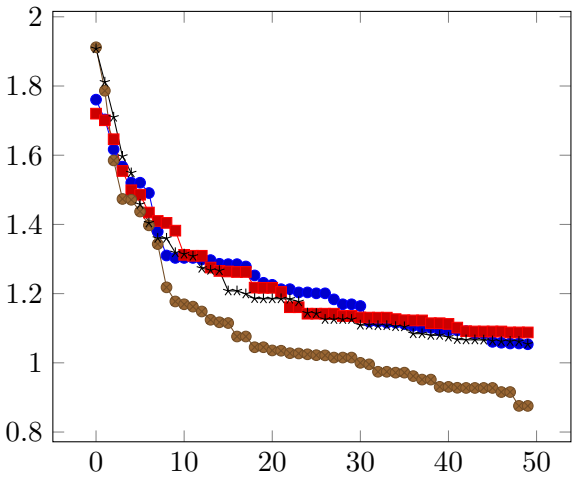

Figure 9: Testes de torneio: melhores soluções, keijzer-7-train.csv

semelhantes. Isso sugere que a primeira função é mais fácil de se aproximar do que a segunda, portanto, uma convergência com muita pressão pode capitalizar muito em manter genes enquanto mais diversidade e abrangência são necessárias para ter melhores possibilidades de lidar com a função. A própria natureza desse segundo caso de teste pode ter contribuído para isso, como é discutido no tópico seguinte.

Portanto, o valor de T pode depender não só das outras características

do algoritmo, mas também da função que se quer aprender.

3.4 Exemplos de melhores soluções e discussão

Vários exemplos de funções aproximativas foram obtidos ao se rodar o algoritmo com os valores $p_m = 0.3$, $p_c = 0.8$, $T = 4$, para cada um dos testes. Os arquivos estão na pasta *test-logs/functions/*.

Podemos fazer algumas observações a respeito, começando pelo dataset *keijzer-10*, que possui apenas duas variáveis e resultados apresentados no arquivo de treino e teste são relativamente uniformes. Isso sugere que muitas funções que aparecem como melhores são funções menores e que impliquem uma relação direta entre as duas variáveis. Há uma grande incidência de respostas do tipo (a/b) ou (b/a) ou mesmo (a/a) entre as melhores (nesse caso, como o resultado é um, e os resultados apresentados no treino estão entre 0.5 e 1, o erro quadrático não será muito elevado. Os erros obtidos nessa função foram, muitas vezes, menores do que 0.1.

Porém, o caso *keijzer-7* não foi fácil de aproximar, sendo que os melhores erros atingidos muitas vezes não chegaram a estar abaixo de 1, mesmo que haja apenas uma variável. Os melhores indivíduos têm funções extremamente extensas e não tem quase nenhuma uniformidade entre si, tanto em genótipo quanto em fenótipo. Ao analisar o arquivo, vemos que ele sugere uma distribuição logarítmica. Isso não é tratado pelos operadores no programa, e complica o aprendizado. Para eventuais casos como esse, ter operadores de logaritmos, funções trigonométricas ou até mesmo potenciação possa ser vantajoso.

Sobre o teste *house*, como foi sugerido na especificação do trabalho prático, trata-se de um problema real e consiste em aprender uma função para deduzir o preço de uma casa a partir de uma série de fatores, cuja natureza não é conhecida. No geral, é notável que algumas variáveis que possuem valor numérico menor, como 'a', 'e' e 'f' aparecem sendo multiplicadas por constantes, o que sugere que elas tem uma grande chance de ser relevantes por intermédio de algum modificador.

Como a ordem de grandeza dos resultados desse último caso é muito alta, e de seus operadores também, espera-se que os erros no geral sejam maiores, porém a sua convergência, como demonstrado nos gráficos acima, ocorra em um ritmo semelhante aos outros dois. Para esse teste, cerca de 250000 de erro é satisfatório, tendo em vista o escopo desse trabalho.

4 Conclusões

A série de experimentos realizada nesse trabalho pode levar a uma série de conclusões. Em primeiro lugar, ao analisar a divergência entre as configurações ótimas do GP entre uma função e outra, entende-se que pode ser possível inferir algo como uma lei de *no free lunch* para algoritmos genéticos, como já existe uma para algoritmos de otimização [2].

Por exemplo, a variância do valor de torneio ótimo entre os teste demonstra que não existe um ótimo global dominante deste valor para um algoritmo.

Para melhorar o algoritmo apresentado, uma das opções mais razoáveis seria adicionar mais operadores para obter um espaço de funções mais versátil. Isso ajudaria em casos como o *keijzer-7*, e muito provavelmente em problemas da vida real, sendo que vários comportamentos na natureza possuem comportamento que depende de funções exponenciais, logarítmicas, ou até que incluem números complexos, como interferências entre ondas [3].

Outra possibilidade para melhorar a implementação em questão de eficiência seria representar o indivíduo como gramática em vez de usar a implementação tradicional como árvore. Tal decisão favorece a localidade de referência, e é um método popular para GPs [1, cap. 6].

5 Referências

1. *A Field Guide To Genetic Programming.* - Riccardo Poli, William B. Langdon, and Nicholas F. McPhee, 2008

2. WOLPERT, David H., and William G. MACREADY, 1997. *No free lunch theorems for optimization.* IEEE Transactions on Evolutionary Computation, 1(1), 67–82.

3. `http://physics.tutorvista.com/waves/wave-interference.html`