Programowanie ekspresji genów

Modelowanie komputerowe

Matejko Marek, Mazur Krzysztof, Paszkot Dawid

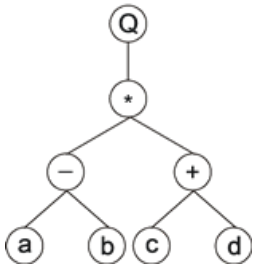
Fizyka Techniczna
Politechnika Krakowska

Plan Prezentacji

Spis treści

Wprowadzenie

Programowanie ekspresji genów (*ang. Gene expression programming*-GEP) to algorytm tworzący programy lub modele. Te programy są złożonymi strukturami drzewiastymi, które uczą się i przystosowują, zmieniając swoje rozmiary, kształty i skład, podobnie jak żywy organizm



Przykład struktury drzewiastej

Wprowadzenie

Gen to symboliczny ciąg mający głowę i ogon. Każdy symbol reprezentuje operację. Operacja $+$ pobiera dwa argumenty i dodaje je. Operacja $*$ również bierze dwa argumenty i mnoży je. Operacja **x** oblicza wartość zmiennej x.
Ogon składa się tylko z operacji, które nie mają argumentów i zapewnia kompletność wyrażeń.
Wyrażenia zapisywane są w notacji prefixowej, czyli **5-3** zapisane jest jako **- 5 3**

Wprowadzenie

Załóżmy, że łańcuch ma w nagłówku h symboli, które są określone jako dane wejściowe do algorytmu, oraz t symboli w ogonie, który jest określany na podstawie h. Więc jeśli n jest maksymalną liczbą argumentów dla operacji musimy mieć

$$h + t - 1 = hn$$

Lewa strona to całkowita liczba symboli z wyjątkiem pierwszego. Prawa strona to całkowita liczba argumentów wymaganych dla wszystkich operacji. Równanie mówi, że musi być wystarczająco dużo symboli, aby służyły jako argumenty dla wszystkich operacji
Z tego możemy wyznaczyć długość ogona $t = h(n-1)+1$

Przykład

Załóżmy że $h = 8$ i $n = 2$. W tedy długość ogona musi być $t = 9$, czyli długość genu wynosi 17. Możemy więc przedstawić wyrażenie

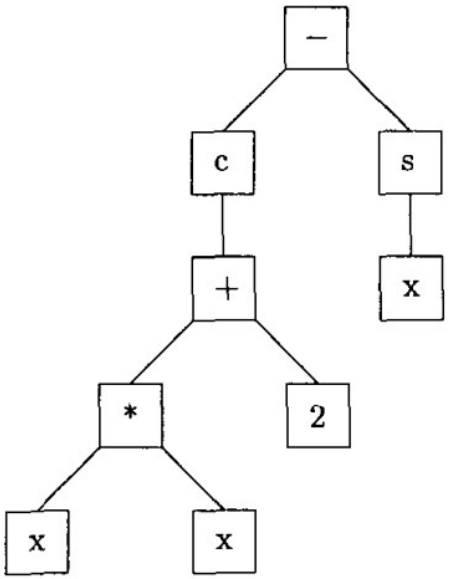
$$\cos(x^2 + 2) - \sin x$$

używając łańcucha -c+ $*xx2s \,|\, x1x226x31$

Znak | jest używany do oznaczenia początku ogona.
Symbol c oznacza cosinus, a s sinus

Wprowadzenie

Możemy przedstawić wyrażenia w postaci drzewa. Dla poprzegniego przykładu korzeniem byłby ”-” z odgałęzieniami dla parametrów. W ten sposób moglibyśmy przedstawić wyrażenie w następujący sposób



Nie wszystkie symbole z ogona są używane. W tym przykładzie użyto tylko jednego symbolu z ogona

Wprowadzenie

Chromosom to zbiór genów. Geny łączą się i tworzą ekspresję za pomocą operacji. Na przykład ekspresje genów chromosomu mogą być sumowane. Często łączymy geny, aby otrzymać pojedynczy ciąg symboli. Załóżmy na przykład, że mamy następujące geny tworzące chromosom:

$$-c+*xx2s\,|\,x1x226x31$$

$$-c+*xx2s\,|\,x1x226x31$$

$$-c++x2*x\,|\,x1x226x31$$

Przez złączenie otrzymujemy chromosom

$$-c+*xx2s|\,x1x226x31|\,-c+*xx2s|\,x1x226x31|\,-c++x2*x|\,x1x226x31$$

gdzie | wskazuje początek części głowy i ogona genów

Wprowadzenie

Na chromosomach stosuje się szereg operacji

- **Replikacja**. Chromosom jest niezmieniony. Wybór proporcjonalny do sprawności (*ang. Fitness proportionate selection*) może być użyty do selekcji chromosomów do replikacji
- **Mutacja**. Symbole w chromosomie zmieniają się losowo. Symbole w ogonie genu nie mogą opierać się na żadnych argumentach np. dla $+*$ **xy2**$|$ **xyy32y** mutacja może wyglądać: $+*yy2|xyy32y$, $++xy2|xyy32y$ lub $+*xy-|xyy32y$

Wprowadzenie

- **Wprowadzanie**. Część chromosomu jest wybrana do wstawienia w głowę genu. Ogon genu jest nienaruszony
- **Transpozycja genów**. Jeden gen w chromosomie jest losowo wybierany jako pierwszy gen. Wszystkie inne geny są przesunięte w dół, aby zrobić miejsce dla pierwszego genu
- **Rekombinacja**. Może to być jeden punkt (chromosomy są dzielone na dwa i odpowiadające im sekcje są zamieniane), dwa punkty (chromosomy są dzielone na trzy, a środkowa część jest zamieniana) lub rekombinacja genów (zamieniany jest cały jeden gen). Zazwyczaj suma prawdopodobieństw rekombinacji wynosi 0,7

W następującym przykładzie realizujemy te operacje.
Dla uproszczenia używamy tylko jednego genu i rekombinacji jedno punktowej

Spis treści

Przykład

W naszym przykładzie mamy jednowymiarową mape $f : R \to R$ mającą właściwości:

$$f(0) = 0, f(1) = 0, f(\tfrac{1}{2}) = 1$$
$$\tfrac{1}{2} < f(\tfrac{1}{4}), \tfrac{1}{2} < f(\tfrac{3}{4} < 1)$$

Na przykład mapa logistyczna $g(x) := 3x(1 - x)$ pełnia te właściwości.
Używamy Gene Expression Programming do wykonywania regresji symbolicznej w celu uzyskania map spełniających powyższe właściwości. Spodziewamy się, że mapa logistyczna zostanie znaleziona, ponieważ ograniczamy funkcje w implementacji do wielomianów.

Przykład

Uogólniamy zestaw właściwości w następujący sposób. Punkty oceny są określone przez podzbiór $X \times Y$ gdzie X i Y są podane przez $f : X \to Y$ oznaczone przez $F_= \subset X \times Y$ podzbiór wszystkich $(x, y) \in X \times Y$ takich że wymaga od funkcji f, że $f(x) > y$ przez $F_< \subset X \times Y$ podbiór wszystkich $(x, y) \in X \times Y$ takich że wymaga od funkcji f, że $f(x) < y$

Przykład

W ten sposób możemy zdefiniować funkcję dopasowania funkcji g (gdzie mniejsza wartość wskazuje na wyższą dopasowanie):

$$fitness(g) : \sum_{(x,y)\in F_=} |g(x) - y| + \sum_{(x,y)\in F_>} |g(x) - y| H(y - g(x)) + \sum_{(x,y)\in F_<} |g(x) - y| H(g(x) - y)$$

gdzie:

$$H(x) := \begin{cases} 1x > 0 \\ 0 \end{cases}$$

Stosujemy programowanie ekspresji genów, dopóki nie znajdziemy funkcji g, która:

$$fitness(x) < \varepsilon$$

dla $\varepsilon > 0$. Typowa wartość $\varepsilon = 0.001$.

Spis treści

Implementacja - eval() i evalr()


```
double evalr(char*& e, double x)
{
    switch (*(e++))
    {
    case '1': return 1.0;
    case 'x': return x;
    case 'y': return pi * x;
    case 'c': return cos(evalr(e, x));
    case 's': return sin(evalr(e, x));
    case '+': return evalr(e, x) + evalr(e, x);
    case '-': return evalr(e, x) - evalr(e, x);
    case '*': return evalr(e, x) * evalr(e, x);
    default: return 0, 0;

    }

}
```

Rysunek: Funckja evalr()

```
double eval(char* e, double x)
{
    char* c = e;
    return evalr(c, x);
}
```

Rysunek: Funckja eval()

Implementacja - printr() i print()

```
void printr(char*& e)
{
    switch (*(e++))
    {
    case '1': cout << '1'; break;
    case 'x': cout << 'x'; break;
    case 'y': cout << "pi*x"; break;
    case 'c': cout << "cos("; printr(e); cout << ")"; break;
    case 's': cout << "sin("; printr(e); cout << ")"; break;
    case '+': cout << '('; printr(e); cout << '+'; printr(e);
        cout << ')'; break;
    case '-': cout << '('; printr(e); cout << '-'; printr(e);
        cout << ')'; break;
    case '*': cout << '('; printr(e); cout << '*'; printr(e);
        cout << ')'; break;

    }
}
```

Rysunek: Funckja printr()

```
void print(char* e)
{
    char* c = e;
    printr(c);
}
```

Rysunek: Funckja print()

Implementacja - fitness() i gep()

```
double fitness(char* c, double* data, int N)
{
    double sum = 0.0;
    double d;
    for (int j = 0; j < N; j++)
    {
        d = eval(c, data[3 * j]) - data[3 * j + 1];
        if (data[3 * j + 2] == 0) sum += fabs(d);
        else if (data[3 * j + 2] > 0) sum -= (d > 0.0) ? 0.0 : d;
        else if (data[3 * j + 2] < 0) sum -= (d < 0.0) ? 0.0 : d;
    }
    return sum;
}
```

Rysunek: Funckja fitness()

```
void gep(double* data, int N, int P, double eps)

{
    int i, j, k, replace, replace2, rlen, rp;
    int t = h * (n - 1) + 1;
    int gene_len = h + t;
    int pop_len = P * gene_len;
    int iterations = 0;
    char* population = new char[pop_len];
    char* elim = new char[P];
    int toelim = P / 2;
    double bestf, f;
    double sumf = 0.0;
    double pmutate = 0.1;
    double pinsert = 0.4;
    double precomb = 0.7;
    double r, lastf;
    char* best = (char*)NULL;
    char* iter;
```

Rysunek: Funckja gep()

Spis treści

Wyniki

Stwierdzamy, że w większości przypadków najlepiej dopasowaną mapą jest -4(x - 1)x i x(1 - x)(2x + 3)

- ((1 - x) ((1 + (1 + 1)) + 1)) x = -4(x - 1)x
- ((x*(1-x))*(((1+1)+1)+1)) = -4(x - 1)x
- ((x+(((((((1-1)-1)-x)-x)*x)+1)*x))+x) = x(1 - x)(2x + 3)
- (((1-x)-(x-1))*(x+x)) = -4(x - 1)x
- ((((1-(1*(((1+x)+x)*x)))*x)+x)+x) = x(1 - x)(2x + 3)
- (((x-((x+x)*((x+x)-1)))+x)*1) = -4(x - 1)x
- ((1-x)*(x+(((x+x)*1)+x))) = -4(x - 1)x

Wyniki

Oczywiście znajdujemy pożądane funkcje z mniejszą liczbą iteracji, jeśli wielkość populacji jest zwiekszona taka funkcją jest x(1 - x)(2x + 3) ale g(x) ma wartości większe niż 1 na przedziale.

$$\left(\frac{1}{2}, \frac{\sqrt{5}}{2} - \frac{1}{2}\right)$$

Stwierdzamy zatem, że prawie wszystkie początkowe wartości z x € [0, 1] wymykają się przedziałowi [0, 1] poniżej iteracji mapy. Zbiór wszystkich punktów, których iteracyjny wartość jest w [0, 1] ma miarę zero tworzą zbiór Cantora.

Spis treści

Bibliografia

[1] https://pl.wikipedia.org/wiki/C%2B%2B

[2] https://pl.wikipedia.org/wiki/Zbiór_Cantora

[3] GeneExpression.pdf

[4] https://en.wikipedia.org/wiki/Gene_ expression