Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

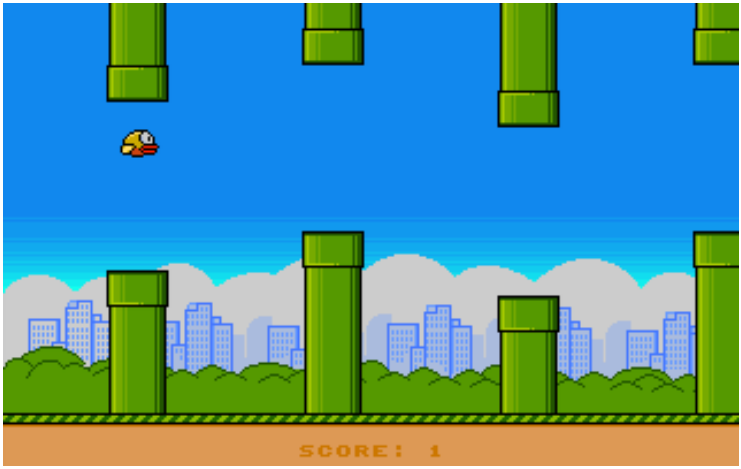
AGH University of Krakow

Implementacja algorytmu NEAT na CUDA

Michał Michniak, Zofia Jankowska, Julia Zoń

Wydział EAIiIB

Gra Flappy Bird

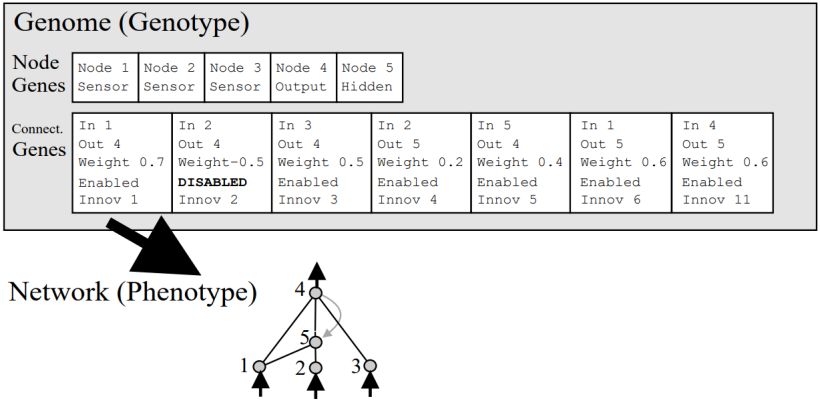


Rysunek: Gra zręcznościowa Flappy Bird

Algorytmy genetyczne

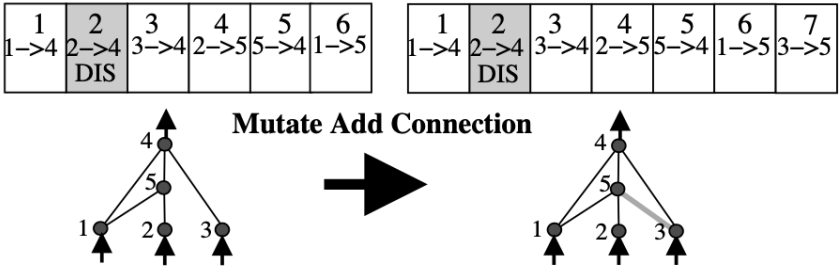
- heurystyczne metody optymalizacji zainspirowane naturalną selekcją i teorią ewolucji
- należą do klasy algorytmów ewolucyjnych
- rozważają całą populację rozwiązań umożliwiając równoległą ewaluację rozwiązań
- elementy prostego algorytmu genetycznego:
  - mechanizm kodowania rozwiązania
  - funkcja celu
  - mechanizm selekcji
  - operatory genetyczne (krzyżowania i mutacji)

NEAT



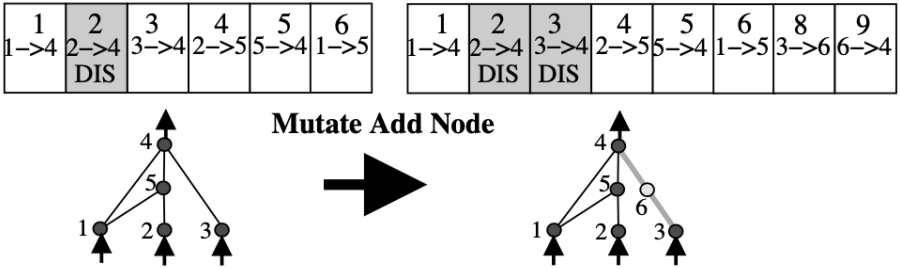
Rysunek: Instancja populacji

NEAT Mutacje



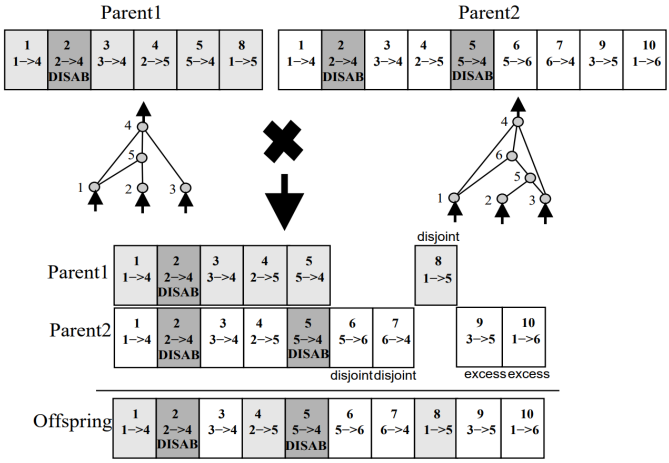
Rysunek: Mutacja: dodanie krawędzi

NEAT Mutacje



Rysunek: Mutacja: dodanie wierzchołka

NEAT Krzyżowanie



Rysunek: Krzyżowanie

NEAT Krzyżowanie

Jeżeli przyjmiemy, że wierzchołki rodziców to zbiory:

$$V_1 = \{v_1, v_2, \cdots\} \quad V_2 = \{v_1', v_2', \cdots\}$$

zbiór wierzchołków potomka przyjmie postać:

$$V_{1,2} = V_1 \cup V_2$$

Podobnie dla krawędzi, tylko względem numeru innowacji. Wartości wag krawędzi potomka są średnią arytmetyczną wag krawędzi rodziców.

$$E_1 = \{(e_1, w_1), (e_2, w_1), \cdots\} \quad E_2 = \{(e_1', w_1), (e_2', w_1), \cdots\}$$

$$E_{1,2} = \left\{(e_i, w_i) : \exists_{(e_i, w_i') \in E_1} \exists_{(e_i, w_i'') \in E_2} w_i = \frac{w_i' + w_i''}{2}\right\}$$
$$\cup \left\{(e_i, w_i) : \exists_{(e_i, w_i') \in E_1} \neg\exists_{(e_i, w_i'') \in E_2} w_i = w_i'\right\}$$
$$\cup \left\{(e_i, w_i) : \neg\exists_{(e_i, w_i') \in E_1} \exists_{(e_i, w_i'') \in E_2} w_i = w_i''\right\}$$

Zapis populacji

Zapis populacji odbywa się w 8-śmiu wektorach (konkatenacja instancji populacji):

- $blocks\_nodes \in \mathbb{N}^{N+1}$ - histogram skumulowany ilości wierzchołków (node) instancji (zaczynający się od 0)
- $blocks\_edges \in \mathbb{N}^{N+1}$ - histogram skumulowany ilości krawędzi (edges) instancji
- $translation \in \mathbb{N}^{blocks\_nodes[N]}$ - mapowanie odcinków liczb naturalnych [0,n] do rzeczywistych numerów wierzchołków (założenie, że w instancjach posortowane rosnąco)
- $in \in \mathbb{N}^{blocks\_edges[N]}$ - wejścia synaps (krawędzi) instancji zmapowane do odcinka [0,n]

Zapis populacji

- $out \in \mathbb{N}^{blocks\_edges[N]}$ - wyjścia synaps (krawędzi) instancji zmapowane do odcinka [0,n]
- $w \in \mathbb{R}^{blocks\_edges[N]}$ - wagi synaps (krawędzi) instancji
- $enabled \in \{0,1\}^{blocks\_edges[N]}$ - flagi opisujące aktywność krawędzi (true jeżeli biorą udział w ewaluacji, w przeciwnym wypadku - false)
- $innov \in \mathbb{N}^{blocks\_edges[N]}$ - numer innowacji, unikatowy numer rozróżniający krawędzie pomiędzy genami

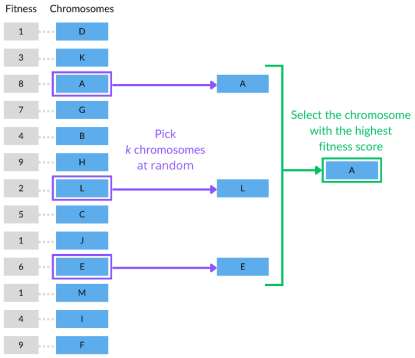
Symulacja

Zmiana zapisu populacji na macierze CSR [wzorzec skanu + zrównoleglenie na poziomie instancji] o postaci (w czasie jednego kroku symulacji jedna iteracja sieci Hopfielda) [zrównoleglane na poziomie wierszy wyjściowych + wzorzec redukcji do kryterium STOP-u 2]:

$$\begin{bmatrix} \begin{bmatrix} w_{11}^1 & w_{12}^1 & \dots \\ w_{21}^1 & \ddots & \\ \vdots & & w_{mm}^1 \end{bmatrix} & & \\ & \ddots & \\ & & \begin{bmatrix} w_{11}^N & w_{12}^N & \dots \\ w_{21}^N & \ddots & \\ \vdots & & w_{kk}^N \end{bmatrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} out(t)_1 \\ in(t)_1 \\ \vdots \end{bmatrix} \\ \vdots \\ \begin{bmatrix} out(t)_N \\ in(t)_N \\ \vdots \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} out(t+1)_1 \\ in(t+1)_1 \\ \vdots \end{bmatrix} \\ \vdots \\ \begin{bmatrix} out(t+1)_N \\ in(t+1)_N \\ \vdots \end{bmatrix} \end{bmatrix}$$

Generacja nowej populacji

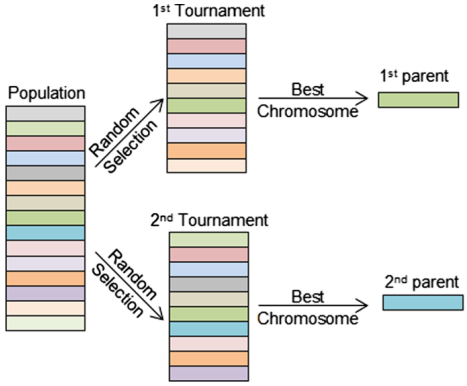
Maska binarna, która instancja przetrwa jest generowana poprzez tzw. selekcje turniejową [zrównoleglenie na poziomie turniejów]:



Następnie na CPU zliczane są instancje z maski i przyporządkowywana jest im kolejność w nowej populacji (jedyna część sekwencyjna w programie).

Generacja nowej populacji

Dobieranie rodziców także za pomocą turniejów [zrównoleglenie na poziomie turniejów]:



Obliczanie pamięci

Po wybraniu rodziców, a także instancji, które przechodzą do następnej populacji oraz instancji, które dołączymy do populacji z mutacjami, liczy się potrzebną ilość miejsca do alokacji (dynamicznie dla każdej populacji) [zrównoleglenie na poziomie instancji + wzorzec skan hierarchiczny].

Nowa populacja

Po obliczeniu pamięci i offsetów dla każdej instancji następuje wygenerowanie (lub przepisanie) ich w wybrane miejsca w nowej populacji. [zrównoleglenie na poziomie instancji]. Następnie mutacje wag w krawędziach [zrównoleglenie na poziomie krawędzi].

Na koniec działania algorytmu wybierana jest najlepsza instancja i jest ona zapisywana [wzorzec skan].

Bibliografia I

Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." Evolutionary computation 10.2 (2002): 99-127. https://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf