Automaty Komórkowe

Wykład 5 - zdalny

Witold Bołt, 03.04.2024

Poprzednio omówiliśmy

- **Wykład 1**: Sprawy organizacyjne, motywację do zajmowania się CA, podstawowe pojęcia / definicje / intuicje.
- **Wykład 2**: Definicja (formalna) i podstawowe w fakty o ECA. Reprezentacja Wolframa.
- **Wykład 3**: Symetrie w zbiorze ECA, relacje do ogólnej teorii układów dynamicznych, własności CA/ECA.
- **Wykład 4**: Alternatywne reprezentacje reguły lokalnej (wielomiany, wyrażenia logiczne), problem klasyfikacji gęstości (DCP).

- **Dwa tygodnie przerwy** 😢

Poszukiwanie CAs

Wprowadzenie do algorytmów ewolucyjnych

Algorytmy ewolucyjne

- **Nie mają** zbyt wiele wspólnego z teorią automatów komórkowych? A może jednak…
  - Inspirowane naturą
  - Użyteczne w poszukiwaniu CAs
  - **Proste**!

- Zanim jednak zastosujemy algorytm ewolucyjny do poszukiwania konkretnych automatów, musimy bliżej zapoznać się z “szablonem” takiego algorytmu w *oderwaniu* od automatów…

Algorytmy ewolucyjne - motywacja

- Niech $g: X \to \mathbb{R}_+$ będzie pewną funkcją, a zbiór X dowolnym zbiorem. Nie zakładamy nic specjalnego o funkcji g. Co więcej, być może nie znamy nawet dobrze wzoru ogólnego na g albo nawet jeśli znamy wzór, to niewiele umiemy z nim zrobić.

- Poszukujemy takiego $\mathbf{x}_0 \in X$, że $\forall_{\mathbf{x} \in X}\ g(x) \leq g(x_0)$. Ponieważ jednak zbiór X może być nieskończony i niezbadany :) to zadowoli nas to, jeśli powyższa nierówność zajdzie dla bardzo wielu x'ów.

- Zwróć uwagę, że gdyby $X \subset \mathbb{R}^n$ i gdyby g było funkcją różniczkowalną, to wystarczyłoby zastosować znane z analizy matematycznej narzędzia i wyznaczyć ekstrema funkcji. Niestety jednak nie jest tak dobrze, bo ani X nie musi być podzbiorem $\mathbb{R}^n$, ani nie musi być różniczkowalna (a nawet jeśli jest, to być może nie umiemy tej pochodnej policzyć).

- Co wtedy począć?

Algorytmy ewolucyjne - motywacja

- Jeśli nie mamy żadnego dobrego pomysłu, w szczególności jeśli nie mamy żadnej dodatkowej wiedzy o funkcji g, to możemy spróbować **losowania**.

- Po prostu *losujemy* “wiele razy” elementy $\mathbf{x} \in X$ i wybieramy taki, który spełnia naszą nierówność “najlepiej”.

- Czy to brzmi mądrze?

- Być może jeszcze nie, bo nie powiedzieliśmy sobie co to znaczy *losujemy*!

Algorytmy ewolucyjne - podstawy

- Załóżmy, że wybraliśmy zbiór $\mathscr{P}^0 = \{\mathbf{x}_1^0, \ldots, \mathbf{x}_K^0\} \subset X$ potencjalnych rozwiązań naszego problemu. Taki zbiór nazwiemy populacją. Nasza populacja składa się z $K > 0$ osobników.
- Pierwszą populację (inicjalną) $\mathscr{P}^0$ wybierzemy zupełnie losowo.
- Algorytm ewolucyjny poda nam przepis na to jak z danej populacji $\mathscr{P}^{j-1}$ wygenerować nową populację $\mathscr{P}^j$, która przy odrobinie szczęścia będzie zawierać “lepsze” potencjalne rozwiązania.
- Co to znaczy lepsze (a właściwie nie gorsze)? To znaczy, że:

$$\max_{\mathbf{x}\in\mathscr{P}^{j-1}} g(\mathbf{x}) \leq \max_{\mathbf{x}\in\mathscr{P}^{j}} g(\mathbf{x})$$

- A w ogóle w idealnym przypadku: $\lim_{j\to\infty} \max_{\mathbf{x}\in\mathscr{P}^j} g(\mathbf{x}) = \max_{\mathbf{x}\in X} g(\mathbf{x})$

Algorytmy ewolucyjne - podstawy

- Algorytm ewolucyjny opierać się będzie na 3 operacjach:
  - **Selekcja** - jak wybierać elementy z poprzedniej populacji?
  - **Krzyżowanie** - jak “łączyć” parę elementy ze sobą, aby powstał nowy, lepszy kandydat
  - **Mutacja** - jak (lekko) modyfikować pojedyncze elementy
- Możliwych wyborów tych operatorów jest bardzo, bardzo wiele i właściwie nie ma idealnej, uniwersalnej recepty jak podejść do wyboru tych operacji. W literaturze znajdziemy sporo sprawdzonych pomysłów, które lepiej lub gorzej sprawdzają się w danym problemie.

Algorytmy ewolucyjne - selekcja

- Zakładamy, że gotowa jest populacja $\mathcal{P}^{j-1}$
- Jak wybierać elementy, które będą następnie podlegać kolejnym operacjom?
- Losowanie!
  - **Pomysł 1** (fitness proportional). Losujemy element z prawdopodobieństwem proporcjonalnym do wartości funkcji g - czym wyższa wartość, to większa szansa na bycie wybranym.
  - **Pomysł 2** (tournament). Losujemy M elementów z $\mathcal{P}^{j-1}$ (losowo - każdy ma taką samą szansę na wybór) a następnie spośród nich wybieramy taki, które ma największą wartość funkcji g.

Algorytmy ewolucyjne - krzyżowanie

- Zakładamy, że wybraliśmy dwa (najlepiej różne) elementy z $\mathscr{P}^{j-1}$ zgodnie z operacją selekcji.
- Chcemy stworzyć ich “potomka”. Jak to zrobić?
- Musimy przyjąć pewne założenia o zbiorze X, a mianowicie, że elementy tego zbioru da się przedstawić jako ciągi, czyli jeśli $\mathbf{x} \in X$, to $\mathbf{x} = (x_0, x_1, \ldots)$, a najczęściej $\mathbf{x} = (x_0, x_1, \ldots, x_{q-1})$ dla pewnego $q > 0$.
- Zwróć uwagę, że wcześniej pisaliśmy $\mathbf{x}_i^{j-1}$ na oznaczenie pewnego elementu zbioru X, który był i-tym elementem populacji $\mathscr{P}^{j-1}$. Teraz natomiast przez x_i rozumiemy $i+1$-wszą współrzędną wektora odpowiadającego elementowi $\mathbf{x} \in X$. Nie wiemy do jakiego zbioru należy x_i, ale wiemy, że **nie** do zbioru X!
- <u>Trzeba bardzo uważać na oznaczenia!</u>

Algorytmy ewolucyjne - krzyżowanie

- Zakładamy, że wybraliśmy dwa (najlepiej różne) elementy $\mathbf{x}, \mathbf{y} \in \mathscr{P}^{j-1}$ zgodnie z operacją selekcji.
- Chcemy stworzyć ich "potomka" $\mathbf{z} \in X$. Jak to zrobić?
  - Krzyżowanie jedno-punktowe: $z = (x_0, \ldots, x_l, y_{l+1}, \ldots, y_q)$, czyli bierzemy "kawałek wektora $\mathbf{x}$ i sklejamy go z kawałkiem wektora $\mathbf{y}$. Miejsce sklejenia l jest wybierane losowo.
  - Krzyżowanie dwu-punktowe: $z = (x_0, \ldots, x_l, y_{l+1}, \ldots, y_{l+p}, x_{l+p+1}, \ldots, x_q)$ gdzie l, p są wylosowane.
  - Krzyżowanie jednorodne: $z = (z_0, \ldots, z_q)$, gdzie $z_i \in \{x_i, y_i\}$ wybierane jest losowo, niezależnie dla każdej współrzędnej.

Algorytmy ewolucyjne - mutacja

- Mamy już naszego “potomka” $\mathbf{z} \in X$. Chcemy go dodać do $\mathscr{P}^j$, ale zanim to zrobimy, chcemy lekko “zamieszać”.
- Jak to zrobić?
- Chcemy skonstruować takie $\mathbf{z}' \in X$, które w jakimś sensie jest blisko $\mathbf{z}$.
- Zależnie od struktury zbioru X być może możemy zaburzyć wybraną współrzędną wektora $\mathbf{z}$. A być może to zaburzenie możemy dodać do większej liczby współrzędnych. Robimy to z bardzo małym prawdopodobieństwem, tak aby oczekiwana liczba zaburzeń nie była duża.

Algorytmy ewolucyjne - czy to wszystko?

- I tak i nie. Popularna modyfikacja - **elite survival**. Najlepsze osobniki z poprzedniej populacji z automatu przechodzą do kolejnej.
- Co może pójść nie tak?
  - **Zbieżność do maksimum lokalnego?** Prawdopodobnie zbyt słaba mutacja.
  - **Brak zbieżności?** Prawdopodobnie zbyt silna mutacja.

Naiwny przykład

Rozwiązujemy **dowolne** równanie z jedną zmienną postaci $f(x) = 0$ dla $x \in [0,1]$.

Wracamy do CA!

- Czas na rozwiązanie głównych zagadek. Co to jest ten zbiór X?
- W naszym przypadku zbiór ten to po prostu zbiór reguł lokalnych CAs zapisanych jako wektory LUT! Innymi słowy jeśli poszukujemy 1-wymiarowych, binarnych CAs o promieniu sąsiedztwa $r > 0$, to $X = \{0,1\}^{2^{2r+1}}$.
- No ale w takim razie cóż to jest funkcja g?
- I tu jest nieco trudniej, bo funkcję g - nazywaną **funkcją dopasowania**, używamy do wyrażenia interesującego nas problemu. Czyli to my sami musimy zdefiniować jaka ma być funkcja dopasowania w naszym konkretnym zagadnieniu.

Przykład - od razu dobry :)

- Załóżmy, że chcemy znaleźć CA, który bardzo dobrze przybliża rozwiązanie DCP.

- Wtedy funkcja g jako argument przyjmuję LUT pewnego CA, a jako wynik podaje liczbą konfiguracji początkowych, z wybranego przez nas zbioru konfiguracji początkowych, dla których ten CA podaję poprawną odpowiedź w DCP.

  - Napisanie analitycznego wzoru na g jest *możliwe*, ale zamiast pomóc nam w zrozumieniu problemu, tylko go utrudni. Zamiast pisać wzór - napiszemy **program** który wylicza wartość $g(\mathbf{x})$.

- Zwróć uwagę, że wtedy ma sens maksymalizacja g, bo w ten sposób znajdziemy CA, który poprawnie rozwiązuje DCP dla jak największej liczby konfiguracji początkowych, a o to dokładnie nam chodzi!

Poszukiwanie rozwiązań DCP

- Co może pójść źle?
- Dużo “hiperparametrów”:
  - Liczebność populacji, liczba potrzebnych populacji
  - Siła mutacji
  - Wybór zbioru warunków początkowych do wyliczenia wartości funkcji dopasowania (zbiór Marcina?)
- **Długi czas obliczeń!!!**

Jak poprawić wydajność?

Kilka słów o informatyce…

Co zrobić gdy program działa wolno?

- Optymalizacja *algorytmiczna* (złożoność obliczeniowa)
- Optymalizacja *implementacji* (“sztuczki” techniczne, wykorzystanie bibliotek, lepsza organizacja kodu, dostosowanie kodu do możliwości sprzętu)

- **Cache** - nie liczmy w kółko tego samego - zapamiętujemy wcześniejsze wyniki
- **Równoległe obliczenia** - nasz komputer ma procesor wielo-rdzeniowy! A poza tym, być może mamy do dyspozycji wiele komputerów?!

Dziękuję bardzo

Witold.Bolt@ug.edu.pl