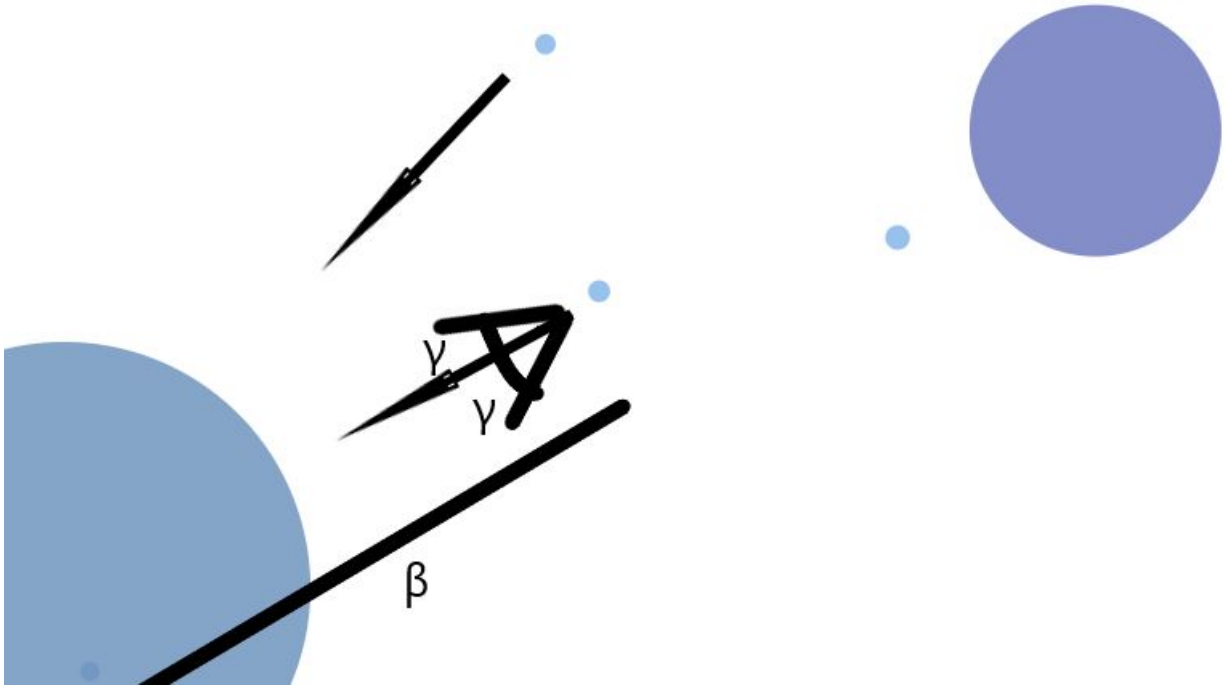


Wizualizacja działania algorytmu imperialistycznego

Tomasz Siemek

Politechnika Śląska Wydział Matematyki Stosowanej

Zhostowana aplikacja:

https://tomsiemek.github.io/imperialistic-competitive-algorithm-visualisation/

Kod projektu:

https://github.com/tomsiemek/imperialistic-competitive-algorithm-visualisation

Tomasz Siemek
15/01/2020

Wstęp

Algorytm imperialistyczny (ang. imperialistic competitive algorithm - ICA) jest heurystyką populacyjną, która stosuje mechanizmy znane z algorytmów rojowych jednak występuje tam również zjawisko konkurencji. Nazwa i zasada działanie inspirowana imperializmem - zjawiskiem uzależniania słabszego narodu/państwa od silniejszego państwa/narodu.

Jego unikatowy mechanizm działania i ciekawość jak wyglądałby taki algorytm “walki imperiów” z “lotu ptaka” zainspirowały mnie do zbudowania aplikacji wizualizującej działanie algorytmu.

Część teoretyczna

Skrócony opis działania algorytmu

1) Wygenerowanie N punktów - państw
2) N` najlepszych to imperia
3) Przydzielenie do imperiów(metropolii) kolonii w ilości proporcjonalnej do jakości rozwiązania jakie reprezentują
4) W każdym kroku:
    a) Asymilacja - przybliżenie kolonii do metropolii
    b) Jeśli kolonia prezentuje lepsze rozwiązanie od obecnej metropolii uczyń ją metropolią imperium
    c) Usunięcie imperiów bez kolonii
    d) Rywalizacja - transfer najsłabszej kolonii z najsłabszego imperium do losowego imperium (z prawdopodobieństwem proporcjonalnym do jego siły)
    e) Jeśli zostało jedno imperium lub osiągnięto zadaną ilość iteracji przerwij
    f) * Rewolucje - losowe zmiany położenia państw
    g) * Mariaż imperiów o metropoliach w bardzo podobnym położeniu

* rewolucje i mariaż nie zostały zaimplementowane, nie są one konieczne do prezentacji

działania

Obliczanie siły imperium

$$C_n = c_n - min\{c_i\}$$

C_n - znormalizowana siła państwa

$$T_n = C_n + \frac{\alpha}{k_n} \sum_{i=1}^{k_n} c_{ni}$$

T_n - całkowita siła imperium

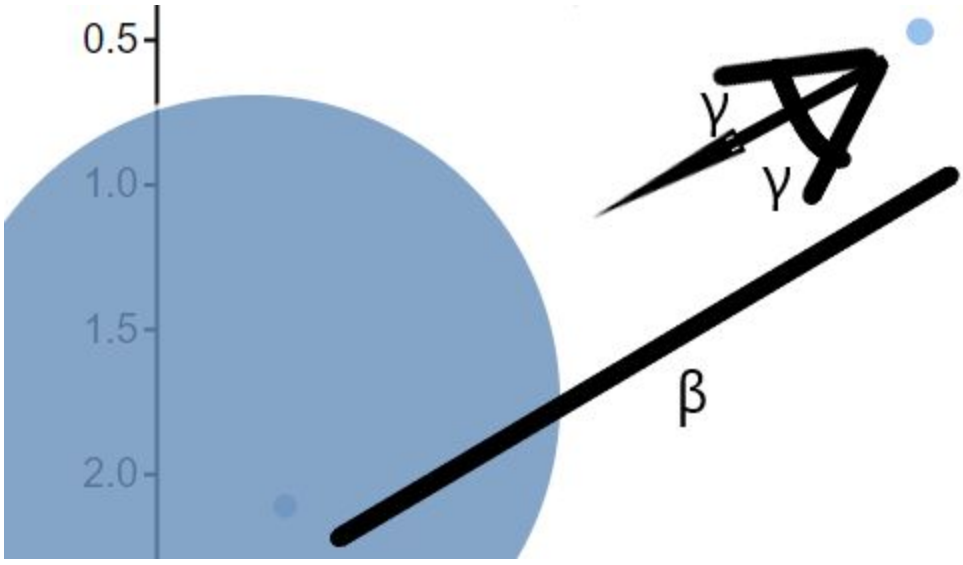
C_n - znormalizowana siła n-tego imperium

α - parametr wpływu siły kolonii na siłę imperium

k_n - ilość kolonii n-tego imperium

c_{ni} - i-ta kolonia n-tego imperium

Parametry β i γ



Obrazek 1. Beta i gamma przy asymilacji

Parametr β

Mnożnik maksymalnej odległości skoku jaki wykonuje kolonia w procesie asymilacji tzn. Kolonia przesunie się o odległość z zakresu [0, β * dist(kolonia, metropolia)].

Parametr γ

Maksymalne odchylenie jakie będzie miał wektor przesunięcia od linii łączącej środki metropolii i kolonii. Tzn. faktyczne odchylenie będzie z przedziału [0,γ].

Część praktyczna

Technologia

Do wizualizacji wykorzystano język javascript+html+css. Jest to oczywisty wybór z powodu uniwersalności - do otworzenia wystarczy przeglądarka i popularności - mnogość bibliotek i bogata dokumentacja.

W tworzeniu wykresu wykorzystano d3.js - https://d3js.org/ . Biblioteka znacznie ułatwiająca wizualizację różnych typów danych. Co ważne generuje ona grafikę wektorową - skalowalną w nieskończoność. Pojedyncze elementy danych są w formie węzłów DOM - można wchodzić z nimi w interakcje.

Do parsowania formuł funkcji wykorzystano biblioteke math.js - https://mathjs.org/ .

Opis działania

Funkcja Imperial

```
function* Imperial(range, N, NumberOfEmpires, iterations = 1000, alpha = 0.5, beta = 2, gamma = 0.01 * Math.PI) {
```

Obrazek 2. Nagłówek funkcji Imperial

Funkcja (generator) implementująca ICA - Imperial - każde poproszenie o kolejną wartość zwraca dane po przejściu pół iteracji (jedno po przejściu tylko asymilacji a drugie po przejściu całej iteracji) zapewnia to większą płynność animacji - wszystko nie dzieje się na raz, tylko iteracja jest podzielona na dwa kroki.

Użycie generatora pozwala na stopniową(leniwą) ewaluacje (ang. Lazy evaluation) tzn. Obliczamy kolejne iteracje dopiero gdy je potrzebujemy zapewniając płynność programu i potencjalnie zapobiegając zbędnym obliczeniom.

Funkcja GetData

```
for (let i = 0; i < iterations; i++) {
    Assimilate();
    empires.sort(sortNations);
    colonies.sort(sortNations);
    yield nations; // zwrocenie polozen po asymilacji
    UpdateColonyMetropolisRelation();
    RemoveEmpiresWithoutColonies();
    if(empires.length == 1) {
        break;
    }
    ImperialisticCompetition();
    empires.sort(sortNations);
    colonies.sort(sortNations);
    yield nations; // zwrocenie polozen po jednej iteracji

}
```

Obrazek 3. Instrukcje dla każdej iteracji algorytmu

```
function* GetData(config) {
```

Obrazek 4. Nagłówek funkcji GetData

```
while (!next.done) {
    const nations = next.value;
    const colonies = GetColonies(nations).sort(sortNations);
    const empires = GetEmpires(nations).sort(sortNations);
    yield MapDots(nations,empires, colonies, range) // zwrocenie danych do rysowania przed przeskalowaniem
    range = DetermineRange(empires, colonies)
    yield MapDots(nations,empires, colonies, range) // zwrocenie danych do rysowania po przeskalowaniem
    next = imperial.next();
}
```

Obrazek 5. Główna pętla funkcji GetData

Funkcja GetData obudowuje funkcje Imperial - działa ona jako łącznik pomiędzy działaniem algorytmu a wizualizacją. Tłumaczy ona współrzędna matematyczne na współrzędne na wykresie i przypisuje odpowiednie średnice wielkości proporcjonalnie do wartości rozwiązania i rodzaju państwa(metropolia lub kolonia). GetData również jest generatorem i zwraca wartości punktów przed przeskalowaniem i po - zapewniając większą płynność animacji.

W sumie na każdą iteracje algorytmu przypadają cztery klatki: po asymilacji przed skalowaniem, po asymilacji po skalowaniu, po całej iteracji przed skalowaniem i po całej iteracji po skalowaniu.

Sterowanie prędkością animacji

Szybkość animacji sterowana jest suwakiem.

Obrazek 6. Suwak regulujący prędkość animacji

Wartość suwaka jest tłumaczona do zmiennej DELAY. (Tak naprawde wartości suwaka są ujemne, ażeby użytkownik myślał że operuje szybkością a tak naprawdę operuje czasem rysowania klatek).

```
    DELAY = -1 * this.value
```

Obrazek 7.

DELAY jest argumentem funkcji setInterval (dostępna w każdej przeglądarce). Funkcja setInteval wywołuje zadaną funkcję co dany interwał czasowy.

```
function Resume() {
    intId = setInterval(DoSth, DELAY);
    SetStateButtonText("STOP")
}
```

Obrazek 8. DELAY przekazany jako argument setInterval

DELAY jest również argumentem jako długość animacji przejścia jednego stanu do drugiego (na przykład procesu asymilacji).

```
const circle = svg.selectAll("circle").data(newData);
circle.exit().remove();
circle.enter().append("circle")
    .merge(circle)
    .transition()
    .duration(DELAY)        You, 17 days ago • Working ra
    .attr("class", "bubbles")
    .attr("r", d => d.r)
    .attr("cx", d => d.x)
    .attr("cy", d => d.y)
    .style("fill", d => d.color)
```

Obrazek 9. Fragment przekazywania DELAY do fragmentu aktualizującego przestrzeń do rysowania.

Mapowanie państw na bąbelki (punkty) na wykresie

Zmapowane wartości są obliczone wg wzoru (wzór na y jest analogiczny):

$$x' = \frac{W(x - x_{min})}{X_{max} - X_{min}},$$

gdzie W - szerokość obszaru rysowania, X_{max}, X_{min} - wycinek przestrzenii jaki będzie rysowany.

```
const nationsMapped = nations.map( n => {
    let result = {
        x: widthMultiplier * (n.x - range.x.start),
        y: heightMultipler * (n.y - range.y.start),
        value: n.val,
        id: n.id,
        realX: n.x,
        realY: n.y
    }       You, 17 days ago • Working raw animation
    if(IsColony(n)) {
        let i = colonies.indexOf(n)
        result.r = GetRadius(i, colonies.length, COLONY_BASE_RADIUS)
        result.color = empires.find(el => el.id == n.metropolis).color
        result.colonies = null
    }
    else {
        result.color = MakeColorDarker(n.color)
        result.colonies = GetColoniesFromEmpire(n, colonies)
        result.r = GetRadius(result.colonies.length, colonies.length, METRO_BASE_RADIUS, METRO_ADD_RADIUS)
    }
    return result
})
```

Obrazek 10. Fragment kodu z mapowaniem wartości na dane do wykresu.

Pełny kod aplikacji można znaleźć tutaj:
https://github.com/tomsiemek/imperialistic-competitive-algorithm-visualisation

Działającą aplikację tutaj:

https://tomsiemek.github.io/imperialistic-competitive-algorithm-visualisation/