# Beyond Worst-Case Analysis

https://dl.acm.org/doi/pdf/10.1145/3232535

## Introducere

Compararea algoritmilor este dificila. In cazul masurarii performantei a 2 algoritmi, cum ar fi timpul de rulare sau calitatea solutiei, fiecare algoritm va avea rezultate mai bune pe anumite instante de input. De exemplu algoritmul de “insertion sort” este mai rapid decat “merge sort” pe array-uri deja sortate dar mai incet pe multe alte input-uri . Cum putem stabili ca un algoritm este “mai bun” decat altul atunci cand nu putem compara performanta ?

Analiza cazului cel mai nefavorabil este o metoda prin care performanta unui algoritm este determinata ca fiind cea mai proasta performanta pe orice input de o anumita dimensiune. Algoritmul “mai bun” este cel care se descurca mai bine in cazul cel mai nefavorabil . “Merge sort-ul” care are in cazul cel mai nefavorabil complexitatea timp teta(n log n) pentru vectori de dimensiune n , este mai bun in acest sens decat “insertion sort-ul” care are complexitatea teta(n^2) .

Analiza cazului cel mai nefavorabil poate fi o abordare in genul “Legilor lui Murphy” in analiza algoritmilor pentru ca este prea bruta pentru a da informatii despre cat de potrivit este un algoritm pentru rezolvarea unor probleme de programare liniara, clustering, caching sau antrenarea retelelor neuronale .

## Metoda simplex pentru problema de programare liniara

Probabil cel mai faimos esec pentru analiza worst-case in cazul programarii liniare, problema optimizarii unei functii liniare in raport cu niste constrangeri liniare . Metoda simplex creata de George Dantzig in 1947 care implica o rezolvare de tip greedy local search . Timpul de rulare se scleaza modest cu dimensiunea inputului si paote rezolva programe liniare cu milioane variabile de decizie si constrangeri . Aceasta performanta observata empiric sugereza ca metoda simplex poate rezolva orice program linear intr-un timp polynomial . S-a aratat prin exemple ca existe programe care forteaza metoda simplex sa ruleze intr-un timp exponential numarului de variabile de decizie. Acest lucru ilustreaza prima capcana a analizei cazului cel mai nefavorabil . Timpul de executie al metodei simplex este polynomial pentru in toate aplicatiile practice, in ciuda complexitatii exponentiale prezise in cazul cel mai nefavorabil. Cu atat mai mult, algoritmul cu complexitate polinomiala pentru cazul cel mai nefavorabil in programarea liniara, method elipsoidului, nu este competitiv cu metoda simplex in practica.

## Clustering si problem NP-dificile de optimizare

Clusterizarea este o forma de invatare automata nesupervizata ( gasirea unui pattern intr-un set de date care nu este “labeled” – etichetat ) , unde scopul (informal) este de a partitiona o multime de puncta in mai multe grupuri “coerente” .

De exemplu, scopul ar putea fi sa alegem k centroizi astfel incat sa minimizam suma distantelor dintre puncte si cel mai apropiat centroid (k-median) sau suma patratelor distantelor (k-means) . Aproape toate problemele de optimizare definite pentru clusterizare sunt NP-dificile. In practica, clusterizarea nu este vazuta ca o problema dificila. Algoritmi precum algoritmul lui Lloyd pentru k-means si variatiile lui, returneaza deobicei clustere intuitiv corecte. Cum putem “impaca” dificultatea marita a clusterizarii in cazul cel mai nefavorabil cu succesul dobandit empiric pe care il au algorimtii relative simpli? O posibila explicatie ar putea fi : “clusterizarea este dificila doar atunci cand nu conteaza” . De exemplu, daca instantele considerate a fi dificile a unei problema de clusterizare NP-dificila arata precum o gramada de puncte, nestructurate, aranjate “random”, cui ii pasa? Use-case-ul comun pentru algoritmii de clusterizare este pentru puncte care reprezinta imagini, documente, proteine sau obiecte care a caror grupare care “are sens” exista.

## Politici de inlocuire cache

Consideram un system cu memorie mica dar rapida (cache-ul) si o memorie mare dar lenta. Datele sunt organizate in blocuri numite “pagini”. In cache incap pana la “k” pagini diferite la un moment dat. Un request catre o pagina poate fi fie un “cache hit” (daca pagina era deja in cache) sau un “cache miss” ( daca pagina nu era deja in cache, caz in care trebuie adusa in cache). Daca cache-ul este plin, atunci una din pagini trebuie eliminate. O politica de inlocuire cache este un algoritm care face aceste decizii de eliminare a paginilor din cache. Orice manual recomanda politica LRU (least recently used ), care elimina pagina a carei referinta este cea mai indeparata in trecut deoarece in lumea reala secventele de request-uri tind sa prezinte o anumita tendinta si anume ca e foarte probabil ca paginile catre care s-a facut request recent sa fie cerute din nou in curand. Politica LRU foloseste trecutul recent pentru a face o predictie pentru viitorul apropiat. Empiric, algoritmul sufera de mai putine “cache miss” decat alte politici precum FIFO (first-in first-out).

Exista o serie de algoritmi care trebuie sa proceseze inputul asa cum este primit de-a lungul timpului ( ca politicile de cache ) . Una din primele observatii care s-a facut asupra aceste categorii de algoritmi numita si “online algorithms” a fost ca analiza cazului cel mai nefavorabil aplicata direct nu aduce nici o informatia utila despre performanta politicii de inlocuire a cache-ului. Pentru orice politica determinista si o marime a cache-ului “k”, exista o secventa care va face a rata de miss sa fie 100% desi algoritmul lui Belady considerat o politica de inlocuire cache optima are un fault rate de (1/k)% .

Superioritatea LRU fata de FIFO este dovedita epiriic, datorita structurii speciale pe care o are o secventa de request-uri in viata reala

## Concluzii

Nu exista algoritmi care sa fie buni pentru toate tipurile de probleme computationale si chiar la nivel de problema speficica nu exista un algoritm magic, mereu optim – cel mai bun algoritm pentru o anumita problema depinde de cele mai relevante instante ale problemei . In acest domeniu raman inca multe diferente neexplciate intre performanta teoretica si cea empirica unui algoritm