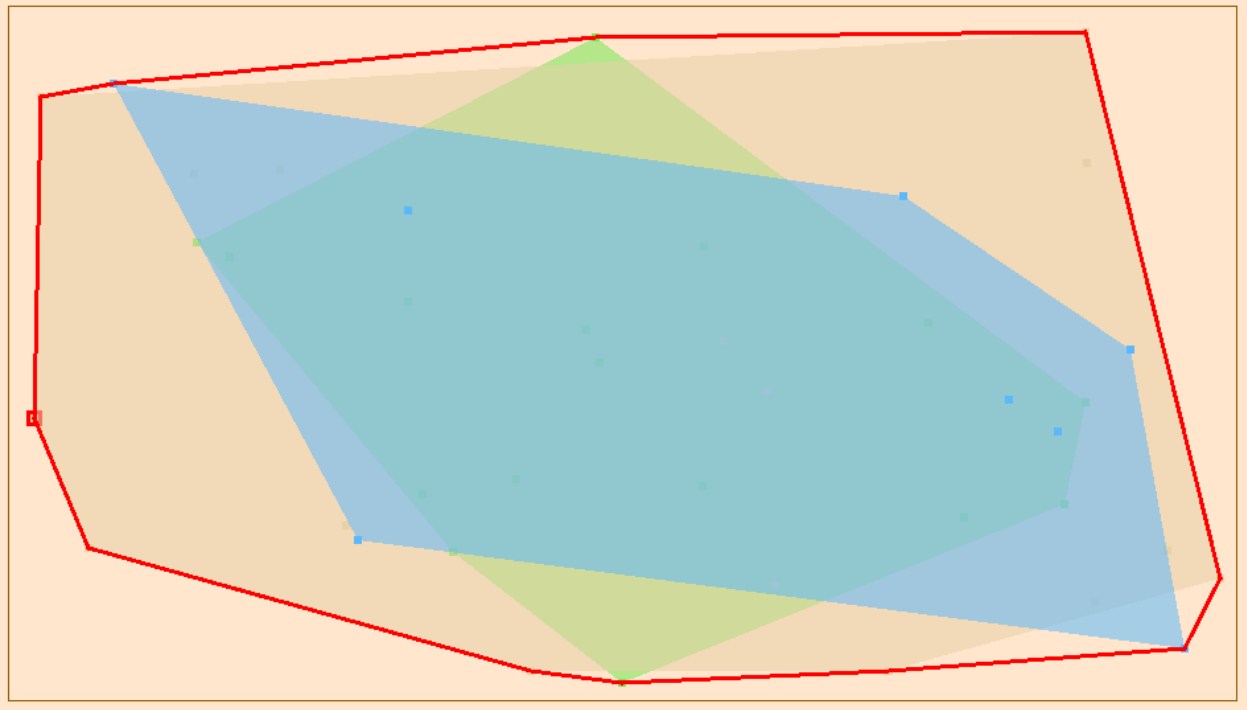
Geometrijski algoritmi @ MATF

Algoritam za izračunavanje konveksnog omotača za dati skup tačaka – Chan’s algorithm

Milica Đurić



Opis problema

Skup tačaka X u ravni je konveksan ako za svake dve tačke A i B iz skupa X važi da svaka tačka duži AB pripada skupu X. Konveksni omotač skupa tačaka je najmanji konveksan skup tačaka koji sadrži X. Rub konveksnog omotača skupa X je podskup skupa X. Za konačan skup tačaka u ravni, konveksni omotač je (konveksni) poligon. Za fiksiran skup tačaka, konveksni omotač je određen jednoznačno.

Zadatak koji treba ispuniti u ovom radu glasi – za dati skup tačaka odrediti njegov konveksan omotač. Tačke koje predstavljaju ulaz u algoritam mogu da se zadaju slučajnim izborom ili putem fajla gde se zadaju koordinate tačaka.

Ulaz: *skup od n tačaka u ravni*

Izlaz: *skup tačaka koje predstavljaju rub konveksnog omotača*

Naivno rešenje problema

Naivno rešenje koristi pristup u kome se proveravaju sve moguće trojke tačaka i njihova orijentisanost. Ovo rešenje je složenosti O(n^3) gde je n broj tačaka koje su prosleđene kao ulaz. Složenost potiče iz činjenice da za svaki od *n(n-1)* par tačaka treba ispitati *n-2* tačke. Algoritam ne daje uvek tačan rezultat zbog zaokruživanja u aritmetici sa pokrenim zarezom. Naredna slika predstavlja psedokod naivnog algoritma i preuzeta je iz knjige “Računarska geometrija” profesora Janičića.

Algoritam: KonveksniOmotac

Ulaz: Skup tačaka u ravni $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$

Izlaz: Lista L koja sadrži temena konveksnog omotača u smeru kazaljke na satu

```
1: E := 0
2: za sve parove različitih tačaka P i Q iz skupa P
3:     valid := true
4:     za sve tačke R iz P koje su različite od P i Q
5:         ako je trojka PQR negativno orijentisana onda
6:             valid := false;
7:     ako je valid onda
8:         dodaj usmerenu duž PQ u E
9: Od skupa stranica E konstruiši listu temena L tako da kreiraju poligon u
   negativnoj orijentaciji
```

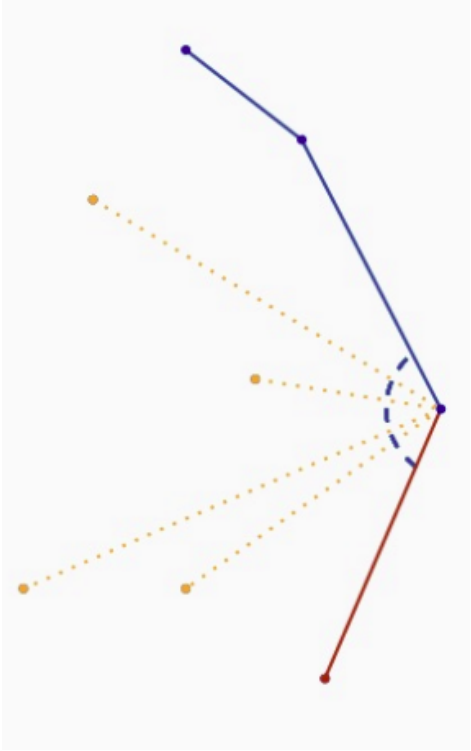
Chanov algoritam

Algoritam je pronašao Timothy Chan 1993. godine. Da bi se opisao ovaj algoritam, pre toga je potrebno opisati osnovne ideje algoritama koje koristi kao deo - Gremov algoritam i algoritam uvijanja poklona.

1. Algoritam uvijanja poklona

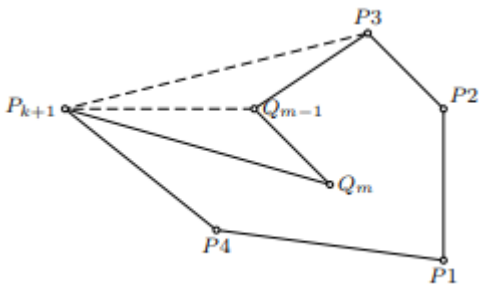
Glavna ideja glasi - uzimajući poslednju stranicu *pq* konveksnog omotača, sledeća stranica *qr* koja se dodaju u konveksni omotač će maksimizovati ugao *pqr.* U svakom koraku posmatramo svih *n* tačaka. Koraka ima koliko i tačaka u konveksnom

omotaču – h. Odatle sledi da je složenost ovog algoritma O(nh). Na sledećoj slici prikazan je korak algoritma i stranica koja se bira je označena crvenom bojom.



2. Gremov algoritam

Prvo što treba uraditi je sortirati tačke iz P u suprotnom smeru od kazaljke na satu u odnosu na tačku koja se nalazi skroz desno. Čuva se trenutno stanje konveksnog omotača u Q. Prolazimo kroz sve tačke u kreiranom redosledu i za svaku tačku ispitujemo ugao koji gradi sa poslednje dve tačke u konveksnom omotaču. Ukoliko smo pregledali do sada k tačaka i izračunali da se za tih k tačaka u konveksnom omotaču nalazi m tačaka, onda u sledećem koraku posmatramo ugao $Q_{m-1}Q_mP_{k+1}$. Ukoliko je on manji od opruženog, tačka P_{k+1} ulazi u konveksan omotač, a ukoliko je veći, ona se takođe dodaje, ali je potrebno proveriti koje tačke, koje su prethodno dodate u konveksan omotač, treba izbaciti. Treba izbaciti one tačke Q_m za koje važi da trojka tačaka $Q_{m-1}Q_mP_{k+1}$ ima negativnu orijentaciju. Na sledećoj slici je prikazana vizuelizacija jednog od koraka u Gremovom algoritmu (takođe preuzeta iz već pomenute knjige). Tačke Q_m i Q_{m-1} je potrebno izbaciti iz Q.

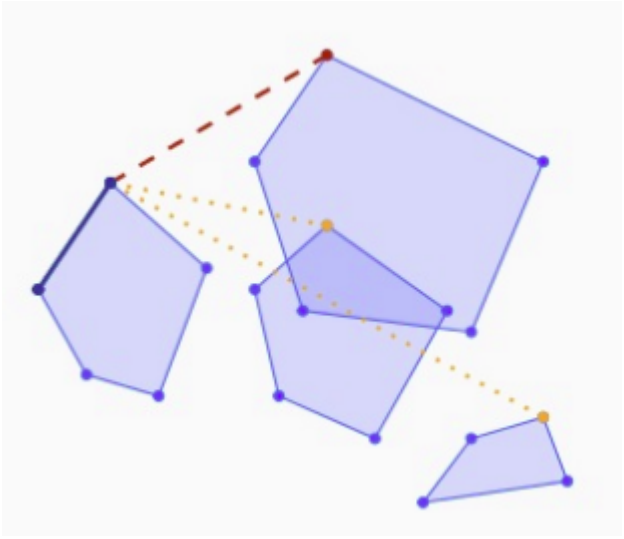


Složenost datog algoritma je O(*nlogn*).

Chanov algoritam je algoritam koji ima najbolju složenost među svim algoritmima za pronalaženje konveksnog omotača skupa tačaka. Njegova složenost je O(*nlogh*), gde je *h* broj tačaka koje ulaze u konveksan omotač.

Ideja se zasniva na tome da neke tačke nikako ne mogu da budu deo konveksnog omotača, tako da možemo da odbacimo njih i da ubrzamo algoritam uvijanja poklona. Ovaj algoritam jeste kombinacija algoritma uvijanja poklona, Gremovog algoritma i algoritma binarne pretrage.

Prvi korak je podeliti ulazni skup tačaka u *n/m* (kasnije će biti rečeno šta je *m*) grupa tačaka veličine najviše *m* (nekad je potrebno *n/m + 1* grupa ukoliko *n* nije deljivo sa *m*) i naći najlevlju tačku među svim tačkama i ubaciti je u omotač. Zatim treba Gremovim algoritmom izračunati konveksan omotač H_i svake grupe tačaka. Složenost ovog dela je O(*(n/m)*(mlogm)*) = O(*nlogm*). Čuva se trenutni konveksan omotač Q skupa ulaznih tačaka. Ukoliko je Q_k poslednja tačka u ovom konveksnom omotaču, a Q_{k-1} pretposlednja tačka, onda za svaki H_i binarnom pretragom pronaći tangentu iz Q_k koja maksimizuje ugao $Q_{k-1}Q_kH_{ij}$, gde je H_{ij} tačka iz skupa H_i kroz koju prolazi tangenta. Zatim iz skupa izabranih tangentni izabrati onu koja maksimizuje ugao $Q_{k-1}Q_kP_i$, gde je P_i neka od tačaka $H_{ij.}$ Složenost jednog ovakvog koraka je O(*(n/m) * logm*). Na sledećoj slici prikazan je korak biranja tangenti.



Crvenom bojom je tangenta koja će biti izabrana jer ona maksimizuje dati ugao.

Složenost celog ovakvog algoritma jeste: O(*nlogm + (h*(n/m)*logm)*). Ukoliko je *m=h*, onda je složenost baš O(*nlogh*). Ali, *h* nije poznato. Zato je ideja napraviti više iteracija ovog algoritma pogađajući vrednost *h,* odnosno uzimajući različite vrednosti za parametar *m*. Svaka iteracija se završi u O(*nlogm*) vremena. Ukoliko je *m* < *h*, onda će algoritam ući u sledeću iteraciju, u suprotnom, algoritam se završava. Problem je na koji način uvećavati vrednost *m*. Postoji efikasan način, a to je: prvo uzmemo *m=2.* U iteraciji *t* uzima se vrednost

$$, m = \min\left(n, 2^{2^t}\right)$$

i u tom slučaju se najviše napravi O(*loglogh*) iteracija. Zato je ukupno vreme izvršavanja algoritma

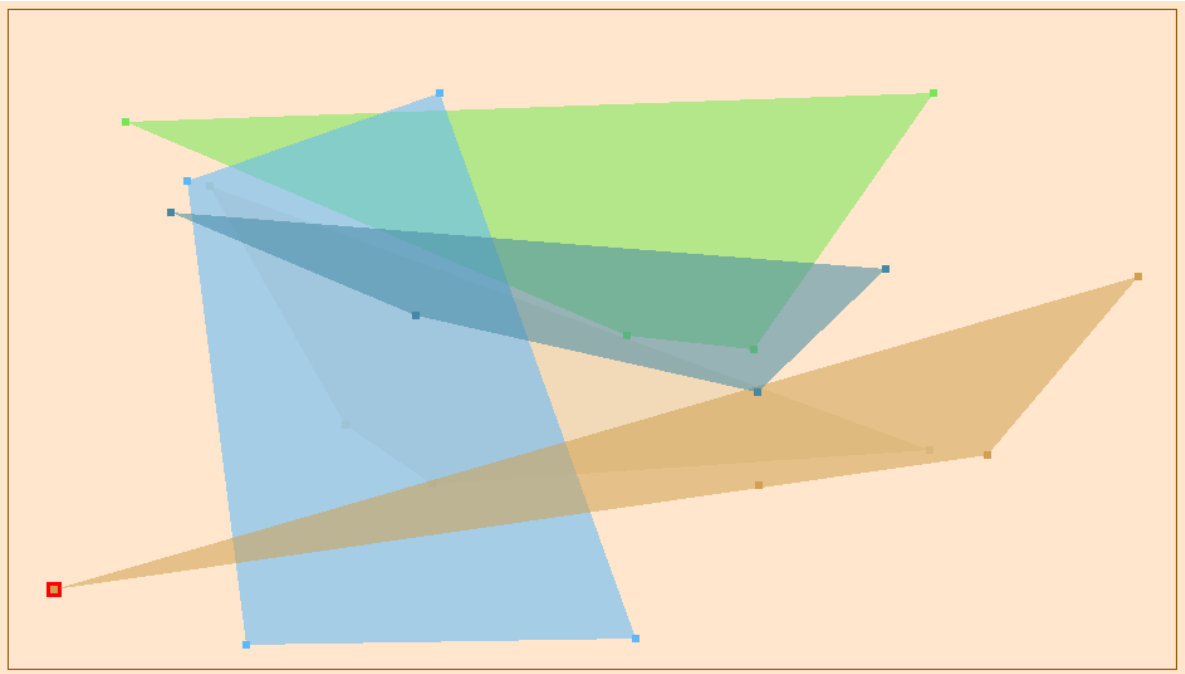
$$\sum_{t=1}^{\lceil \log\log h \rceil} O\left(n \log(2^{2^t})\right) = O(n) \sum_{t=1}^{\lceil \log\log h \rceil} O(2^t) = O\left(n \cdot 2^{1+\lceil \log\log h \rceil}\right) = O(n \log h).$$

Na sledećoj slici je prikazana implementacija jedne iteracije algoritma. Vidimo da se iz čitavog algoritma izlazi ukoliko dođe do situacije da smo se ponovo vratili u početnu tačku ili ako konveksan omotač sadrži sve tačke skupa (zbog specijalnog slučaja u kome su date samo kolinearne tačke).

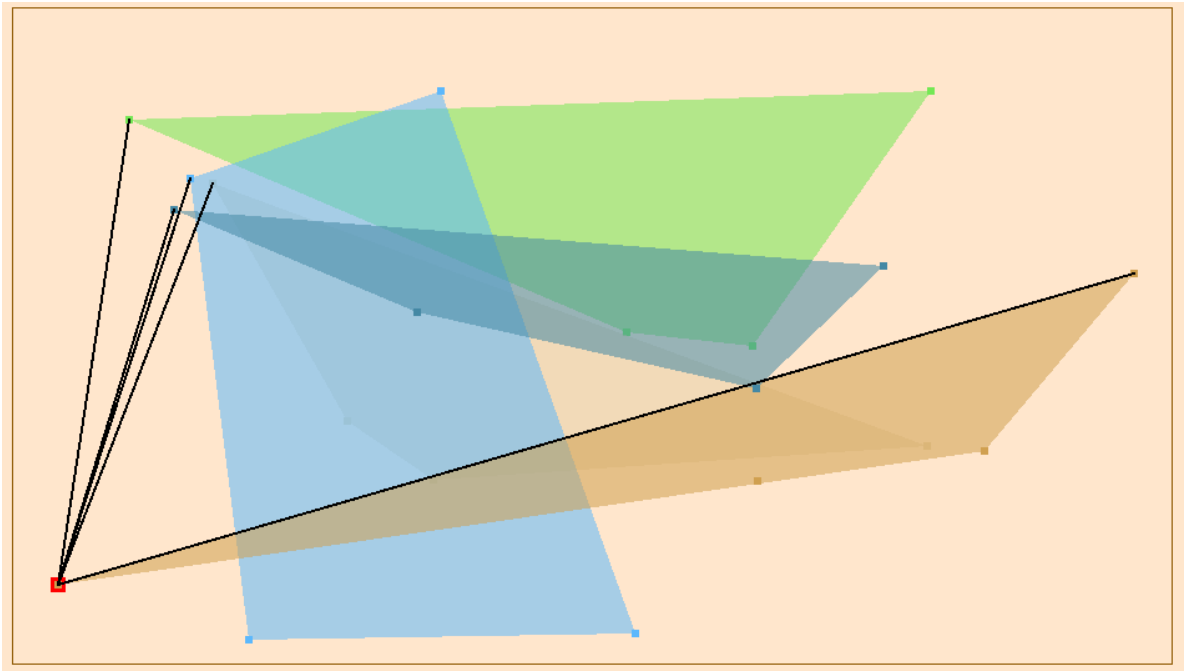
```
unsigned temp = 0;
do{
    QPoint bestTangent(0,0);
    for(int i=0; i < numOfParts;i++){
        int tangent = findTangent(_hullsOfParts[i]);
        _tangents[i] =  _hullsOfParts[i][tangent];
        //ako smo prvi put u petlji, moramo da postavimo sta je najbolja do sad tangenta
        //a sledecih puta proveravamo da li je neka bolja
        if((i == 0) || compareTangents(_tangents[i], bestTangent) )
            bestTangent = _tangents[i];
    }
    _convexHull.push_back(bestTangent);
    _beforeP = _P;
    _P = bestTangent;
    _tangentsFound = true;
    //zbog crtanja svih tangenti iz tacke
    AlgorithmBase_updateCanvasAndBlock();
    _drawConvexHull = true;

    temp++;
    //ovaj drugi uslov nam je bitan zbog kolinearnih tacaka,
    //slucaj kada imamo samo kolinearne tacke jer nece prepoznati da treba da stavi i prvu tacku na pocetak
    if((_P.x() == _firstPoint.x() && _P.y() == _firstPoint.y()) || (temp+1) == _points.size()){
        if((temp+1) == _points.size() && !(_P.x() == _firstPoint.x() && _P.y() == _firstPoint.y())){
            _convexHull.push_back(_firstPoint);
            AlgorithmBase_updateCanvasAndBlock();
        }
        _tangentsFound = false;
        AlgorithmBase_updateCanvasAndBlock();
        complete = true;
        break;
    }
    _tangentsFound = false;
    _drawConvexHull = false;
}
while(temp < m);
```

Vizuelizacija algoritma

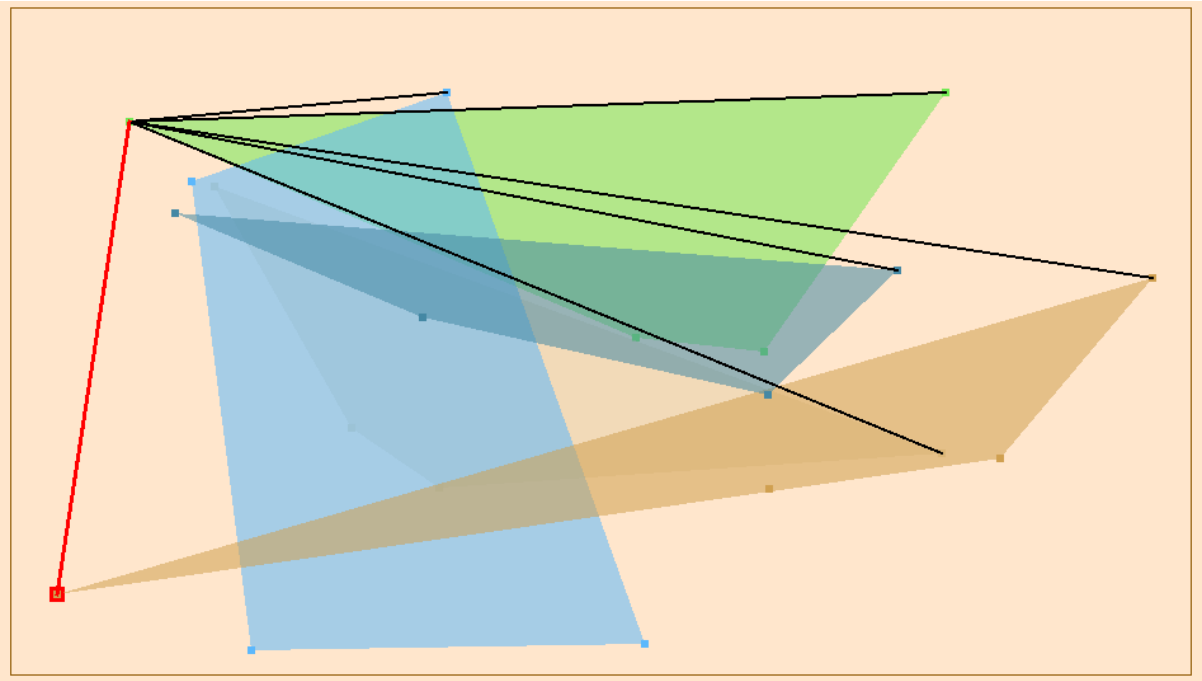
U ovoj sekciji biće prikazana vizulizacija algoritma za ulaz veličine $n=20$. Prvo se za m bira vrednost 4, pa se ulaz deli na 5 grupa od po 4 tačke. Za svaku grupu se računa konveksan omotač. Crvenom tačkom je označena najlevlja tačka, odnosno prva tačka konveksnog omotača.



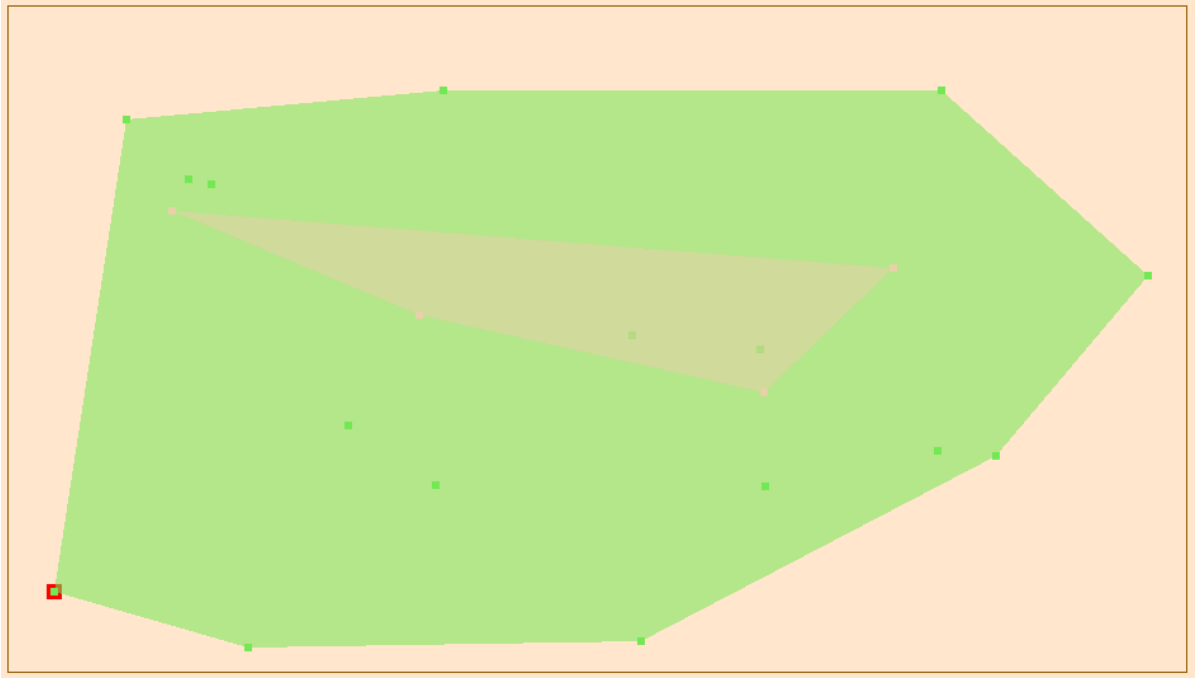
Zatim se kreće sa pronalaženjem tangenti za svaki skup.



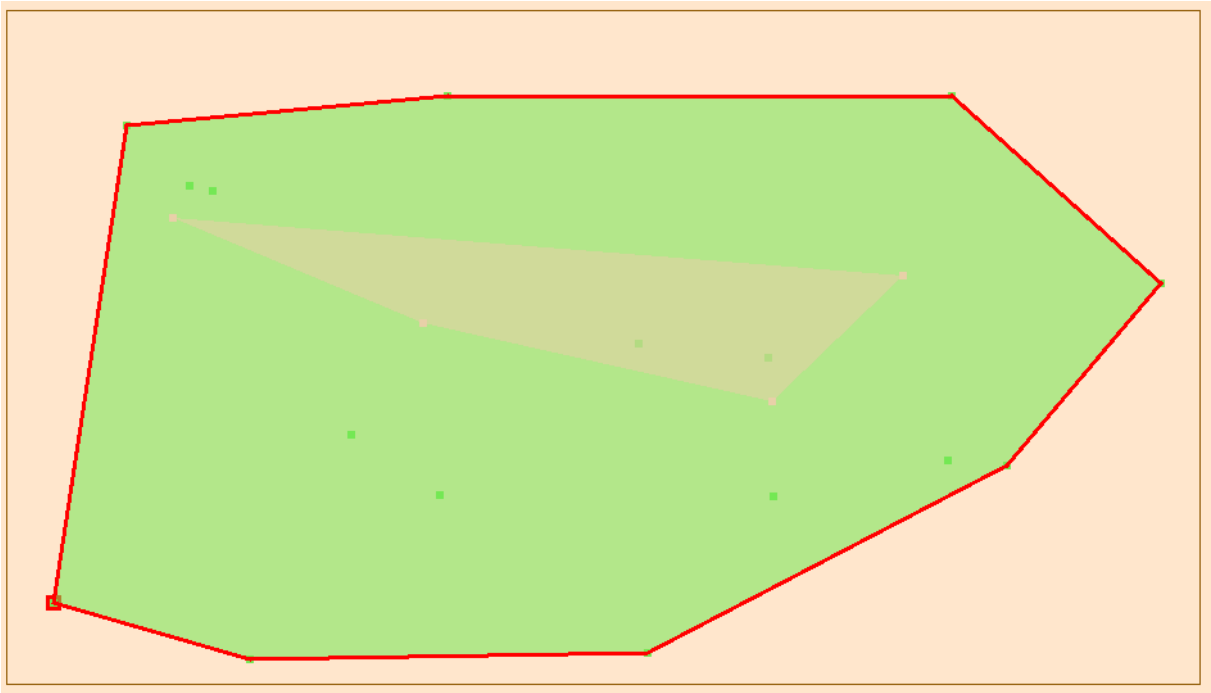
Bira se tangenta koja maksimizuje ugao (prva tačka u omotaču je najlevlja tačka, a za tačku pre nje se uzima tačka (0, -10000)).



Sa biranjem tangenti se nastavlja sve dok se shvati da u omotaču imamo m tačaka, a čitav konveksni omotač još uvek nije pronađen. U tom trenutku, ulaz se resetuje, a za novu vrednost *m* se uzima 16. Imamo jednu grupu od 16 elemenata, a drugu od 4 elementa. Za obe grupe Gremovim algoritmom računamo konveksan omotač.

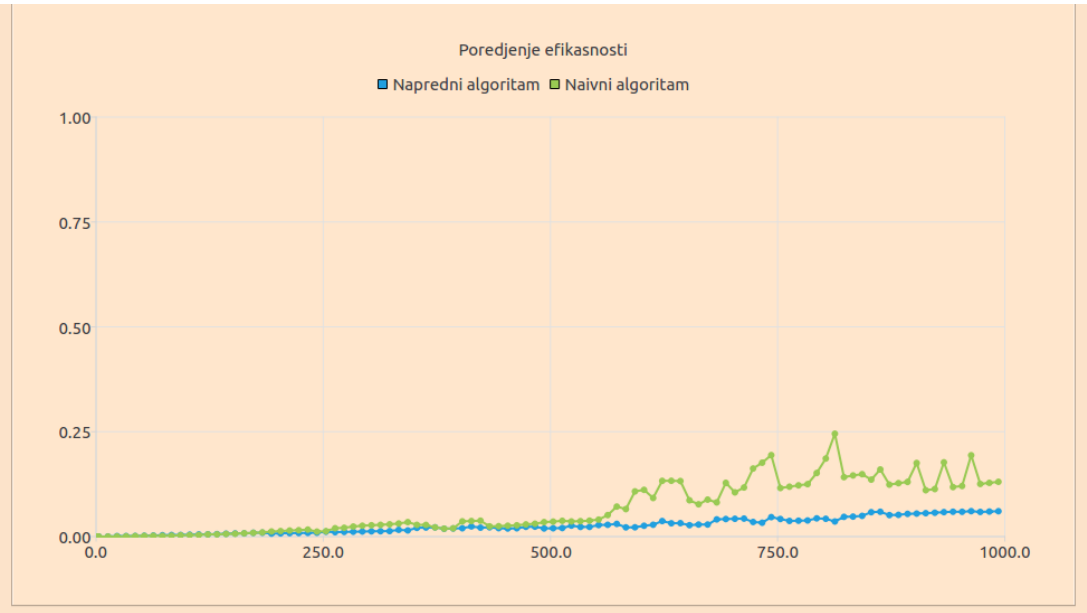


Traženje tangenti se zatim opet sprovodi, a algoritam se zaustavi u trenutku kada je se ponovo dođe do prve tačke u konveksnom omotaču. Traženi konveksan omotač je onda pronađen.

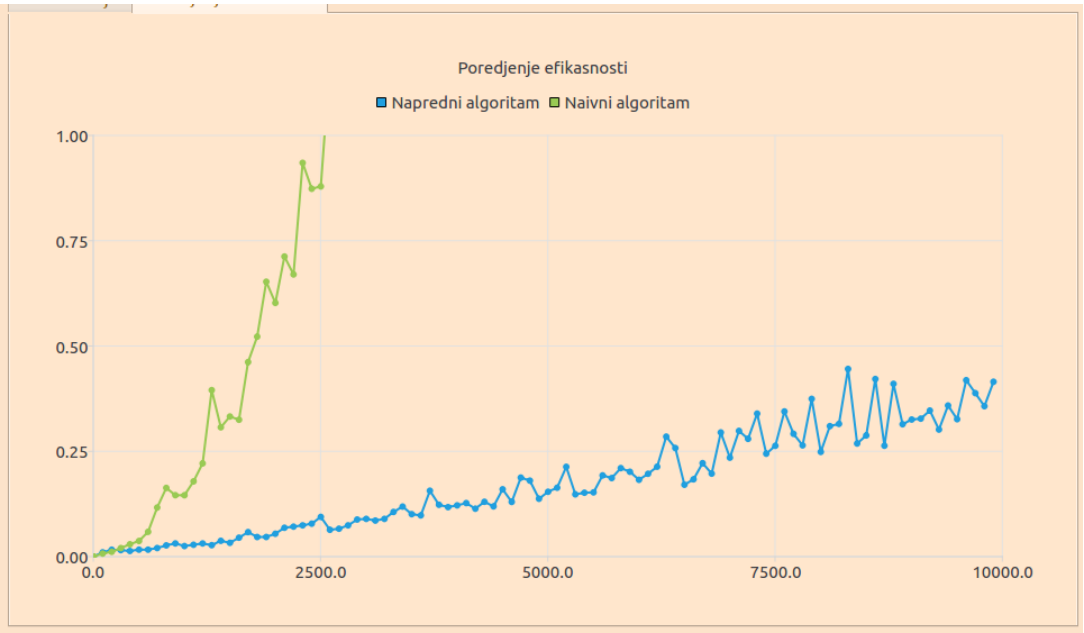


Poredjenje efikasnosti naivnog i naprednog algoritma

Ukoliko u programu stavimo maksimalnu dimenziju ulaza 1000, dobijamo sledeću vizuelizaciju poređenja efikasnosti:



Ukoliko u programu stavimo maksimalnu dimenziju ulaza 10000, dobijamo sledeću vizuelizaciju poređenja efikasnosti:



Vreme izvršavanja naivnog i optimalnog algoritma u odnosu na dimenziju ulaza dato je u sledećoj tabeli.

alg. / dim. ulaza	10	100	1000	5000	10000	50000
naivni	0.000047	0.003251	0.174418	4.803955	21.06887	562.2577
optimalni	0.000143	0.003980	0.043236	0.180666	0.378279	4.370436

Testiranje ispravnosti algoritma

Testiranje je izvršeno koristeći testove sa random generisanim tačkama i tačkama koje se čitaju iz fajla. Tačke koje se čitaju iz fajla su specijalni slučajevi u kojima su tačke kolinearne horizontalno, vertikalno, po dijagonali, ili kombinacija nekih od ovih slučajeva (neki od testova su preuzeti od QuickHull algoritma).

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
randomTest30	Programski generisan ulaz	Niz dimenzije 30	Poklapanje rezultata naivnog i

			naprednog algoritama
randomTest100	Programski generisan ulaz	Niz dimenzije 100	Poklapanje rezultata naivnog i naprednog algoritma
randomTest200	Programski generisan ulaz	Niz dimenzije 200	Poklapanje rezultata naivnog i naprednog algoritama
horizontalTest	Kolinearne tačke koje su paralelne sa x-osom	100 100 200 100 300 100 400 100 500 100	Poklapanje rezultata naivnog i naprednog algoritama
diagonalUpTest	Kolinearne tačke na dijagonala	50 250 100 200 150 150 200 100 250 50	Poklapanje rezultata naivnog i naprednog algoritama
diagonalDownTest	Kolinearne tačke na dijagonali	50 50 100 100 150 150 200 200 250 250	Poklapanje rezultata naivnog i naprednog algoritama
verticalTest	Kolinearne tačke koje su paralelne sa y-osom	100 100 100 200 100 300 100 400 100 500	Poklapanje rezultata naivnog i naprednog algoritama
rectangleTest	Ručno generisan ulaz u obliku pravougaonika	50 50 200 50 200 200 50 200	Poklapanje rezultata naivnog i naprednog algoritama
duplicatePointsTest	Nekoliko dupliranih tačaka	50 50 50 50 130 130 130 130 200 50 200 50 200 200 50 200	Poklapanje rezultata naivnog i naprednog algoritama

		50 200 50 200	
lessThanThreePoints	Manje od tri tačke	200 50 300 400	Poklapanje rezultata naivnog i naprednog algoritama
triangleHUpRTest	Kolinearne tačke u obliku trogla	50 100 100 100 200 100 300 100 500 300	Poklapanje rezultata naivnog i naprednog algoritama
triangleHDownRTest	Kolinearne tačke u obliku trougla	50 100 100 100 200 100 300 100 500 50	Poklapanje rezultata naivnog i naprednog algoritama
triangleVUpRTest	Kolinearne tačke u obliku trougla	100 100 100 200 100 300 100 400 300 500	Poklapanje rezultata naivnog i napredniog algoritama
triangleVDownRTest	Kolinearne tačke u obliku trougla	100 200 100 300 100 400 100 500 300 100	Poklapanje rezultata naivnog i naprednog algoritama
triangleHUpLTest	Kolinearne tačke u obliku trougla	100 300 200 200 300 200 400 200 500 200	Poklapanje rezultata naivnog i naprednih algoritama
triangleHDownLTest	Kolinearne tačke u obliku trougla	100 100 200 200 300 200 400 200 500 200	Poklapanje rezultata naivnog i naprednog algoritama
triangleVUpLTest	Kolinearne tačke u obliku trougla	100 400 200 100 200 200 200 300	Poklapanje rezultata naivnog i naprednog algoritama
triangleVDownLTest	Kolinearne tačke	100 100	Poklapanje

	u obliku trougla	200 200 200 300 200 400 200 500	rezultata naivnog i naprednog algoritama
triangleLDownTest	Kolinearne tačke u obliku pravouglog trougla	50 50 50 100 50 150 50 200 50 250 100 250 150 250 200 250 250 250	Poklapanje rezultata naivnog i naprednog algoritama
triangleRDownTest	Kolinearne tačke u obliku pravouglog trougla	250 50 250 100 250 150 250 200 250 250 50 250 100 250 150 250 200 250	Poklapanje rezultata naivnog i naprednog algoritama
triangleRUpTest	Kolinearne tačke u obliku pravouglog trougla	250 50 250 100 250 150 250 200 250 250 50 50 100 50 150 50 200 50	Poklapanje rezultata naivnog i naprednog algoritama
triangleLUpTest	Kolinearne tačke u obliku pravouglog trougla	50 50 50 100 50 150 50 200 50 250 100 50 150 50 200 50 250 50	Poklapanje rezultata naivnog i naprednog algoritama