Grid-peeling

Gašper Pust, Mitja Mandić

14. november 2020

1 Predstavitev problema

V projektu si bomo podrobneje ogledali konveksne ovojnice $m \times n$ mreže. Konveksna ovojnica množice je najmanjša konveksna množica, ki vsebuje dano množico. Najlažje si jo predstavljamo tako, kot da bi okoli elementov množice napeli elastiko - kar elastika obkroži, je konveksna ovojnica. Lupljenje konveksnih ovojnic mreže, oziroma angleško *grid - peeling* je proces, ko iz mreže iterativno odstranjujemo konveksne ovojnice. S simboli lahko to zapišemo takole: $P_0 = G_{n,m} = \{1, \ldots, n\} \times \{1, \ldots, m\}$. Naj bo $C_i = \mathcal{CH}(P_{i-1})$ za $i = 1, \ldots$. V_i naj bo množica vozlišč C_i - kot vozlišče razumemo točko, ki je na vogalu mreže (torej za katero bi zataknili elastiko). Naj bo sedaj $P_i = P_{i-1} \setminus V_i$. Začnemo torej z $n \times m$ mrežo in iterativno lupimo konveksne ovojnice, dokler ne odstranimo vseh točk.

V projektni nalogi bova s pomočjo simulacij opazovala v literaturi navedene številke za $n \times n$ mrežo - teorija napoveduje $\theta(n^{\frac{4}{3}})$ ovojnic. Za $n \times m$ mrežo v literaturi ni navedenih podatkov, zanimala naju bo morebitna povezava. Simulacije bova izvedla tudi za točke na neenakomerni mreži.

Po izvedenem eksperimentalnem delu, bomo rezultate analizirali in jih primerjali z rezultati iz literature. Zanimalo nas bo, kako drugačno je število ovojnic na $m \times n$ mreži v primerjavi s simetrično.

2 Orodja in algoritmi

2.1 Jarvisov obhod

Jarvisov obhod (angl. *Jarvis March*) ali algoritem zavijanja darila je postopek, ki dani množici točk poišče konveksno ovojnico v eni ali več dimenzijah (osredotočili se bomo na dve dimenziji). Algoritem se imenuje po R.A. Jarvisu, ki ga je objavil leta 1973. Časovna zahtevnost algoritma je $O(nh)$, kjer n predstavlja število vseh točk, h pa število točk, ki ležijo na konveksni ovojnici. V najslabšem primeru, ko so vse podane točke tudi elementi konveksne ovojnice, torej v primeru $h = n$, je njegova časovna zahtevnost $O(n^2)$. Jarvisov obhod se največkrat uporablja za majhne n ali pa v primeru, ko pričakujemo, da bo h zelo majhen glede na n.

```
def jarvis_march(points):
    # find the leftmost point
    a = min(points, key = lambda point: point.x)
    index = points.index(a)

    # selection sort
    l = index
    result = []
    result.append(a)
    while (True):
        q = (l + 1) % len(points)
        for i in range(len(points)):
            if i == l:
                continue
            # find the greatest left turn
```

```
            # in case of collinearity, consider the farthest point
            d = direction(points[l], points[i], points[q])
            if d > 0 or (d == 0 and distance_sq(points[i], points[
                l]) > distance_sq(points[q], points[l])):
                q = i
        l = q
        if l == index:
            break
        result.append(points[q])

    return result
```

2.2 Grahamov pregled

Alternativa prejšnjemu algoritmu je tako imenovani Grahamov pregled (angl. *Graham's scan*). Algoritem se imenuje po Ronaldu Grahamu, ki ga je objavil leta 1972. V primerjavi z Jarvisovim obhodom je Grahamov pregled hitrejši, saj ima časovno zahtevnost $O(n \log n)$.

```
def convex_hull_graham(points):
    TURN_LEFT, TURN_RIGHT, TURN_NONE = (1, - 1, 0)

    def cmp(a, b):
        return (a > b) - (a < b)

    def turn(p, q, r):
        return cmp((q[0] - p[0])*(r[1] - p[1]) - (r[0] - p[0])*(q
            [1] - p[1]), 0)

    def _keep_left(hull, r):
        while len(hull) > 1 and turn(hull[-2], hull[-1], r) !=
            TURN_LEFT:
            hull.pop()
        if not len(hull) or hull[ - 1] != r:
            hull.append(r)
        return hull

    points = sorted(points)
    l = reduce(_keep_left, points, [])
    u = reduce(_keep_left, reversed(points), [])
    return l.extend(u[i] for i in range(1, len(u) - 1)) or l
```

3 Rezultati

4 Zaključek