

# Minimum K-Center (p-center) Problem

Jovan Bjegovic 37/2021

## 1. Uvod

Minimum K-Center (p-center) problem predstavlja važan izazov u teoriji grafova i optimizaciji, koji se koristi za efikasno lociranje k-centara u grafu, pri čemu se minimizuje maksimalna udaljenost od bilo kog čvora do najbližeg centra. Ovaj problem ima široku primenu u oblasti transporta, rasporeda, kao i u mnogim industrijskim i tehnološkim aplikacijama gde je efikasno lociranje tačaka od kritičnog značaja. U ovom radu istražujemo različite algoritme koji rešavaju ovaj problem, sa posebnim fokusom na brzinu, tačnost i efikasnost algoritama.

## 2. Opis rešenja

Za rešavanje Minimum K-Center problema, implementirali smo nekoliko različitih algoritama koji variraju u pogledu računarske složenosti, preciznosti i metoda optimizacije. Algoritmi koje razmatramo uključuju:

1. **Brute-Force Algoritam**
2. **Pohlepni Algoritam (Greedy)**
3. **Genetski Algoritam**
4. **Algoritam K-sredina (K-means)**
5. **Simulirano kaljenje Algoritam**
6. **Algoritam optimizacije mravljevom kolonijom (Ant Colony Optimization)**

Svi ovi algoritmi imaju za cilj da efikasno lociraju k-centre u grafu, minimizirajući maksimalnu udaljenost (radius) između centra i svih ostalih čvorova u grafu. Različite pristupe testiramo na grafovima sa različitim parametrima, kako bismo izabrali najefikasniji algoritam za rešavanje ovog problema.

### 2.1 Brute-Force Algoritam

**Brute-force algoritam** je jednostavan, ali vrlo računski zahtevan metod koji podrazumeva isprobavanje svih mogućih kombinacija k-centara u grafu. Ovaj pristup osigurava tačno rešenje jer isprobava svaku moguću konfiguraciju, međutim, brzina izvršenja drastično opada kako broj čvorova i centara raste. Ovaj metod se često koristi kao referentni model, jer pruža tačno rešenje i može se koristiti za upoređivanje sa drugim, bržim, ali manje preciznim algoritmima. U situacijama kada je broj čvorova i centara relativno mali, brute-force algoritam može biti koristan zbog svoje jednostavnosti i tačnosti.

#### Prednosti:

- Tačno rešenje.
- Jednostavna implementacija.

#### Nedostaci:

- Veoma sporo za velike grafove.
- Računarska složenost eksponencijalno raste sa brojem čvorova i centara.

## 2.2 Pohlepni Algoritam (Greedy)

**Pohlepni algoritam** je pristup koji rešava problem iterativnim odabirom centra na osnovu trenutne konfiguracije grafova. U svakom koraku, algoritam bira čvor koji čini najviše smanjenje maksimalne udaljenosti od bilo kojeg čvora do najbližeg centra. Iako ovaj pristup ne garantuje uvek globalno optimalno rešenje, često daje vrlo dobra rešenja u praktičnim slučajevima i značajno je brži od brute-force algoritma. Pohlepan pristup je jednostavan i efikasan za probleme sa velikim brojem čvorova i centara, ali ponekad može da se zaglavi u lokalnim optimumima.

### Prednosti:

- Brz.
- Laka implementacija.
- Često daje dobre rezultate za većinu malih problema.

### Nedostaci:

- Ne garantuje optimalno rešenje.
- Može se zaglaviti u lokalnim minimumima.

## 2.3 Genetski Algoritam

**Genetski algoritam** je evolucioni algoritam koji koristi principe selekcije, ukrštanja i mutacije za pronalaženje optimalnih rešenja. U ovom algoritmu, populacija potencijalnih rešenja evoluira kroz nekoliko generacija. Svaka generacija proizvodi nove, "bolje" pojedince (rešenja) koji imaju manji maksimalni radijus, bazirano na kriterijumima selekcije. Genetski algoritam je vrlo fleksibilan i sposoban je da pronađe dobra rešenja čak i za velike grafove. Iako genetski algoritam obično ne garantuje globalno optimalno rešenje, on je efikasan u pronalaženju vrlo dobrih rešenja za složene probleme.

### Prednosti:

- Dobro funkcioniše za velike grafove.
- Može pronaći dobra rešenja u problemima sa velikim brojem varijabli.

### Nedostaci:

- Nema garanciju za globalno optimalno rešenje.
- Može biti računarski intenzivan.

## 2.4 Algoritam K-sredina

**Algoritam K-sredina** je algoritam za grupisanje koji minimizuje udaljenost između čvorova i njihovih centara. U kontekstu Minimum K-Center problema, ovaj algoritam počinje sa nasumično izabranim centrima i zatim ponavlja proces grupisanja čvorova i ažuriranja centara. Algoritam se izvodi u nekoliko iteracija dok centri ne postanu stabilni.

Algoritam funkcioniše tako što:

1. Nasumično bira k početnih centara među čvorovima grafa.
2. Dodeljuje svaki čvor najbližem centru na osnovu udaljenosti između čvorova.
3. Izračunava nove centre za svaki klaster kao čvorove najbliže težištu svojih klastera.
4. Ponavlja korake dok se centri ne stabilizuju.

Iako je algoritam jednostavan i brz, može se desiti da pronađeni centri nisu globalno optimalni. U standardnoj implementaciji, centri su uvek čvorovi grafa, što osigurava njihovu interpretabilnost u okviru problema.

**Prednosti:**

- Brz i jednostavan za implementaciju.
- Dobar za grafove sa relativno ravnomerno raspoređenim čvorovima.

**Nedostaci:**

- Može se zaglaviti u lokalnim minimumima.
- Osetljivost na inicijalni izbor centara.
- U istim grafovima može imati veoma različita rešenja, u različitim pokretanjima algoritma
- Ne funkcioniše dobro sa nepravilno raspoređenim podacima.

Algoritam K-sredina je koristan za brzu aproksimaciju, ali se često koristi kao početni korak u složenijim metodama optimizacije.

## 2.5 Algoritam Simuliranor Kaljenja

**Simulirano kaljenje** (Simulated Annealing) je stohastički algoritam koji koristi nasumične promene u konfiguraciji kako bi istražio prostor rešenja. Ovaj algoritam ima mehanizam za balansiranje između istraživanja novih rešenja i iskorišćavanja postojećih, uz pomoć parametra temperature koji se postepeno smanjuje tokom vremena. Kada je temperatura visoka, algoritam dozvoljava veće promene, dok sa smanjenjem temperature algoritam postaje precizniji i fokusira se na lokalna poboljšanja. Simulirano kaljenje može pronaći globalno optimalno rešenje, ali je njegov uspeh zavistan od parametara temperature i strategije pretrage.

**Prednosti:**

- Može da pronađe globalno optimalno rešenje.
- Fleksibilan u pretrazi prostora rešenja.

**Nedostaci:**

- Potrebno je pažljivo podešavanje parametara.
- Može biti sporo.

## 2.6 Ant Colony Optimization Algoritam

**Algoritam optimizacije mravljom kolonijom (ACO)** je inspirisan ponašanjem mrava koji traže najkraće puteve do izvora hrane. Algoritam koristi kolektivnu inteligenciju kolonije mrava koja komunicira putem feromona na putu. Mravi koriste feromone da označe puteve koji vode do dobrih rešenja, i tokom vremena, algoritam koristi informacije o feromonima kako bi navigirao kroz prostor rešenja. ACO je vrlo efikasan za rešavanje složenih optimizacijskih problema, posebno onih sa velikim brojem varijabli. U kontekstu Minimum K-Center problema, ACO se koristi za pronalaženje optimalnih k-centara na osnovu ponašanja kolonije mrava.

#### **Prednosti:**

- Efikasan za složene probleme.
- Koristi kolektivnu inteligenciju za pronalaženje optimalnih rešenja.

#### **Nedostaci:**

- Potrebno je pažljivo podešavanje parametara.
- Računarski intenzivan za vrlo velike grafove.

### **3. Rezultati**

#### **Napomena: Parametri za testiranje algoritama**

Testiranje svih sedam algoritama implementiranih u ovom radu izvršeno je na grafovima sa različitim brojem čvorova, brojem grafova, vrednostima za k (broj centara) i opsegom pozicija čvorova. Parametri korišćeni za generisanje testnih grafova su sledeći:

- **Prvi graf:**
  - Broj čvorova: 10
  - Broj centara: 4
  - Opseg pozicija: (0, 10)
- **Drugi graf:**
  - Broj čvorova: 15
  - Broj centara: 6
  - Opseg pozicija: (0, 20)
- **Treći graf:**
  - Broj čvorova: 20
  - Broj centara: 8
  - Opseg pozicija: (0, 30)

Svaki primer je generisan 100 puta, odnosno svaki algoritam je izvršen 100 puta na 100 nezavisnih grafova sa slučajnim pozicijama unutar datih opsega pozicija.

Sledeći primeri prikazuju konkretne pozicije čvorova za svaki skup:

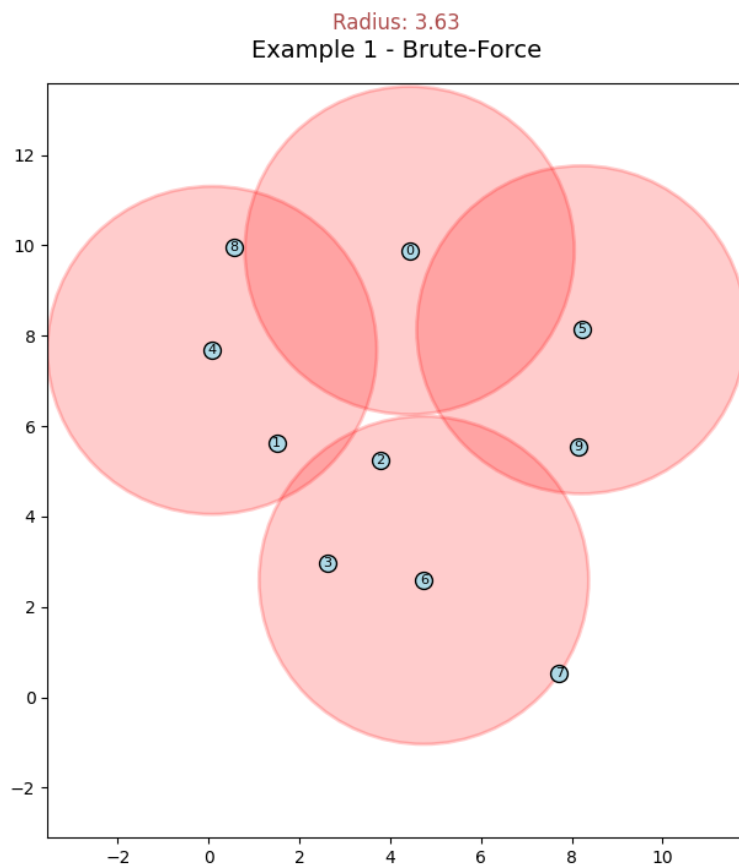
- **Prvi skup pozicija:**
  - [(4.428, 9.886), (1.499, 5.612), (3.778, 5.241), (2.626, 2.982), (0.078, 7.678), (8.214, 8.132), (4.742, 2.592), (7.724, 0.524), (0.561, 9.949), (8.132, 5.529)]
- **Drugi skup pozicija:**

- [(9.877, 15.019), (8.461, 0.267), (7.928, 5.218), (1.693, 17.310), (18.529, 4.300), (4.056, 2.441), (7.274, 14.513), (12.056, 4.176), (6.776, 3.256), (7.023, 11.880)]
- **Treći skup pozicija:**
  - [(24.613, 4.480), (10.056, 27.268), (17.605, 27.556), (19.818, 4.807), (11.632, 16.746), (19.048, 29.152), (8.293, 13.965), (20.602, 19.268), (22.415, 3.592), (21.563, 23.743)]

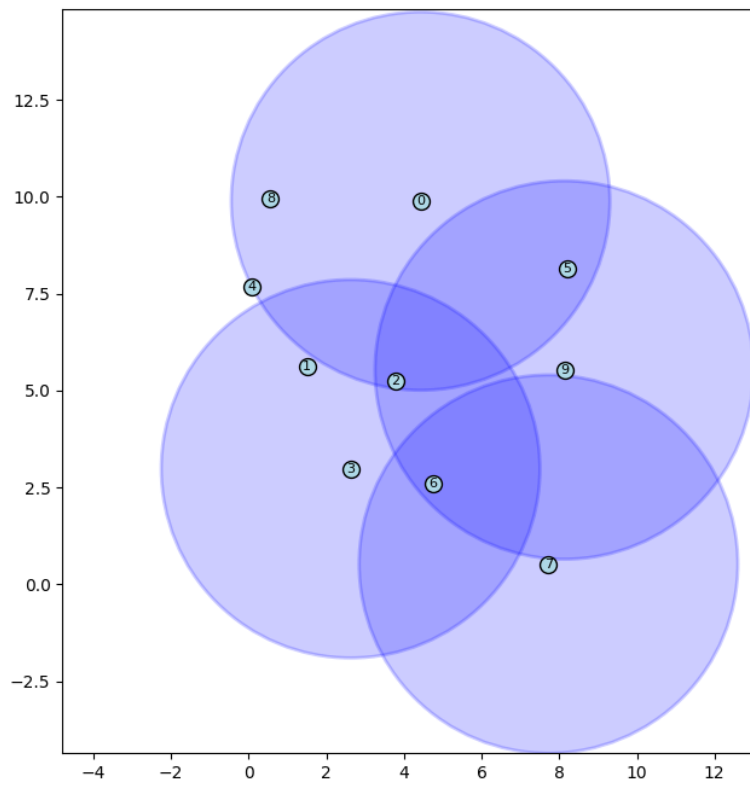
Ovi parametri predstavljaju primere sa kojima ćemo raditi radi testiranja efikasnosti i tačnosti algoritama pri rešavanju problema Minimum K-Center. Od slučajno generisanih primera izabrana su 3 tako da se na njima može jasno videti razlika između svih algoritama.

Testirali smo sve algoritme na različitim grafovima sa različitim parametrima. Algoritmi su ocenjeni na osnovu vremena izvršenja i preciznosti (minimalnog radijusa).

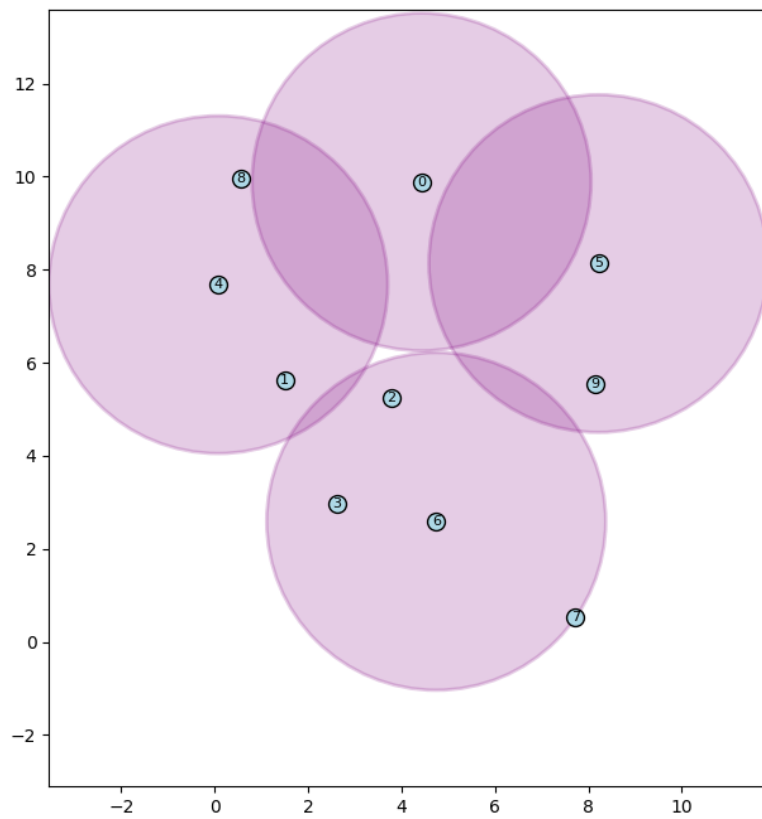
### 3.1 Testiranje prvog grafa



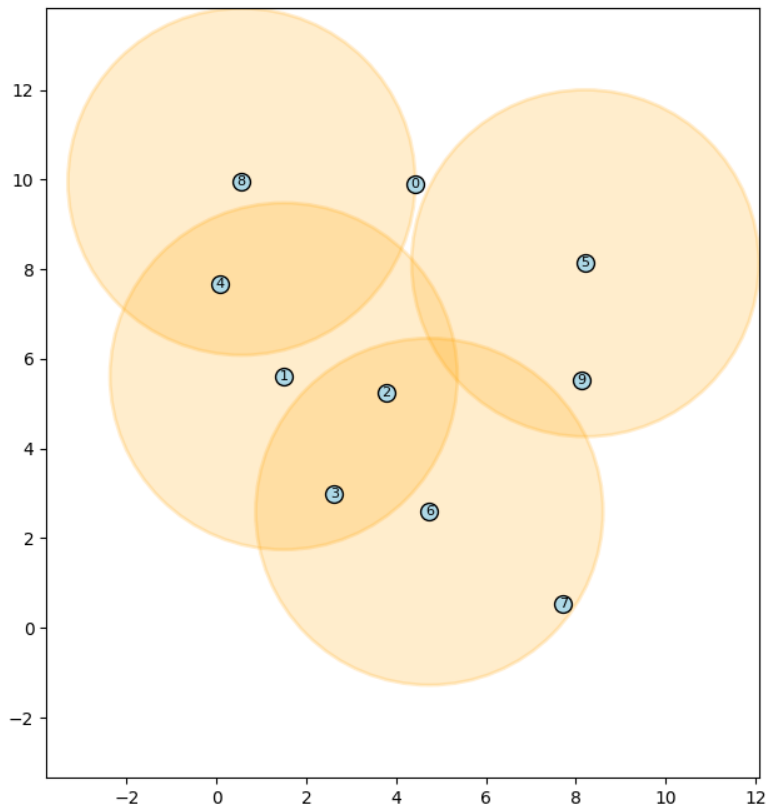
Radius: 4.88  
Example 1 - Greedy



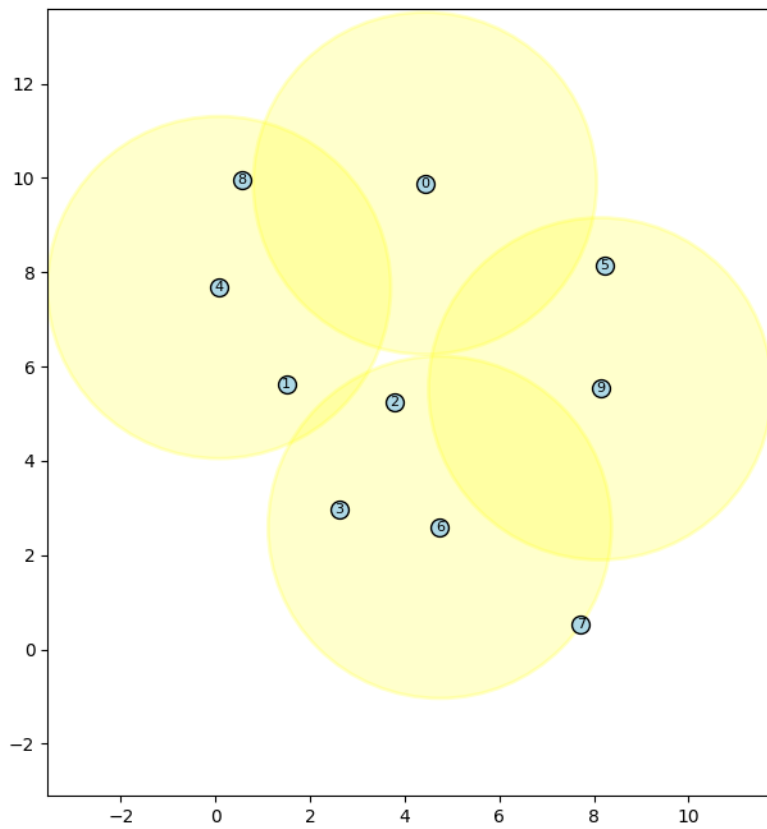
Radius: 3.63  
Example 1 - Genetic

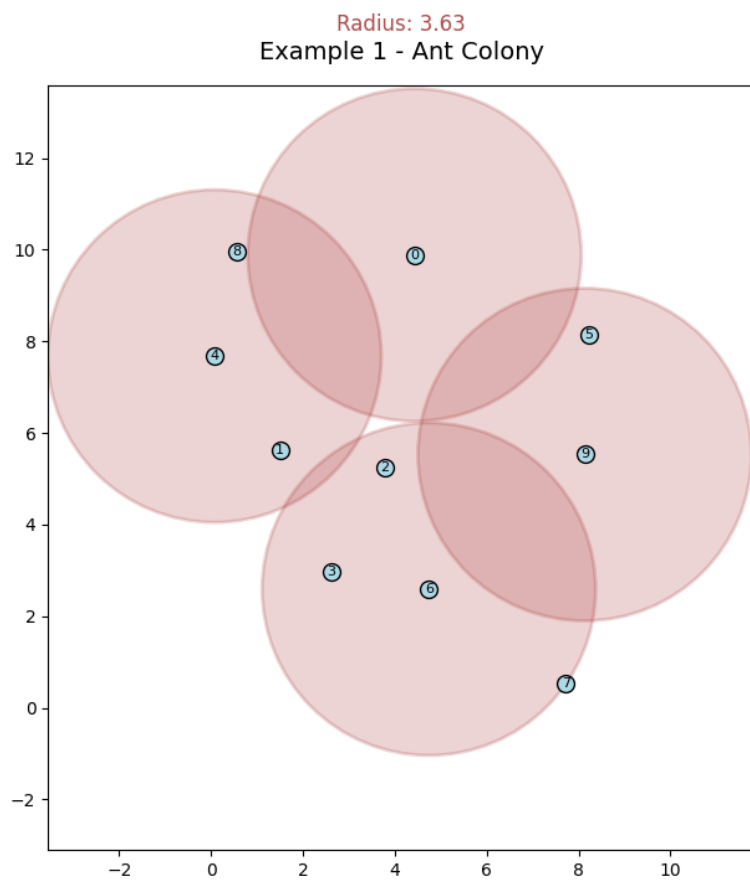


Radius: 3.87  
Example 1 - K-Means Heuristic

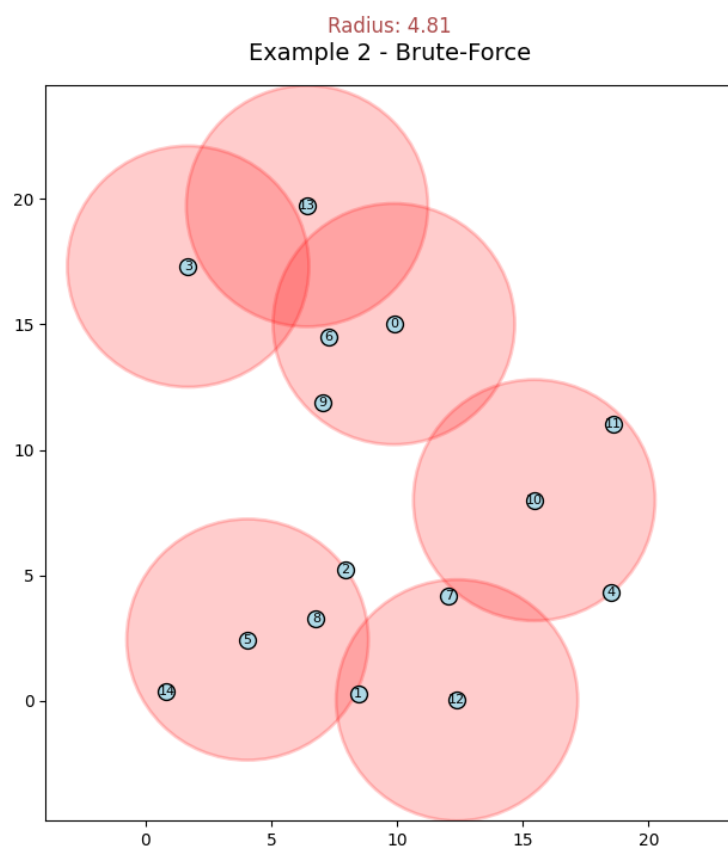


Radius: 3.63  
Example 1 - Simulated Annealing



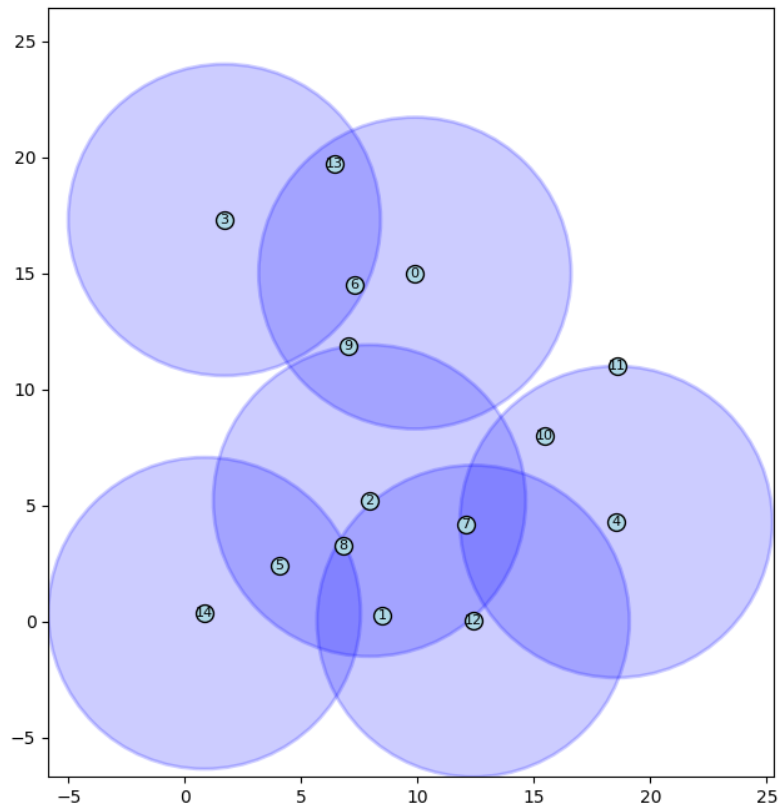


### 3.2 Testiranje drugog grafa

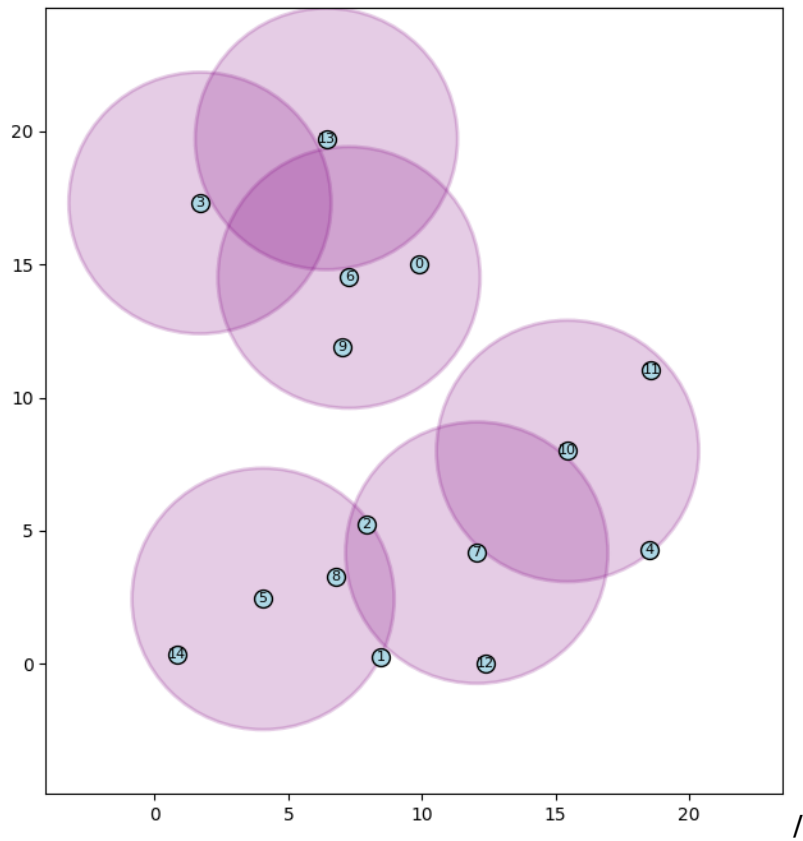




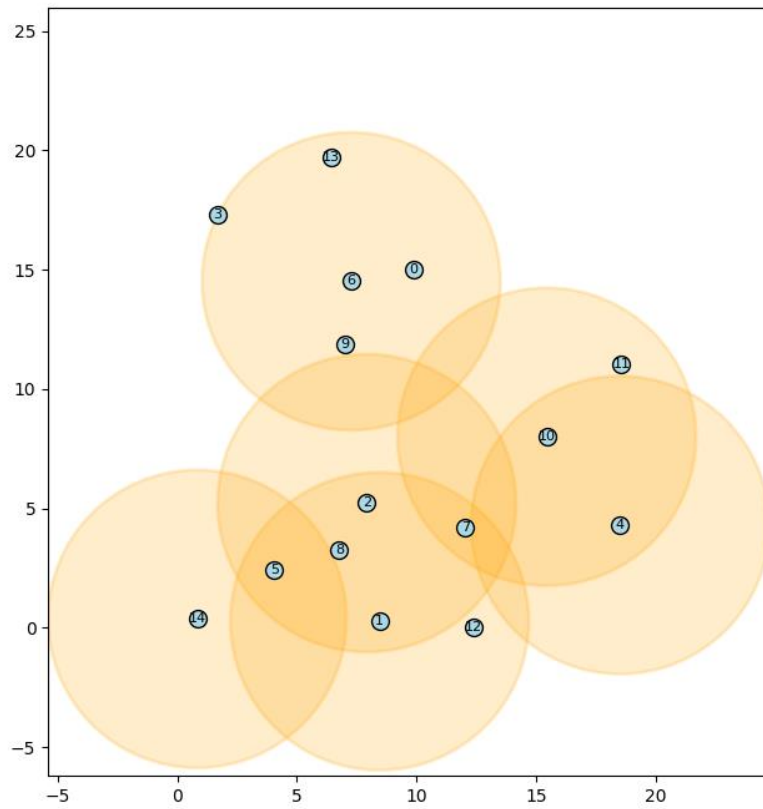
Radius: 6.72  
Example 2 - Greedy



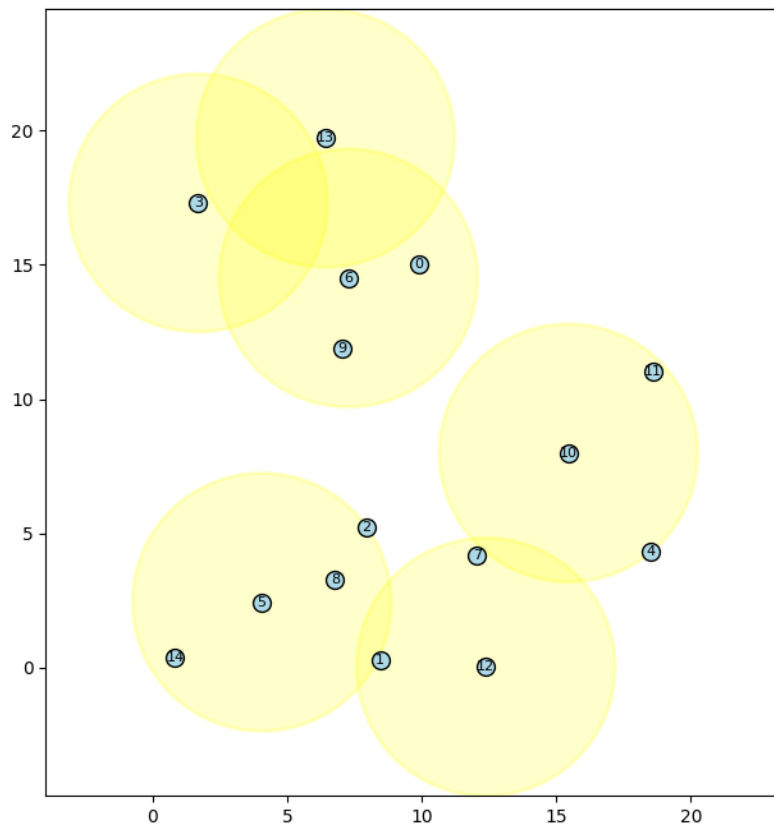
Radius: 4.91  
Example 2 - Genetic



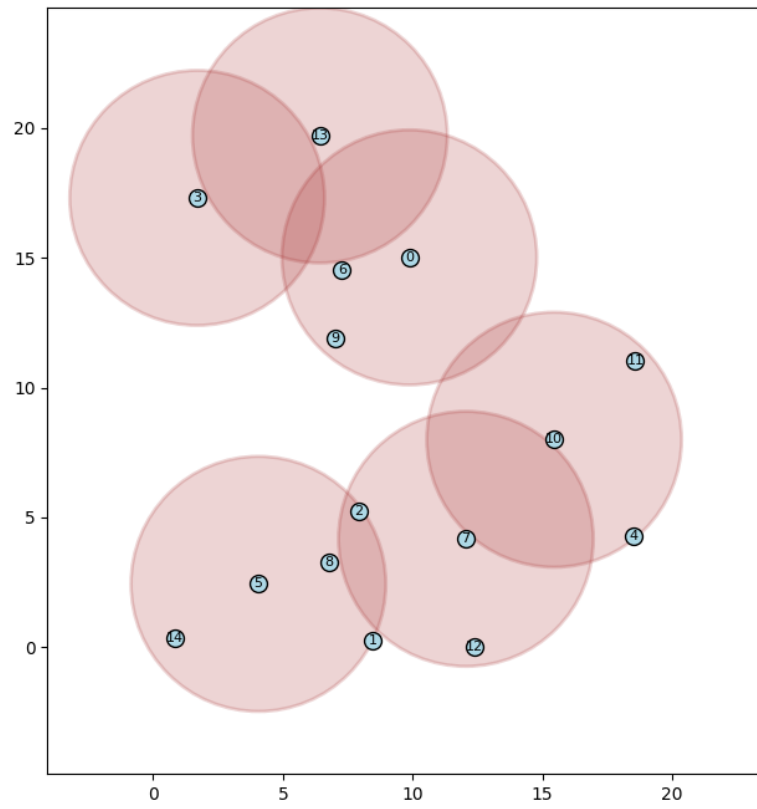
Radius: 6.24  
Example 2 - K-Means Heuristic



Radius: 4.81  
Example 2 - Simulated Annealing

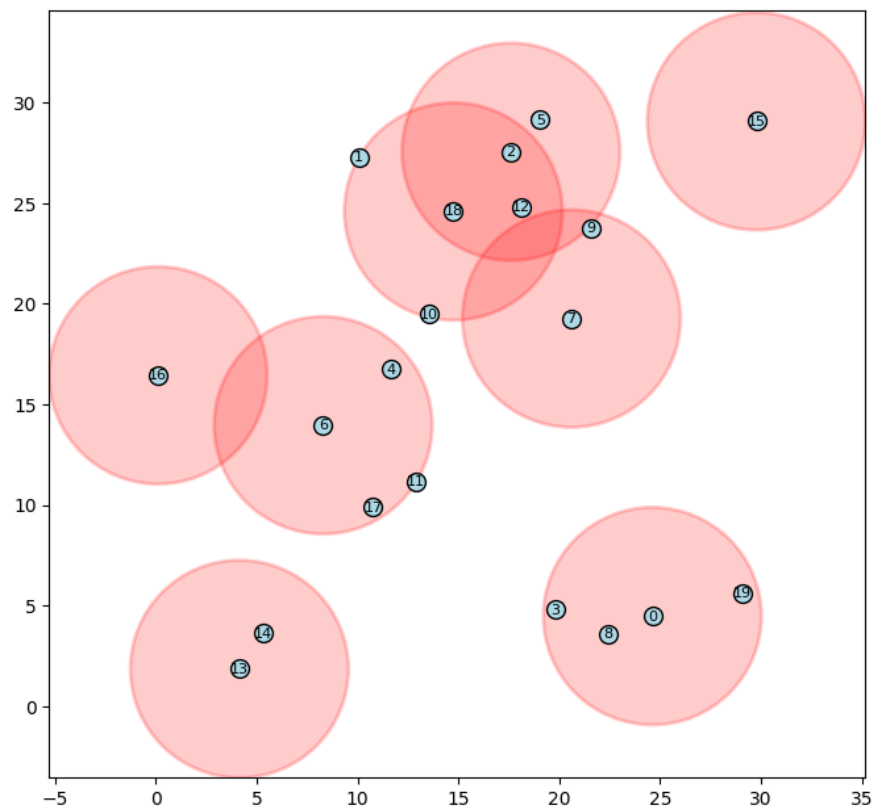


Radius: 4.91  
Example 2 - Ant Colony

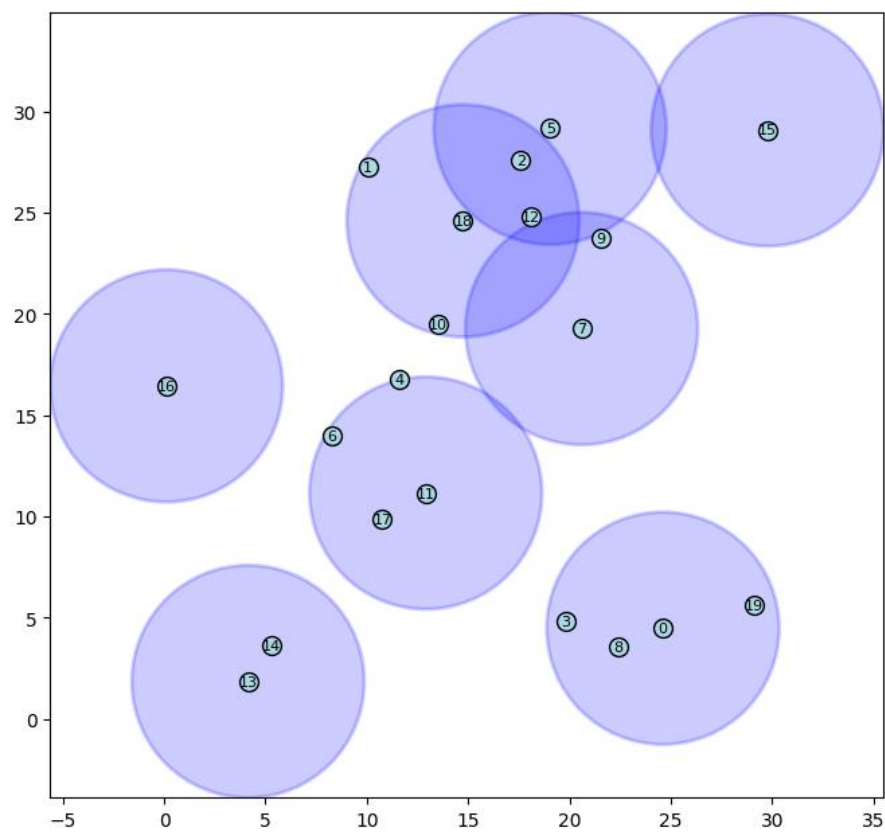


### 3.3 Testiranje trecega grafa

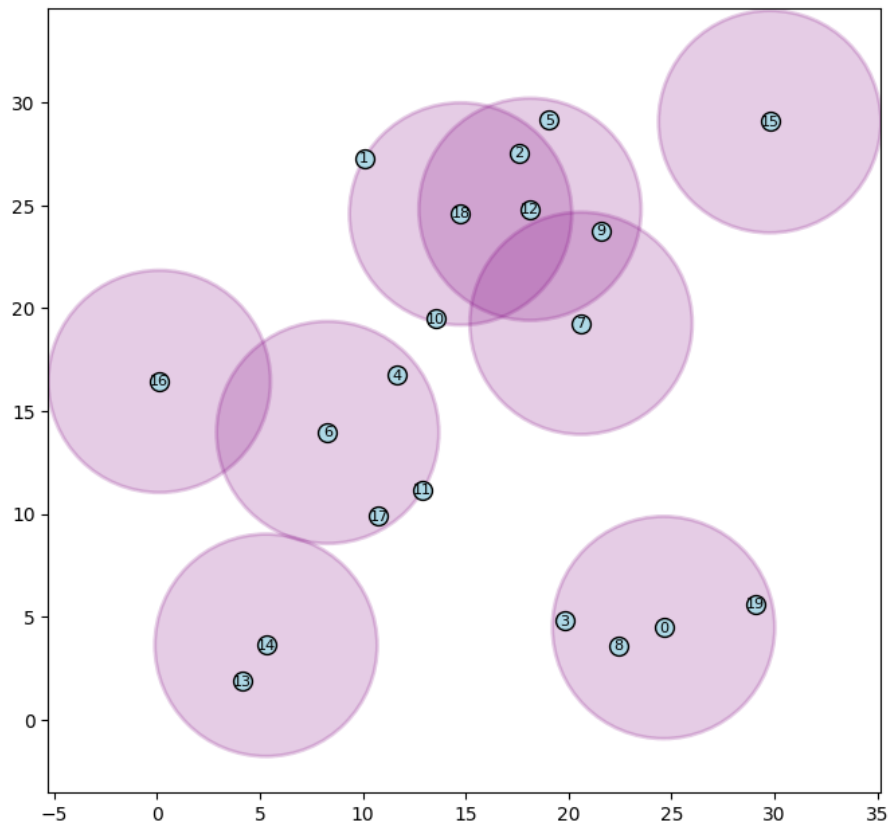
Radius: 5.40  
Example 3 - Brute-Force



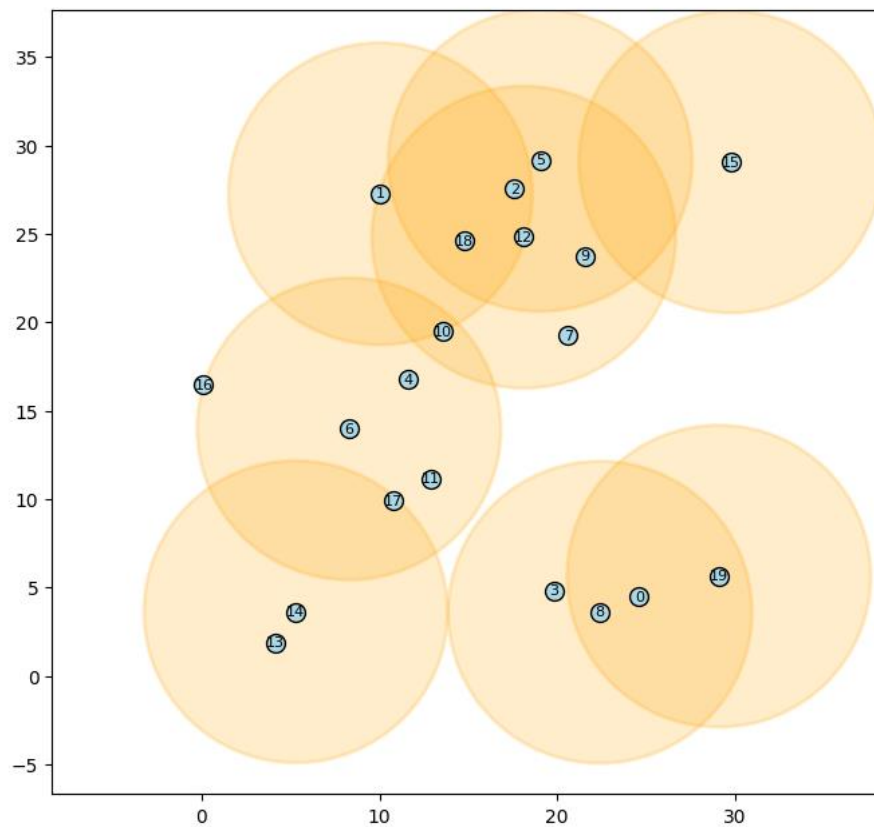
Radius: 5.74  
Example 3 - Greedy



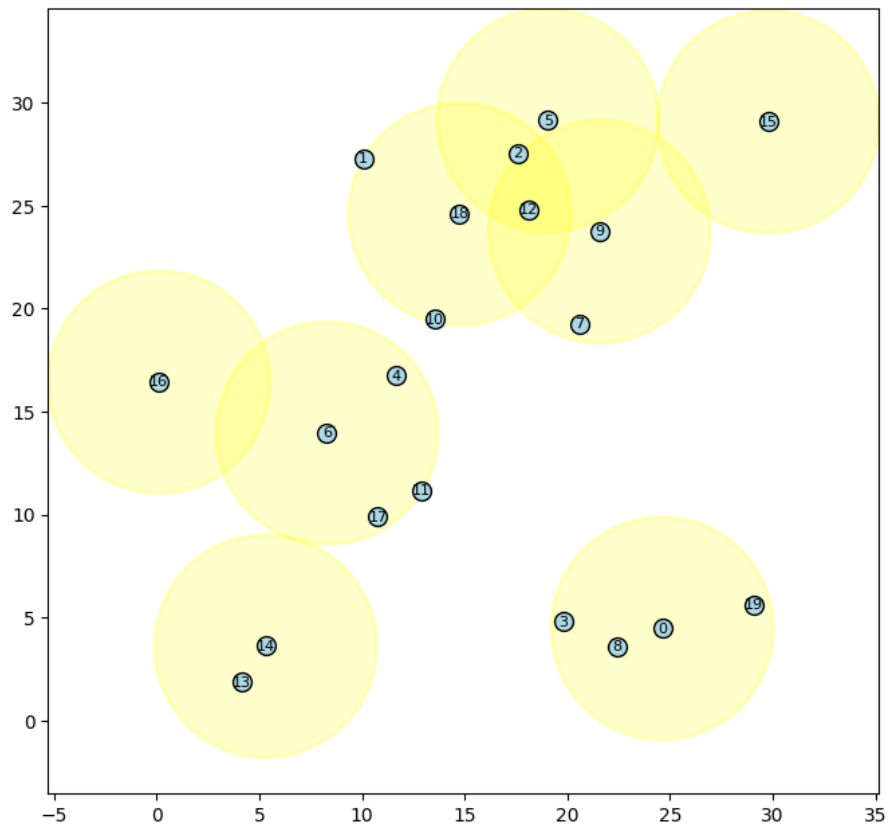
Radius: 5.40  
Example 3 - Genetic



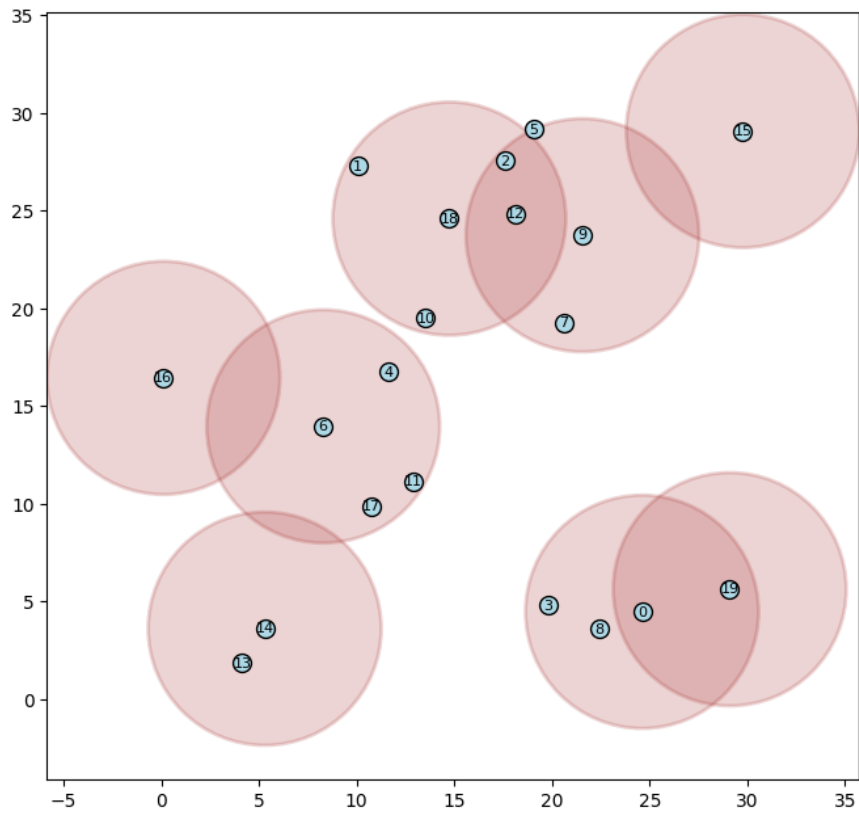
Radius: 8.55  
Example 3 - K-Means Heuristic



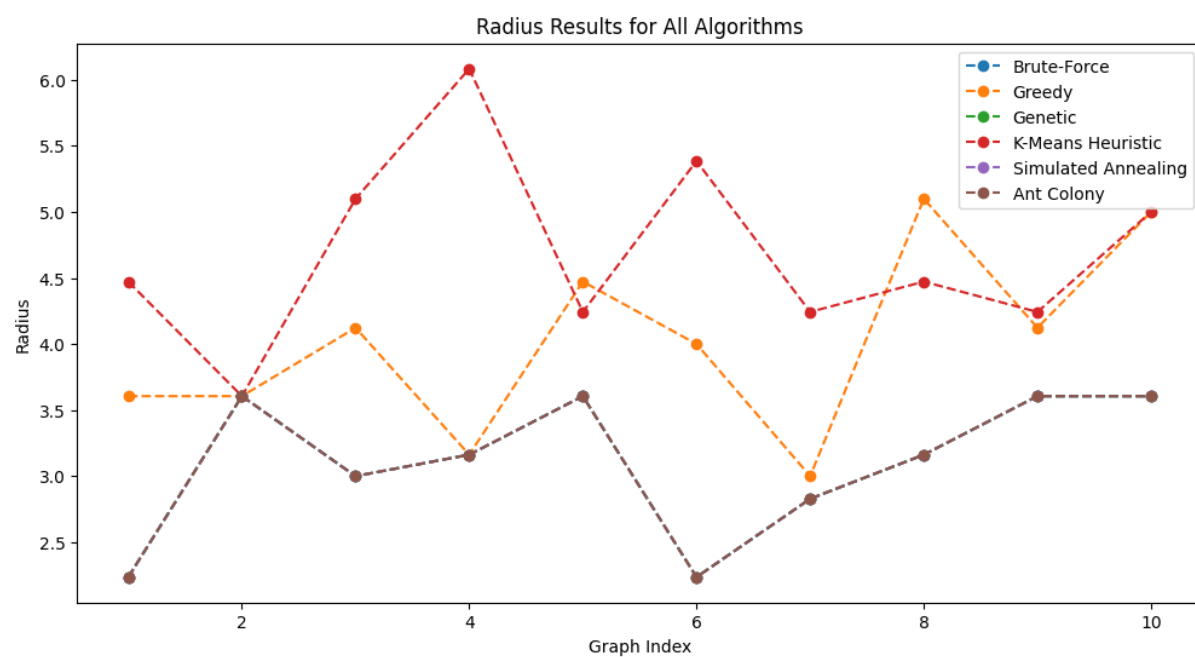
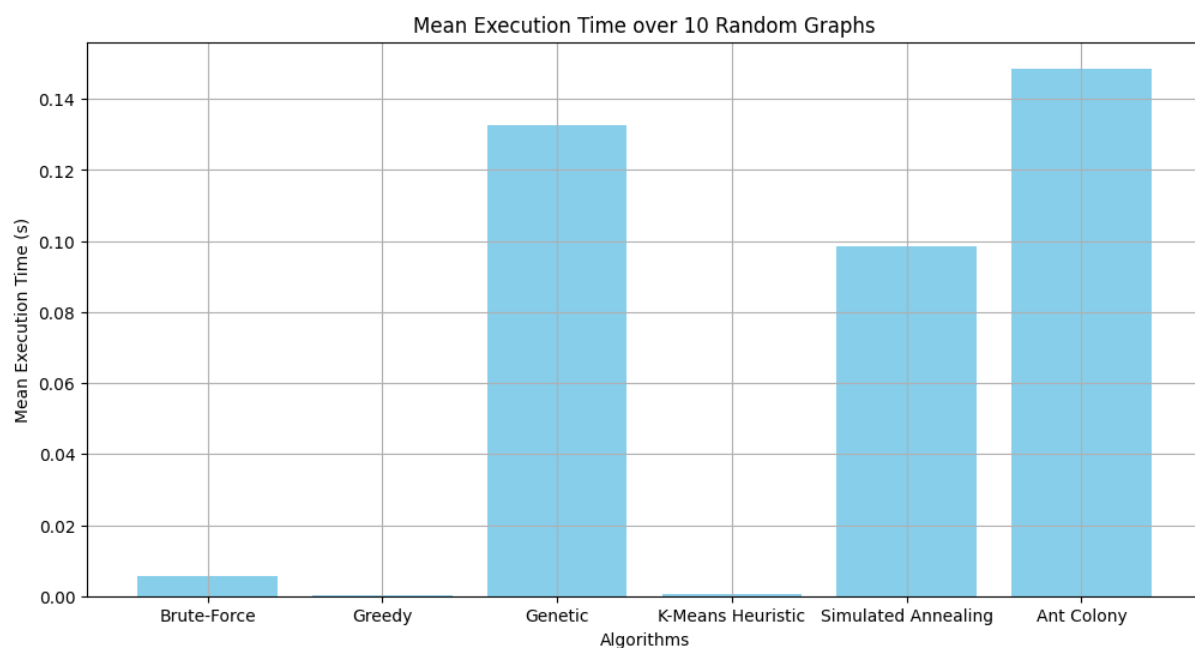
Radius: 5.40  
Example 3 - Simulated Annealing

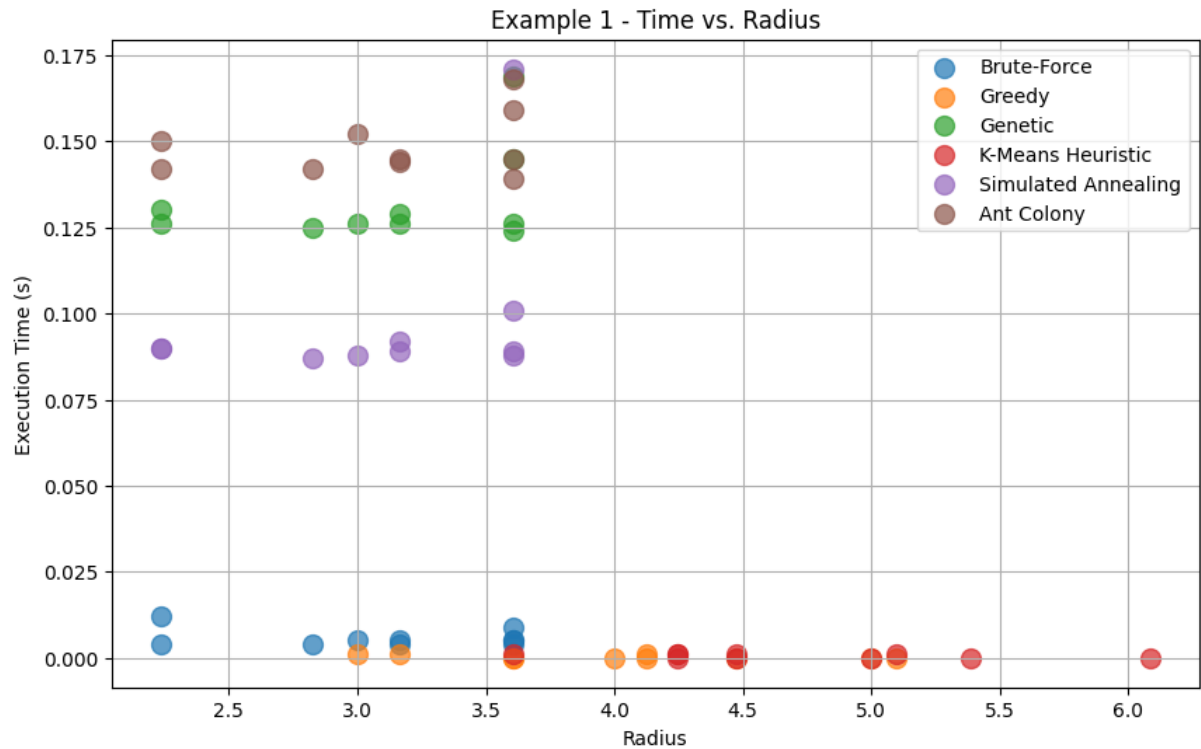


Radius: 5.97  
Example 3 - Ant Colony



### 3.4 Testiranje 10 različitih grafova prvog tipa sa svim algoritmima





Summary Table for Example 1

Algorithm	Min Radius	Max Radius	Mean Radius	Min Time	Max Time	Mean Time
<b>Brute-Force</b>	<b>2.2361</b>	<b>3.6056</b>	<b>3.1047</b>	<b>0.004</b>	<b>0.012</b>	<b>0.0057</b>
Greedy	3.0	5.099	4.0191	0.0	0.001	0.0003
Genetic	2.2361	3.6056	3.1047	0.124	0.169	0.1326
K-Means Heuristic	3.6056	6.0828	4.6845	0.0	0.001	0.0005
Simulated Annealing	2.2361	3.6056	3.1047	0.087	0.171	0.0985
Ant Colony	2.2361	3.6056	3.1047	0.139	0.168	0.1486

Example 1 Algorithm Performance Summary:

**Brute-Force:** Min Radius = 2.2361, Mean Radius = 3.1047, Mean Execution Time = 0.0057s

**Greedy:** Min Radius = 3.0000, Mean Radius = 4.0191, Mean Execution Time = 0.0003s

**Genetic:** Min Radius = 2.2361, Mean Radius = 3.1047, Mean Execution Time = 0.1326s

**K-Means Heuristic:** Min Radius = 3.6056, Mean Radius = 4.6845, Mean Execution Time = 0.0005s

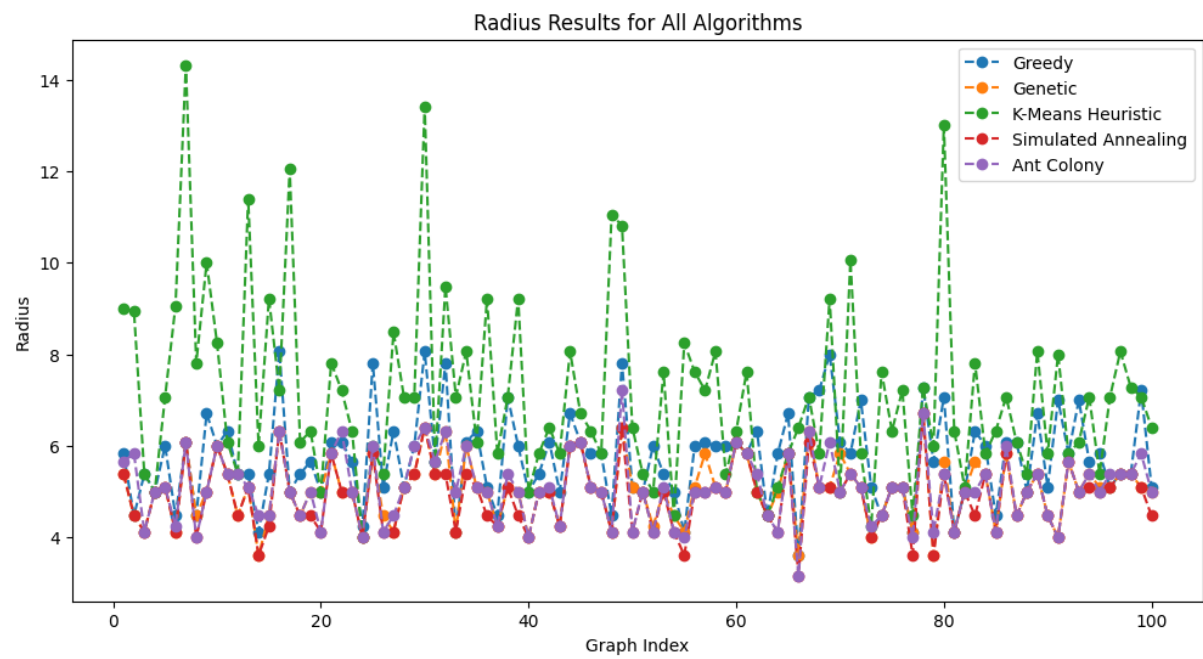
**Simulated Annealing:** Min Radius = 2.2361, Mean Radius = 3.1047, Mean Execution Time = 0.0985s

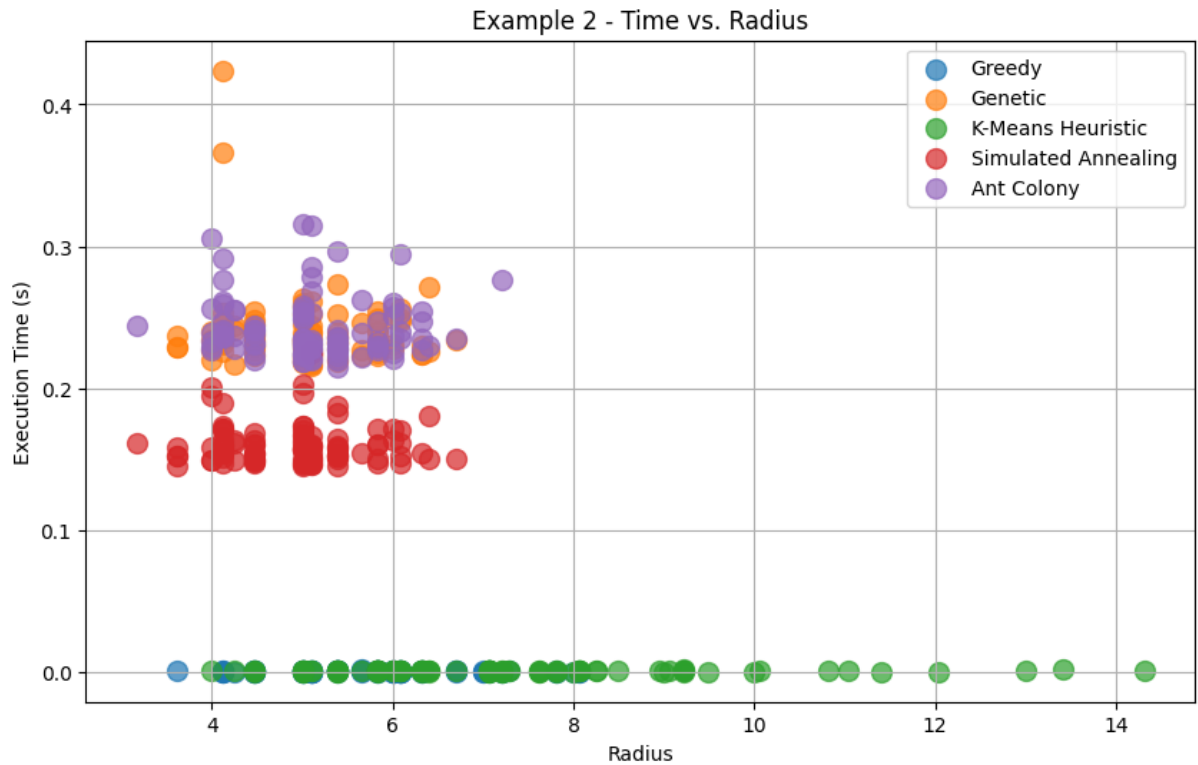
**Ant Colony:** Min Radius = 2.2361, Mean Radius = 3.1047, Mean Execution Time = 0.1486s

Best Algorithm for Example 1 (Weighted by Time and Radius): **Brute-Force**



### 3.5 Testiranje 100 različitih grafova drugog tipa sa svim algoritmima





Summary Table for Example 2

Algorithm	Min Radius	Max Radius	Mean Radius	Min Time	Max Time	Mean Time
Brute-Force	-	-	-	-	-	-
Greedy	3.6056	8.0623	5.7447	0.0	0.002	0.0006
Genetic	3.6056	6.7082	5.039	0.216	0.424	0.2385
K-Means Heuristic	4.0	14.3178	7.0928	0.0	0.002	0.0008
<b>Simulated Annealing</b>	<b>3.1623</b>	<b>6.7082</b>	<b>4.9067</b>	<b>0.145</b>	<b>0.226</b>	<b>0.1606</b>
Ant Colony	3.1623	7.2111	5.0815	0.215	0.316	0.2426

Example 2 Algorithm Performance Summary:

**Greedy:** Min Radius = 3.6056, Mean Radius = 5.7447, Mean Execution Time = 0.0006s

**Genetic:** Min Radius = 3.6056, Mean Radius = 5.0390, Mean Execution Time = 0.2385s

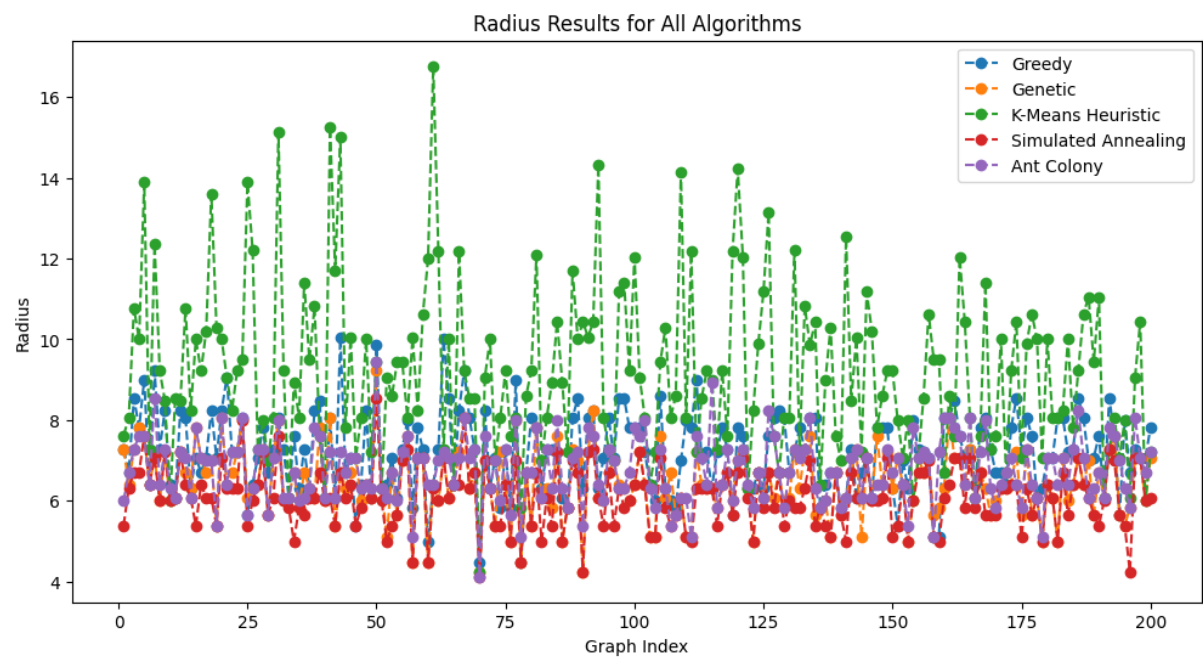
**K-Means Heuristic:** Min Radius = 4.0000, Mean Radius = 7.0928, Mean Execution Time = 0.0008s

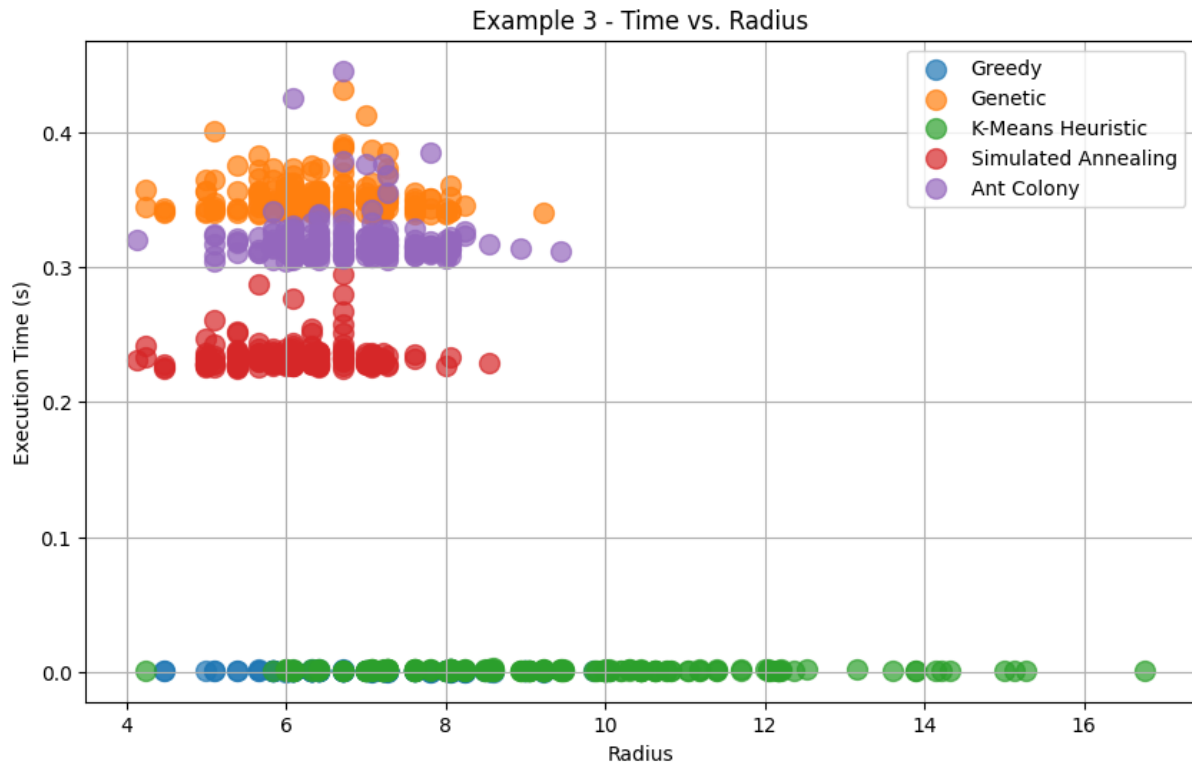
**Simulated Annealing:** Min Radius = 3.1623, Mean Radius = 4.9067, Mean Execution Time = 0.1606s

**Ant Colony:** Min Radius = 3.1623, Mean Radius = 5.0815, Mean Execution Time = 0.2426s

Best Algorithm for Example 2 (Weighted by Time and Radius): **Simulated Annealing**

### 3.6 Testiranje 200 različnih grafova trećeg tipa sa svim algoritmima





Summary Table for Example 3

Algorithm	Min Radius	Max Radius	Mean Radius	Min Time	Max Time	Mean Time
Brute-Force	-	-	-	-	-	-
Greedy	4.4721	10.0499	7.2131	0.0	0.002	0.001
Genetic	4.2426	9.2195	6.4613	0.339	0.432	0.3522
K-Means Heuristic	4.2426	16.7631	9.1796	0.001	0.003	0.0013
<b>Simulated Annealing</b>	<b>4.1231</b>	<b>8.544</b>	<b>6.1254</b>	<b>0.225</b>	<b>0.295</b>	<b>0.2341</b>
Ant Colony	4.1231	9.434	6.7787	0.304	0.446	0.3196

Example 3 Algorithm Performance Summary:

**Greedy:** Min Radius = 4.4721, Mean Radius = 7.2131, Mean Execution Time = 0.0010s

**Genetic:** Min Radius = 4.2426, Mean Radius = 6.4613, Mean Execution Time = 0.3522s

**K-Means Heuristic:** Min Radius = 4.2426, Mean Radius = 9.1796, Mean Execution Time = 0.0013s

**Simulated Annealing:** Min Radius = 4.1231, Mean Radius = 6.1254, Mean Execution Time = 0.2341s

**Ant Colony:** Min Radius = 4.1231, Mean Radius = 6.7787, Mean Execution Time = 0.3196s

Best Algorithm for Example 3 (Weighted by Time and Radius): **Simulated Annealing**

## 4. Zaključak

U ovom radu sam testirao sve algoritme na 3 vrste grafova, od manjih do većih i rezultati su sledeći:

- Brute-force algoritam koji prolazi kroz sva rešenja i među njima traži najbolje nalazi najmanji radijus u najkraćem roku nad malim grafovima (u mom slučaju nad slučajnim grafovima sa 10 čvorova i 4 centra). Ali kada su u pitanju veći grafovi, algoritam će uvek naći najbolje rešenje ali po cenu mnogo većeg vremena od ostalih algoritama, što ga čini neefikasnim.
- Algoritmi Greedy i K-means heuristic su ubedljivo najbrži algoritmi, ali iako sa malim grafovima daju jako dobre rezultate, kada se broj čvorova i centara samo malo poveća nisu u stanju da nađu minimalni radijus ni blizu kao ostali algoritmi.
- Genetski i Ant colony algoritam su jako efikasni algoritmi kako za male tako i za ogromne grafove radeći za veoma kratko vreme. Genetski algoritam je malo bolji i nekad približno dostiže brute-force algoritam po efikasnom nalaženju minimalnog radijusa, a s obzirom na malo vreme izvršavanja, mnogo je korisniji.
- Definitivno najbolji i najefikasniji algoritam je simulated annealing algoritam, odnosno algoritam simuliranog kaljenja. Sposoban i za male i za velike grafove, brži i efikasniji od genetskog i ant colonz algoritama, algoritam simuliranog kaljenja je najbolji algoritam za pronalaženje minimalnog broja k-centra.

## 5. Literatura

- **Lima, A., Rodrigues, B., Wang, F., & Xu, Z.** (2004). k-Center problems with minimum coverage. *Computers & Operations Research*  
[https://www.researchgate.net/publication/220149263\\_k-Center\\_problems\\_with\\_minimum\\_coverage](https://www.researchgate.net/publication/220149263_k-Center_problems_with_minimum_coverage)
- **Zhang, X., & Wang, Z.** (2017). *Solving the p-Center Problem Using Simulated Annealing Algorithm*. *Computers & Operations Research*,  
[https://www.researchgate.net/publication/220857373\\_A\\_Simulated\\_Annealing\\_Algorithm\\_for\\_the\\_Circles\\_Packing\\_Problem](https://www.researchgate.net/publication/220857373_A_Simulated_Annealing_Algorithm_for_the_Circles_Packing_Problem)
- **Cornell University.** (2009). *CS 1114: Introduction to Computer Science: Lecture 23 - Graph Algorithms* [PDF]. Cornell University.  
<https://www.cs.cornell.edu/courses/cs1114/2009sp/lectures/CS1114-lec23.pdf>
- **Dorigo, M., & Stützle, T.** (2004). *Ant Colony Optimization* [PDF].  
<https://people.idsia.ch/~luca/aco2004.pdf>