Multidiszciplináris tudományok, 11. kötet. (2021) 4 sz. pp. 127-132 **https://doi.org/10.35925/j.multi.2021.4.15**

Konstrukciós körút alapú algoritmusok használata klaszterezésre

Agárdi Anita
tanársegéd, Miskolci Egyetem, Informatika Intézet, Általános Informatikai Intézeti Tanszék
3515 Miskolc, Miskolc-Egyetemváros, e-mail: agardianita@iit.uni-miskolc.hu

Absztrakt

Jelen cikk a konstrukciós algoritmusok használatát klaszterezési problémára mutatja be. A konstrukciós körút készítő algoritmusok közül a legközelebbi szomszéd algoritmust, a beszúró heurisztikákat (legközelebbi pont beszúrása, legtávolabbi pont beszúrása, legolcsóbb beszúrás, véletlen pont beszúrása), és a greedy algoritmus lett tesztelve. A módosított algoritmusok lényege, hogy nem kell megadni a klaszterszámot, azt maga az algoritmus alakítja ki.

Kulcsszavak: *klaszterezés, konstrukciós algoritmusok, legközelebbi szomszéd, beszúró heurisztikák, greedy*

Abstract

This paper presents the use of construction circuit-based algorithms in clustering. Among the construction tour algorithms, the nearest neighbor algorithm, the insertion heuristics (nearest insertion, farthest insertion, cheapest insertion, arbitrary insertion), and the greedy algorithm were tested. The essence of the modified algorithms is that it is not necessary to specify the cluster number, it is formed by the algorithm itself.

Keywords: *clustering, construction algorithm, nearest neighbour, insertion heuristics, greedy*

1. Bevezetés

A klaszterezés során elemeket csoportosítjuk. Egy jó klaszterező algoritmus eredménye olyan klaszterezés, mely során az egymáshoz közel álló, egymáshoz hasonlító elemek egy klaszterbe kerülnek. Azok az elemek, amelyek pedig egymástól különböznek külön klaszterbe kerülnek. Egy adatsort sokféleképpen klaszterezhetünk, számtalan klaszterező algoritmus látott már napvilágot. A partíciós módszerek [1] egy-egy kezdeti klaszterezést iteratívan javítanak. A hierarchikus klaszterezések [1] egy hierarchikus adatszerkezetet (fát) alakítanak ki. A sűrűség alapú klaszterezés [1] során igaz az, hogy az egyes elemektől egy bizonyos távolságra találhatóak elemek (adott mennyiségű elemek). A távol eső pontok nem kerülnek klaszterekbe, így zajos adatok kiszűrésére is alkalmasak.
A klaszterezések egyik nagy hátránya, hogy a felhasználónak meg kell adni a klaszterszámot. A partíciós módszerek és a hierarchikus módszerek többségének ez a bemenete. Amennyiben nem a klaszterszámot kell megadni, egyéb paramétereket, például a DBSCAN [1] sűrűség alapú klaszterezésnél a sugarat és a pontok számát (amennyi pontnak minimum egy elem sugarában el kell helyezkednie). A felhasználó gyakran nem tudja megadni a klaszterszámot, és nem érti az egyéb, klaszterezés során használt paramétereket.

A klaszterezés irodalmának feltérképezése során számos algoritmust találunk, melyek nem klasszikus klaszterező algoritmusok, ilyenek például a metaheurisztikák (genetikus algoritmus [2] szimulált lehűtés [3], hangya kolónia optimalizáció [4], particle swarm optimization [5], tabu keresés [6], hegymászó algoritmus [7], flower pollination [8]).

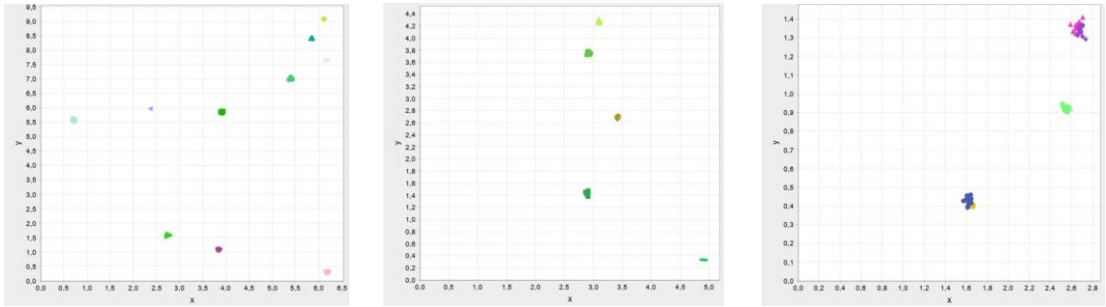
2. Konstrukciós algoritmusok

Jelen cikkben olyan algoritmusok kerülnek bemutatásra, amelyek nem klasszikus klaszterező algoritmusok. Ezen algoritmusok körút alapú algoritmusok, a klasszikus algoritmusok egy-egy körutat konstruálnak úgy, hogy lokálisan a legjobb lépéseket teszik meg. A legközelebbi szomszéd algoritmus [9] mindig az utoljára választott ponthoz legközelebbi, még ki nem választott pontot választja ki. A beszúró heurisztikák [10] egy-egy körutat úgy képeznek, hogy a részkörútba folyamatosan szúrnak be egy-egy elemet. Arra a helyre szúrják be, ahol a beszúrás költsége minimális. Azt, hogy melyik pontot választják ki éppen beszúrandónak, arra az alábbi stratégiák léteznek: legközelebbi pont, legtávolabbi pont, legolcsóbb pont, véletlen pont. A legközelebbi pont beszúrása algoritmus [10] azt a pontot szúrja be a körútba, amely a körútbeli elemektől legközelebb található. A legtávolabbi pont beszúrása algoritmus [10] azt a pontot szúrja be a körútba, amely a körútbeli elemektől legtávolabb található. A legolcsóbb beszúrás algoritmus [10] azt a pontot szúrja be, amelyik beszúrása a körutat a legkisebb mértékben növeli. A véletlen pont beszúrása algoritmus [10] a beszúrandó pontot véletlenszerűen választja ki. A greedy [11] algoritmus a gráf éleit sorba rendezi. Folyamatosan adja hozzá a körúthoz a gráf egyes éleit és csomópontjait úgy, hogy addig amíg minden csomópontot ki nem választottunk ne alakuljon ki körút a gráfban. Az algoritmusok úgy lettek kialakítva, hogy a felhasználótól a klaszterezendő adatsoron kívül semmilyen egyéb bemeneti paramétert nem vár az algoritmus. A cikk következő fejezetében az algoritmusok teszt eredményeit mutatom be három különböző adatsoron.

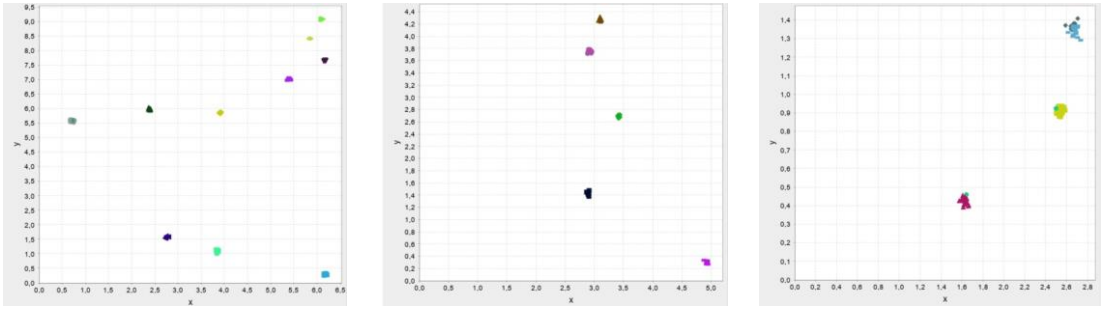
3. Futási eredmények

Ezen fejezetben az algoritmusok futási eredményeit mutatjuk be. Három különböző adatsor eredményei kerülnek bemutatásra. Az adatsorok jól klaszterezhetőek (kompaktak, egymástól elkülönültek). Az első adatsor 100 pontot, a második 50 pontot, a harmadik pedig 90 pontot tartalmaz. Az első adatsornál az optimális a 10 klaszter, a másodiknál az 5 klaszter, a harmadiknál pedig a 3 klaszter.

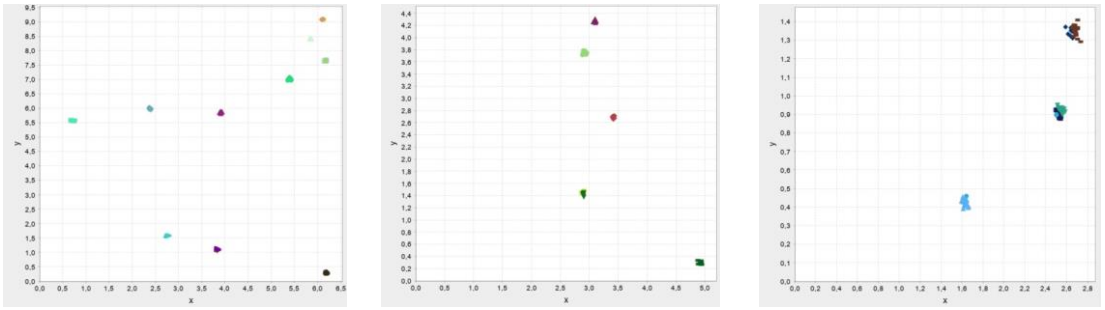
Az algoritmusok miután kialakították a körutat, a klaszterezést úgy alakítják ki, hogy azon elemek, amelyek egymáshoz közel helyezkednek el azonos klaszterbe kerülnek, míg amelyek nagyon távol azok külön klaszterbe. Az adatsor és a klaszterezés kialakítási stratégia a [12] cikkben bemutatottak alapján készült.



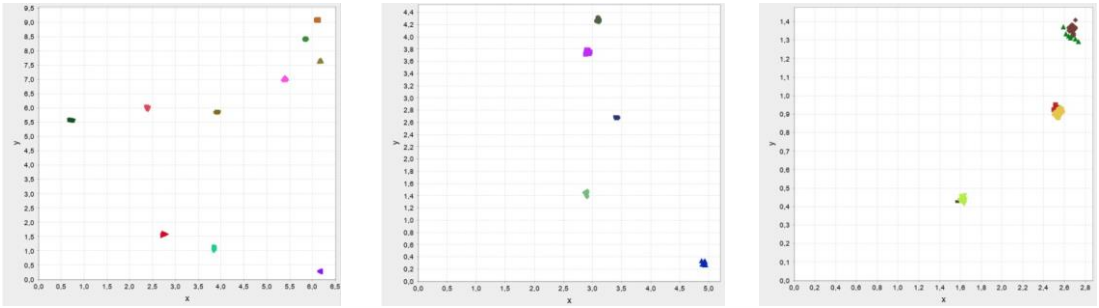
1. ábra. *Véletlen pont beszúrása algoritmus (Arbitrary Insertion) eredménye az első, második és harmadik adatsorra*



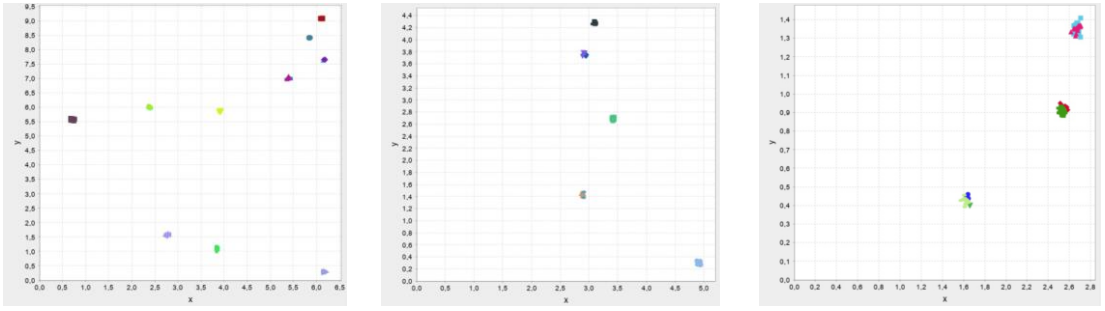
2. ábra. *Legolcsóbb pont beszúrása algoritmus (Cheapest Insertion) eredménye az első, második és harmadik adatsorra*



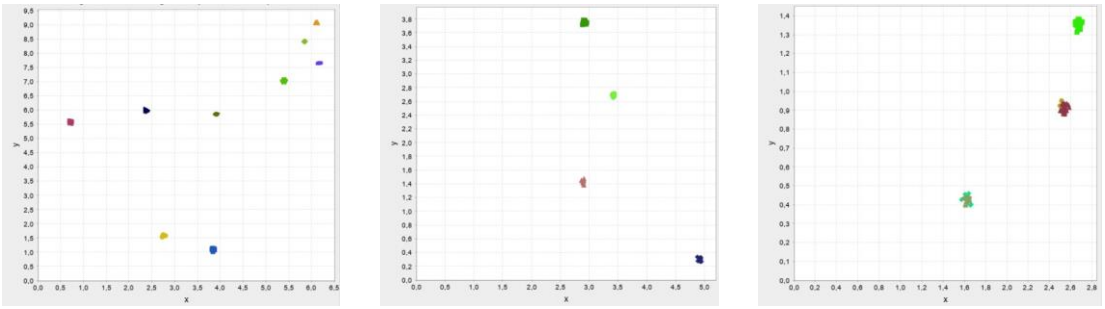
3. ábra. *Legtávolabbi pont beszúrása algoritmus (Farthest Insertion) eredménye az első, második és harmadik adatsorra*



4. ábra. *Greedy algoritmus eredménye az első, második és harmadik adatsorra*



5. ábra. *Legközelebbi pont beszúrása (Nearest Insertion) algoritmus eredménye az első, második és harmadik adatsorra*



6. ábra. *Legközelebbi szomszéd (Nearest Neighbour) algoritmus eredménye az első, második és harmadik adatsorra*

A teszt eredmények azt mutatják, hogy az első két adatsorra megtalálták az algoritmusok a klaszterhatárokat. A harmadik adatsort kicsit több részre osztották a vártnál. A következőkben bemutatom táblázatosan is az eredményeket, ahol a klaszterszámot, fitnesz értéket és a futási időt szemléltetem. A fitnesz értékek a klaszterbeli elemek egymástól vett távolságainak összegét mutatják, ennek a minimalizálása lenne a cél, tehát az, hogy minél kompaktabbak legyenek az egyes klaszterek. A túl távol elhelyezkedő elemeket külön klaszterbe, az egymáshoz nagyon közel elhelyezkedőket pedig egy klaszterbe sorolom, így sose teljesül az, hogy minden elem egy külön klaszter legyen.

1. táblázat. *Futási eredmények az első adatsorra*

Algoritmus	**Klaszterek száma**	**Fitnesz érték**	**Futási idő (perc)**
Arbitrary Insertion	10	0.3434	$5.6905*10^{-5}$
Cheapest Insertion	11	0.3327	$5.6395*10^{-5}$
Farthest Insertion	14	0.2949	$3.066*10^{-4}$
Greedy	10	0.3399	$5.2329*10^{-5}$
Nearest Insertion	13	0.3034	$3.0034*10^{-4}$
Nearest Neighbour	9	0.3531	$5.8233*10^{-6}$

Az 1. táblázat eredményei azt mutatják, hogy a Greedy és az Arbitrary Insertion az a két algoritmus, amelynek sikerült 10 csoportra bontani az adatsort. Ezek eltérő klaszterezést adtak, mert a fitnesz értékük is eltérő. A legkisebb fitnesz értéke a Farthest Insertion algoritmusnak volt. Az algoritmusok futási ideje alacsony volt.

2. táblázat. *Futási eredmények a második adatsorra*

Algoritmus	**Klaszterek száma**	**Fitnesz érték**	**Futási idő (perc)**
Arbitrary Insertion	5	0.3173	$4.0341*10^{-5}$
Cheapest Insertion	5	0.3205	$1.6565*10^{-4}$
Farthest Insertion	6	0.2951	$2.8171*10^{-5}$
Greedy	5	0.3329	$1.9781*10^{-5}$
Nearest Insertion	7	0.2579	$2.3386*10^{-5}$
Nearest Neighbour	4	0.3574	$1.1303*10^{-5}$

A második adatsornál 5 csoport az optimális. Ezt az Arbitrary Insertion, Cheapest Insertion és a Greedy algoritmusnak sikerült megtalálnia. A Nearest Neighbour csak 4 csoportra bontotta az adatsort, míg a Farthest Insertion 6-ra, a Nearest Insertion 7 klaszterre. Minden algoritmus futási ideje alacsony volt a teszt során.

3. táblázat. *Futási eredmények a harmadik adatsorra*

Algoritmus	Klaszterek száma	Fitnesz érték	Futási idő (perc)
Arbitrary Insertion	5	0.5804	$3.3926*10^{-5}$
Cheapest Insertion	5	0.7828	$1.9795*10^{-5}$
Farthest Insertion	6	0.5994	$1.8061*10^{-4}$
Greedy	6	0.5593	$4.2401*10^{-5}$
Nearest Insertion	7	0.4879	$2.0339*10^{-4}$
Nearest Neighbour	5	0.6738	$5.21*10^{-6}$

A harmadik adatsornál 3 csoport lenne az optimális, ezt egyik algoritmusnak sem sikerült megtalálnia. A Nearest Insertion adta a legrosszabb megoldást, mert 7 csoportra bontotta az adatsort. Minden algoritmus gyors futási idővel dolgozott.

4. Összefoglalás

A cikk konstrukciós algoritmusokat mutat be, melyeket kisebb módosítással klaszterezésre használtam. A konstrukciós algoritmusok egy-egy lehetséges megoldást készítenek rövid futási idővel, lokálisan a legjobb lépéseket megtéve. Ilyen algoritmusok a legközelebbi szomszéd, beszúró heurisztikák és a greedy algoritmus. A cikkben három adatsor klaszterezése került bemutatásra, amely alapján elmondható, hogy az algoritmusok hatékonyan klasztereznek, megtalálják a klaszterhatárokat.

Köszönetnyilvánítás

„A cikkben/előadásban/tanulmányban ismertetett kutató munka az EFOP-3.6.1-16-2016-00011 jelű „Fiatalodó és Megújuló Egyetem – Innovatív Tudásváros – a Miskolci Egyetem intelligens szakosodást szolgáló intézményi fejlesztése” projekt részeként – a Széchenyi 2020 keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg."

Irodalom

[1] Bodon Ferenc, Buza Krisztián (2014): Adatbányászat. https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0064_55_adatbanyaszat/ar01s06.html

[2] Maulik, U., & Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique. Pattern recognition, 33(9), 1455-1465. **https://doi.org/10.1016/S0031-3203(99)00137-5**

[3] Brown, D. E., & Huntley, C. L. (1992). A practical application of simulated annealing to clustering. Pattern recognition, 25(4), 401-412. **https://doi.org/10.1016/0031-3203(92)90088-Z**

[4] Shelokar, P. S., Jayaraman, V. K., & Kulkarni, B. D. (2004). An ant colony approach for clustering. Analytica Chimica Acta, 509(2), 187-195. **https://doi.org/10.1016/j.aca.2003.12.032**

[5] Van der Merwe, D. W., & Engelbrecht, A. P. (2003, December). Data clustering using par-ticle swarm optimization. In The 2003 Congress on Evolutionary Computation, 2003. CEC'03. (Vol. 1, pp. 215-220). IEEE.

[6] Al-Sultan, K. S. (1995). A tabu search approach to the clustering problem. Pattern recogniti-on, 28(9), 1443-1451. **https://doi.org/10.1016/0031-3203(95)00022-R**

[7] Chandana, B., Srinivas, K., & Kumar, R. K. (2014). Clustering algorithm combined with hill climbing for classification of remote sensing image. International Journal of Electrical and Computer Engineering, 4(6), 923. **https://doi.org/10.11591/ijece.v4i6.6608**

[8] Łukasik, S., Kowalski, P. A., Charytanowicz, M., & Kulczycki, P. (2016, July). Clustering using flower pollination algorithm and Calinski-Harabasz index. In 2016 IEEE Congress on Evolutionary Computation (CEC) (pp. 2724-2728). IEEE. **https://doi.org/10.1109/CEC.2016.7744132**

[9] Kizilateş, G., & Nuriyeva, F. (2013). On the nearest neighbor algorithms for the traveling salesman problem. In Advances in Computational Science, Engineering and Information Tech-nology (pp. 111-118). Springer, Heidelberg. **https://doi.org/10.1007/978-3-319-00951-3_11**

[10] Kindervater, G. A., Lenstra, J. K., & Shmoys, D. B. (1989). The parallel complexity of TSP heuristics. Journal of Algorithms, 10(2), 249-270. **https://doi.org/10.1016/0196-6774(89)90015-1**

[11] Nilsson, C. (2003). Heuristics for the traveling salesman problem. Linkoping University, 38, 00085-9.

[12] Agárdi, A. (2021) Klasszikus klaszterező algoritmusok módosítása körút alapon, Multidiszcip-lináris Tudományok, 11(4), 81-86. **https://doi.org/10.35925/j.multi.2021.4.9**