Útvonaltervező algoritmusok

Az útvonaltervezés a mindennapok egyik leggyakrabban előkerülő problémája, mivel a közlekedésben, ahogy minden más területen, az energiaminimumra törekszünk, ezért a lehető „legjobb” utat szeretnénk megtalálni. Ez a terület a 20. században értékelődött fel igazán, ahogyan a világ egyre „kisebbé” vált, az utazások vagy áruszállítások egyre gyakoribbá és egyre inkább a hétköznapok részévé váltak, akár kis, akár nagy távolságokat tekintve.

Katona Géza
e-mail: geza.katona@mail.bme.hu

1. BEVEZETÉS

A kutatás célja, hogy megszülessen egy olyan útvonaltervező algoritmus, amely nagyméretű, több országot tartalmazó hálózatokon is képes hatékonyan működni, emellett integráltan kezeli az egyéni és a közösségi közlekedést.

A kutatás első lépéseként feltérképezésre kerültek a jelenleg széles körben alkalmazott útvonaltervező algoritmusok. Ennek eredményei jelennek meg a cikkben. Összegzéseképpen pedig kiválasztásra kerülnek azok az algoritmusok, amelyek alkalmasak lehetnek arra, hogy nagyobb méretű hálózatokon gyorsan, hatékonyan és optimális megoldásokat adjanak.

2. ALGORITMUSOK

Az útvonaltervezés problémájára számos algoritmus létezik, ezek közül a legismertebbek, illetve a leginkább ígéretes algoritmusok a következőkben szerepelnek.

2.1. DIJKSTRA

Napjainkban a leginkább elterjedt útvonalkereső algoritmus az Edsger W. Dijkstra-ról elnevezett módszer. Az alap probléma lényege az volt, hogy melyik a legrövidebb út Rotterdamból Groningenbe [1]. A munkáját végül 1959-ben publikálta a német *Numerische Matematik* című újságban [2]. Dijkstra a problémát a következőképpen fogalmazta meg [2][3]:

„Tekintsünk n darab pontot, amelyek közül néhányat vagy mindegyiket él köt össze, az élek hossza adott. Tegyük fel, hogy legalább egy él létezik valamely két pont között.

Nézzük az alábbi problémát:
Készítsük el az n csúcs minimális költségű gráfját. (Azt a gráfot, amelyben minden csúcs között egy és csakis egy út vezet.)
Az algoritmus első lépéseként három csoportra osztjuk az éleket.

I. Az eljárás során már elfogadott élek.
II. A következő lépésben az I. csoportba választandó élek halmaza.
III. A kimaradó élek (vagy már elutasítottuk őket, vagy még nem vizsgáltuk meg).

A csúcsokat két csoportba sorolhatjuk.
A: Azon élek végpontjai, amelyek az I. csoportban vannak.
B: A kimaradó csúcsok (egy és csakis egy II. csoportbeli él vezet minden ilyen kimaradó csúcshoz).

Kezdjük az eljárást egy tetszőleges A csoportbeli csúccsal, majd válasszuk ki azokat a II. csoportbeli éleket, amelyeknek egyik végpontja az A csúcs.

Kezdetben az I. csoport üres. Ezek után ismételjük az alábbi két lépést:

1. lépés: A II. csoport legrövidebb élét tegyük I-be és az eddig a B csoportban lévő végpontját tegyük A-ba.

2. lépés: Tekintsük azokat az éleket, amelyek az éppen előbb az A csoportba helyezett csúcsból indulnak, és egy B csoportbeli csúcsba érkeznek. Ha ezen élek közül valamelyik hosszabb, mint a neki megfelelő II. csoportbeli él, akkor elutasítjuk; amennyiben rövidebb annál, akkor kicseréljük őket és ezt az élt tesszük a II. csoportba, és a másikat vetjük el.

Ezután visszatérünk az 1. lépéshez és a két lépést addig ismételgetjük, amíg a B és a II. csoport üres nem lesz. Végül az I. csoportban lévő élek megadják a keresett fát"[3].

Az algoritmus működése az 1. ábrán látható.

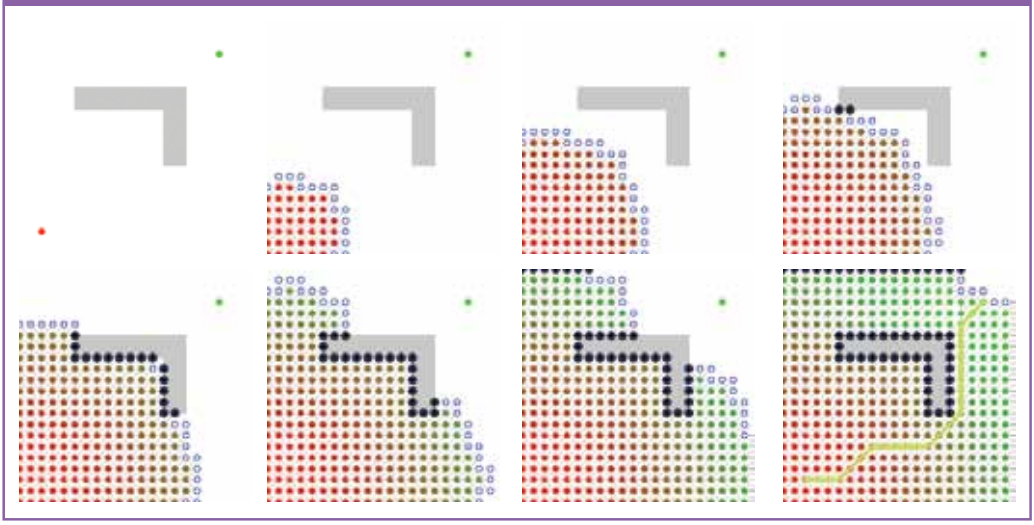
2.2. A* (A-star, A-csillag)

Az A* algoritmust Bertram Raphael, Nils Nilsson és Peter Hart alkototta meg 1968-ban [3]**.** Ez egy olyan eljárás, amelyben a céltól függően a csúcsokat súlyozzák. Az algoritmus a *legjobbat-először elnevezésű keresést valósítja meg***.** Az alapötlet az, hogy a gráf azon csúcsa felé indul el, amelynek az értéke a legkisebb. A csúcsok értékét egy úgynevezett kiértékelő függvény *f(n)* segítségével adja meg. Az algoritmus kulcseleme a *h(n)*-nel jelölt heurisztikus függvény: *Az n csomóponttól a célig vezető „legolcsóbb" út becsült útköltsége, amely ebben az esetben az a légvonalbeli távolság lesz, amely mindig a célponttól számolható[3].*

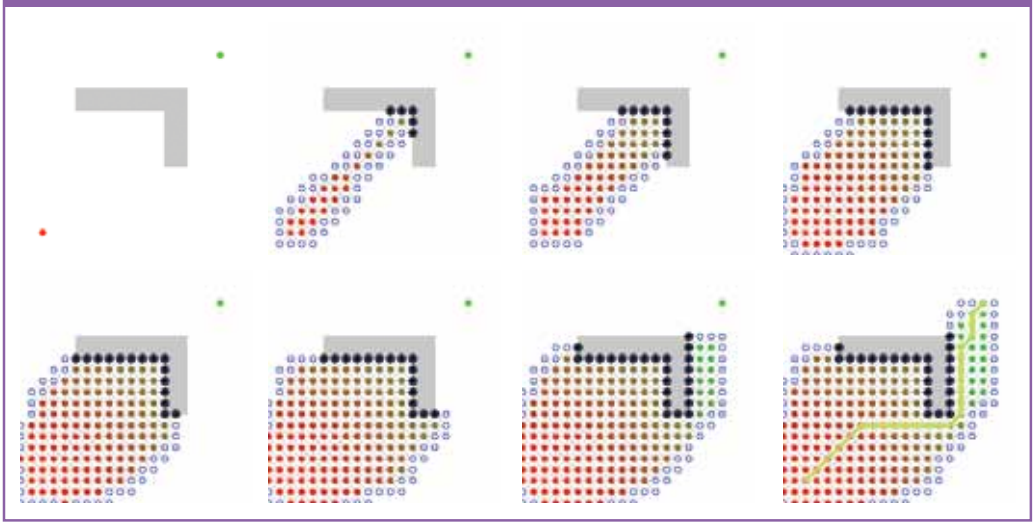
Az algoritmus a következőképpen működik: a keresés kezdetekor rendelkezésre áll a keresési gráf, amelyben adott a kezdő és célpont. A *h(n)* heurisztikus függvény rendelkezésre áll minden *n* csúcshoz, és ismertek a szomszédos csúcsok távolságai is. A működéshez szükséges egy, a már megvizsgált elemeket (jelölje: *Z*) és a már felderített, de be nem járt csúcsokat (jelölje: *Y*) tartalmazó lista. Itt tárolja az eljárás azon pontokat, amelyeket felderített és megvizsgált, illetve a még be nem járt, de már ismert pontokat. A hatékonyabb és gyorsabb lefutás érdekében a csak felderített csomópontokat az *f(n)* szerint sorba rendezve érdemes tárolni, így a legkedvezőbb csomópont már rendelkezésre áll [4].

A futtatás elején *Z* üres, és *Y*-ban is csak a kiindulópont szerepel. A futás során egy *g* változóban tároljuk a megtett távolságot. Az algoritmus az első lépésben kiveszi *Y*-ból a legkisebb *f*-fel rendelkező csomópontot, amely a rendezett tárolásnak köszönhetően mindig az első lesz. A kivett csomóponton végezzük el a vizsgálatot. Elsőként a

1. ábra A Dijkstra útvonaltervezés [23]



2. ábra Az A* útvonaltervezés [24]



környező elemekről eldöntjük, hogy bejárásra kerültek-e már? Abban az esetben, ha igen, tehát szerepel *Z*-ben, akkor továbblép az algoritmus, a többi esetben pedig kiszámításra kerül, hogy mekkora költséggel érhető el az a jelenlegi csomóponton keresztül. Ha valamelyik már szerepel *Y*-ban, akkor összehasonlítjuk, hogy a jelenlegi csomópontokon keresztüli út kisebb-e, mint az ott tárolt út. Ha „nem" a válasz, akkor az ilyen csomópontokat is kihagyjuk. A fennmaradó pontokhoz kiszámítjuk a *g* értékét, és útvonal elemeit, majd bekerülnek *Y*-ba. Ha az *Y*-ban szereplők között található olyan, amely pontnak az elérése így kisebb költségű, akkor azt töröljük. A lépés végeztével az aktuális pont bekerül *Z*-be és az *Y*-ból választunk egy új pontot. Ez mindaddig tart, amíg a soron következő elem nem a célpont, vagy *Y* üressé nem válik. Első esetben megtaláltuk a legrövidebb utat, a második esetben viszont kijelenthetjük, hogy nincs kapcsolat a két pont között. Amennyiben *Y* nem vált üres halmazzá, akkor visszafejtéssel meghatározható az optimális útvonal [4].

Az ábrasorozaton (2. ábra) látható az algoritmus működése.

2.3. Frederickson algoritmusa [6]

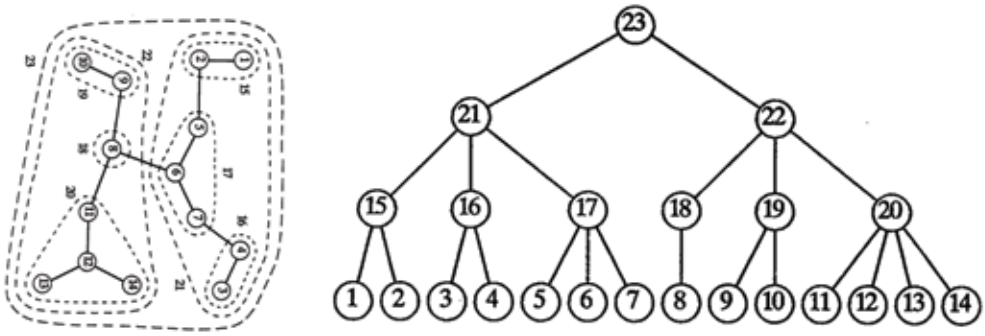
Lipton és Trajan megmutatták, hogy egy adott *n* csomópontú sík gráfból megtalálható időben lineárisan a csomópontok egy olyan méretű sorozata, amely eltávolítja a töréseket úgy, hogy a gráf minden részének a mérete legfeljebb $\frac{2}{3}n$ legyen. [5] Ez alapján G. N. Frederickson kidolgozta a gráfokra az r-részlegek fogalmát. Ezzel megtette az első lépést, hogy legyőzze az időbeli kötöttséget, azzal az ötletével, hogy több prioritású sorok különböző méretűek. Algoritmusa a gráfot kisebb régiókra, majd ezeket is további kisebb régiókra bontja. Az elő- és utófeldolgozásban a régiók között végrehajtja a Dijkstra algoritmust. Mivel ezek a régiók kicsik, így a Dijkstra prioritású sor számolás is kis terjedelmű – mivel csak kevés elemet tartalmaz – így a sor műveletek nem költségesek. Az algoritmus fő célja az, hogy így a Dijkstra számítás a gráfon lényegesen kevesebb határcsomópontot tartalmaz, mint *n*, ezért a sorbarendezési műveletek száma is sokkal kisebb, mint *n*. Ennek eredménye az úgynevezett topológia alapú kupac.

Az algoritmus működését ábrázolja a 3. ábra.

2.4. Fejlesztett Frederickson algoritmus [6]

Frederickson algoritmusából kiindulva Monika R. Henzinger, Philip Klein, Satish Rao és Sairam Subramanian kifejlesztettek egy gyorsabb útvonalkereső algoritmust a

3. ábra A Frederickson algoritmus működése [7]

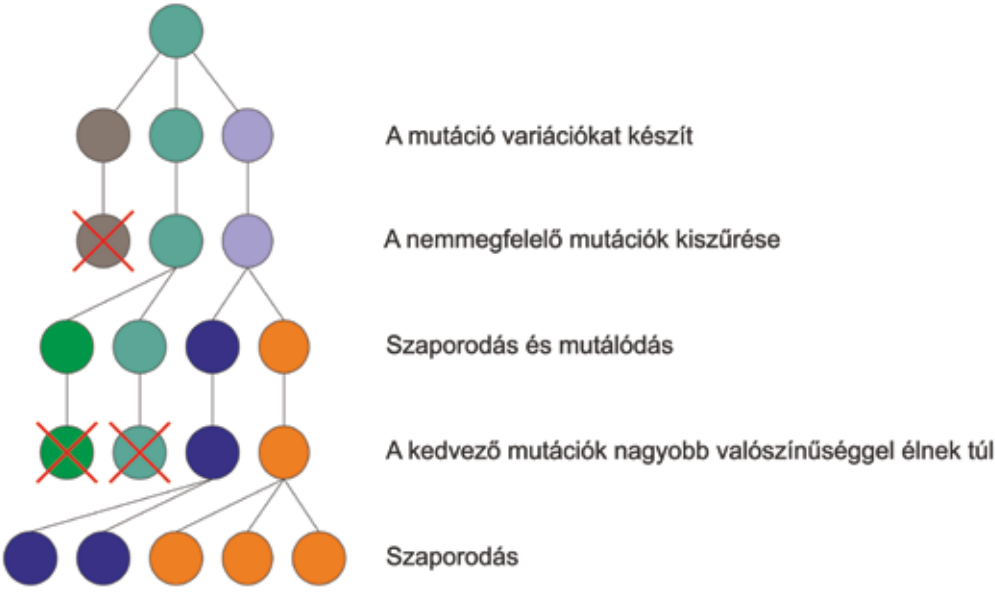


síkbéli gráfokra. Az algoritmusuk a nem negatív élhosszúságú gráfokra egy lineáris idejű maximális áramlású megoldást ad, ahol a kiinduló- és célpont azonos oldalon van. Abban az esetben, ha negatív élhosszúságok is megengedettek, akkor az algoritmus időszükséglete $O\left(n^{\frac{4}{3}}\log(n\cdot L)\right)$, ahol L a legnegatívabb hossz abszolút értéke. Az algoritmus alkalmas arra, hogy egy síkbeli hálózat megvalósítható számításának határai hasonlóak legyenek a legjobban illeszkedő síkbeli bipartit gráfhoz, és ahhoz, hogy megtalálja a maximális áramlást, ha a síkbeli gráf kiinduló- és végpontja nem azonos oldalon van. Az algoritmusnak párhuzamos és dinamikus verziója is létezik.

2.5. Genetikus algoritmusok

A genetikus algoritmusokat olyankor alkalmazzák, amikor valamilyen optimalizálási feladatot kell megoldani. Ebből kifolyóan al-

4. ábra Genetikus algoritmus működése [9]



kalmas lehet útvonalkeresési feladatok megoldására is. Általában egy olyan problématérben alkalmazzuk azokat, ahol ezt a teret egy folytonos függvénnyel írják le, és adott megkötések és feltételek mellett a függvény maximumát vagy minimumát keressük. Az ilyen eljárások a Darwin-i evolúciós elvekre vezethetők vissza, amelynek lényege, hogy a sikeresebb populáció marad életben, ami ebben az esetben azt jelenti, hogy a feltételeknek leginkább megfelelő. Ebből következik, hogy egy futás eredményeképpen nem egy megoldás, hanem a megoldások egy populációja áll elő, amelyek különböző keresztezéseken, mutációkon és szelekciókon mennek át, hogy végezetül egy jobb megoldást adjanak. [8]
Ezt szemlélteti a 4. ábra.

A genetikus algoritmusok működésének lelke a fitness-függvény, amely eldönti, hogy mennyire optimális az aktuális megoldásjelölt (kromoszóma). Minél inkább megfelel a feltételeknek, annál erősebbnek tekintjük, így az evolúció szabályai szerint ő fog nagyobb eséllyel öröklődni. A gyengébbek pedig egyre inkább kiszelektálódnak [8].

Az algoritmus futása a következő lépésekből áll:
– kezdeti populáció inicializálása,
– fitness-függvények kiértékelése a kezdeti populáció tagjaira,
– szelekció,
– keresztezés-mutáció,
– az új egyedek fitness-értékeinek kiszámítása,
– az új egyedek elhelyezése az új populációba.

Az algoritmus addig fut, amíg a következő feltételek valamelyike be nem következik:
– generációs limit: az algoritmus maximális futásának száma,
– max. fitness-limit: olyan érték, amelynél már feltételezzük, hogy elég jó,
– átlag fitness-limit: több megoldás keresése esetén alkalmazandó,
– konvergencia beállása: ha egy-egy újabb futásnál már érdemben nem javulnak az eredmények.

A genetikus algoritmusokat útvonalkeresésre eddig ritkán használták. Azonban az utóbbi években kezd felértékelődni a terület. 2009-ben Youfang Huanga, Chengji Lianga és Yang Yang publikáltak egy cikket a témában, amelynek lényege, hogy a Hollandiai Európai Konténer Terminálban a kikötői daruk által bejárt utat optimalizálják. Az útvonal kromoszómákat a következőképpen lehet megjeleníteni [10].

5. ábra Útvonal kromoszóma[10]

	x_1	x_2	x_3	…	x_{n-3}	x_{n-2}	x_{n-1}
v_i-edik kromoszóma	y_1	y_2	y_3	…	y_{n-3}	y_{n-2}	y_{n-1}

Az 5. ábra által mutatott megjelenítésben az útvonal egyértelműen azonosítható Y értékével, amely az elérhető területen fekszik. Adaptív közelítés alkalmazható a fitnessz-függvény meghatározására [11]. Az első lépésben a minimalizációs problémát maximalizációs problémára kell átalakítani, ehhez fel kell venni, hogy $f_1=\frac{1}{l}$ és f_2=d, így ez egy maximalizációs problémává vált két céllal: $max\{z_1=f_1(v), z_1=f_1(v)\}$. Az egyes generációk megoldásához két szélsőséges pont definiálható, a $z^+=\{z_1^{max}, z_1^{max}\}$ és a $z^-=\{z_1^{min}, z_1^{min}\}$. Ez alapján az aktuális populációra felírhatóak a következő egyenletek [10]:

$$\begin{aligned} z_1^{max} &= \max\{z_1(v_i), i = 1, 2, \dots, n\} \\ z_1^{min} &= \min\{z_1(v_i), i = 1, 2, \dots, n\} \\ z_2^{max} &= \max\{z_2(v_i), i = 1, 2, \dots, n\} \\ z_2^{max} &= \min\{z_2(v_i), i = 1, 2, \dots, n\} \end{aligned} \quad (1)$$

ahol n a populáció mérete. Ez alapján az adaptív súlyok:

$$w_1 = \frac{1}{z_1^{max} - z_1^{min}}$$
$$w_2 = \frac{1}{z_2^{max} - z_2^{min}} \quad (2)$$

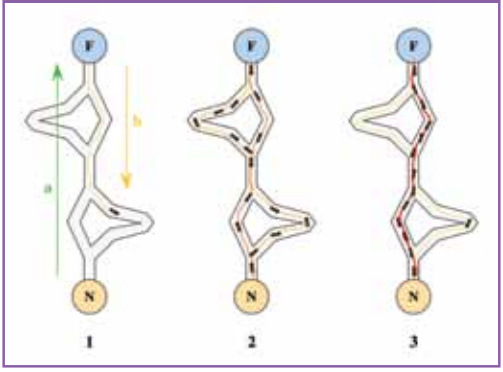
A genetikus algoritmusok hiányossága azonban, hogy ugyan nagyon jól tudnak globális optimumot keresni egy ígéretes területre, azonban ezen belül a pontos minimum vagy maximum megtalálására nem alkalmasak, ezért érdemes valamilyen egyéb heurisztikus módszerrel kiegészíteni azokat [12].

2.6. Hangyakolónia algoritmus

A modell alapjait Marco Dorigo rakta le.**[13]** Kutatásai során megfigyelte a hangyák élelemszerzési metódusát. Ennek lényege, hogy a hangyák teljesen véletlenszerű úton elindulnak a bolyból élelmet keresni. Ha élelmet találtak, akkor a visszafele úton feromont bocsájtanak ki. A feromon a többi hangya számára vonzó tulajdonságú, ezért amikor egy újabb hangya indul élelemért, akkor az irány választásakor nagyobb valószínűséggel ebbe az irányba indul. Minél több hangya halad el az adott útvonalon, annál erősebb a feromon nyom az útvonalon, így egyre több hangya választja majd ezt az útirányt. Emellett minél közelebb van az adott élelemforrás, annál többször tudnak megfordulni a hangyák, ami szintén a feromon nyom erősödését vonja maga után [14][15][16].
Ezt a folyamatot szemlélteti a 6. ábra.

Az útvonalkeresésre alkalmazott hangyakolónia algoritmusban a hangyák egy-egy járművet vagy utast reprezentálnak. Az útvonal addig növekszik, amíg az összes meglátogatandó hely bele nem kerül az útvonalba. Az algoritmus működése során minden hangya a kiindulási pontból indul. A következő meglátogatandó helyszín választása során az elérhető helyszínek közül történik a választás a jármű kapacitásának figyelembevételével. A hangya akkor tér vissza a kiinduló ponthoz, ha az összes meglátogatandó helyszínt bejárta, vagy a jármű a kapacitását elérte. Az *L* távolság a virtuális hangya által megtett útból számítódik. A második hangya az első hangya visszaérkezése után indul, és mindez addig folytatódik, amíg az előre megadott hangyaszámot el nem éri a program. Az algoritmus működése során minden hangyának minden pontot érintenie kell. A kiválasztáshoz a következő formula alkalmazható [14]:

6. ábra A hangyakolónia algoritmus működésének ábrázolása[25]



$$j=arg\ max\ \{(\tau_{iu})\ (\eta_{iu})^{\beta}\}\ amíg\ u \notin M_k,\ ha\ q \leq q_0,\ különben\ S, \quad (3)$$

ahol τ_{iu} az *i* jelenlegi pozíció és az *u* lehetséges pozíció közötti feromon mennyiség, η_{iu} pedig *i* és az *u* közötti távolság inverze, β a távolság fontosságát a feromon mennyiségével szemben kifejező tényező, M_k pedig a hangya által már bejárt utat tartalmazza. *q* egy véletlen változó a [0,1] zárt intervallumon, q_0 pedig egy paraméter. Ha minden választás megtörtént, a hangya a legnagyobb értékű élt választja az 3. egyenletből, hacsak *q* nem nagyobb, mint q_0. Ekkor a hangya egy véletlenszerű *S*-t választ, mely a p_{ij} eloszlás valószínűségén alapszik, ami a magas feromon tartalmú útvonalaknak kedvez [14]:

$$p_{ij} = \begin{cases} \frac{(\tau_{ij})(\eta_{ij})^{\beta}}{\sum_{w \in M_k} (\tau_{ij})(\eta_{ij})^{\beta}} & \text{ha } j \notin M_k \\ 0 & \text{ha } j \in M_k \end{cases} \quad (4)$$

Az (3)-as és (4)-es kifejezéseket alkalmazva a hangyák vagy a legkedvezőbb utat követik, vagy véletlenszerűen választanak a feromonok alapján. Az algoritmus addig folytatódik, amíg minden helyszínt nem látogatták meg és a bejárás nincs teljesen kész.

Az algoritmus legfontosabb része a feromon mennyiséget frissítő algoritmus. Az útvonalak aktualizálása magában foglalja a lokális és a globális frissítését a legjobb útvonalaknak egy előre meghatározott *m*-ig. A lokális frissítés metódusa a következő [15]

$$\tau_{ij}=(1-\alpha)\ \tau_{ij}+(\alpha)\ \tau_0 \quad (5)$$

ahol α a feromon kibocsájtását szabályozó paraméter, τ_0 pedig az él kezdeti feromon értéke.

Miután az előre meghatározott *m* darab hangya létrehozta a lehetséges útvonalakat, a globális aktualizálás során a legjobb útvonal által tartalmazott élekhez feromon kerül hozzáadásra egy darab hangya által az *m*-ből [15].

1. táblázat Az algoritmusokat összehasonlító táblázat

Jellemzői	Dijkstra	A*	Frederickson	Fejlesztett Frederickson	Genetikus algoritmusok	Hangyakolónia algoritmus
Pozitív	minden lehetséges útra is elkészíthető egyszerűen mihanzálható	nem becsüli felül a cél eléréséhez szükséges értéket optimális megoldással tér vissza [3]	nagy hálózatok alrégiókra bontása[6]	db szintet alkalmaz gyorsabb mint az alap Frederickson	sok lehetőségből hatékonyan választ globális optimumot jól számol	gyorsan található optimum a lokális optimumnál való leragadás ellen védett több hangyával/ hangya kolóniával növelhető a hatékonyság
Negatív	túl sok felesleges lépés a cél megtalálásának szempontjából negatív éleket nem kezeli	helytelen utakon elindult keresés által okozott időveszteség nagy memóriaigény nagy méretű problémákhoz nem alkalmas	csak három szintet használ[6] negatív éleket nem kezeli	negatív éleket nem kezeli	a véletlen szám generálás módja hat az eredményekre az eredmények két futás között eltérhetnek	a hangyák illetve a hangyakolóniák számának növekedésével drasztikusan nő az erőforrásigény alacsony hangyaszámnál nem biztosítható, hogy a legjobb út kerüljön kiválasztásra
Időszükséglet	0(n log n)	–	0(n log n)	$O\left(n^{\frac{4}{3}}\log(n\cdot L)\right)$	–	–

$$\tau_{ij}=(1-\alpha)\,\tau_{ij}+\alpha(L)^{(-1)} \qquad (6)$$

Ez a megoldás a legjobb út használatára ösztönöz. Az eljárás egy előre meghatározott számszor fut le. **[15]**

A hangyakolónia algoritmust továbbfejlesztve lehetőség nyílik többféle befolyásoló tényező figyelembevételére, mint például a járműkihasználtság, pénzbeli haszon, időszükséglet, stb…. Ez a következőképpen írható fel [15]:

$$\tau_{ij}=\tau_{ij}+B\cdot\tau_{ij}\cdot H_{lij} \qquad (7)$$

ahol a B konzervatív és a felfedező keresés egyensúlyát biztosítandó szám, H_{lij} a hasznossági függvény [14].

Az egyenletekben szereplő α és β értékekeket a szakirodalom alapján célszerű kiválasztani [13] [14][15][17][18].

2.7. Algoritmusok összehasonlítása

Az 1. táblázatban összefoglaltuk a korábban ismertetett algoritmusok főbb jellemzőit. Elmondható, hogy a Dijkstra algoritmus az egyik legegyszerűbben elkészíthető az ismertetett algoritmusok közül. Előnye még, hogy a teljes hálózat felderítésére, így az összes lehetséges út megtalálására alkalmas. Azonban az ismertetett algoritmusok közül ez rendelkezik a leghosszabb válaszadási idővel. Emiatt kell vizsgálni további algoritmusokat, amelyekkel gyorsabban elvégezhető az útvonaltervezés. A nagy időszükséglet miatt általában csak addig fut az algoritmus, ameddig meg nem találja a megadott pontok közötti útvonalat, ezután már nem folytatja az útvonaltervezést.

Az A* algoritmus az olyan jellegű útvonaltervezést valósítja meg, mint amikor „torony iránt” próbálunk menni. Ez nagyon jól működik, ha

2. táblázat Futási idők a Dijkstra, az A* és a Genetikus algoritmusokra[20]

Gráf mérete	Futási idők (s)			
	Dijkstra	A* egyenesvonalú heurisztikáva	A* halmozott vonali csomópontok	Genetikus algoritmus
40	7	7	22	5
80	120	120	456	51

nincsenek zsákutcák vagy akadályok, mert akkor kifejezetten gyors, ellenkező esetben viszont nagyon lelassul a működése.

Frederickson algoritmusa azt az ötletet használja fel, hogy megpróbálja csoportosítani a gráf csomópontjait, és ezen csomópontok között futtatja le a Dijkstra algoritmust. Ilyen csoportok lehetnek például a kerületek, városok, azonban egy nagyobb hálózaton a szintek száma is nagymértékben növekszik, így ez csak korlátozott mértékben javítja az eredeti Dijkstra algoritmus működését.

Az eddig felsorolt algoritmusok jellemzően a legjobb útvonalat keresték, azonban nagyon sokszor alternatív útvonalak meghatározására is szükség van, mivel egy bonyolult hálózaton két adott pont között több azonosan jó útvonal is előfordulhat, amelyek közül egyéb szempontok szerint történik a választás. Emiatt szükség van olyan algoritmusokra, mint a genetikus vagy a hangyakolónia, amelyekkel a komplexebb hálózatok jól kezelhetők, illetve képesek több alternatívát is megvizsgálni és kezelni.

Ezek alapján tehát az egyik legfontosabb szempont az összehasonlításnál az időszükséglet, mivel a keresést végzők szempontjából ez a legfontosabb. Sajnos egzakt módon nem lehet összehasonlítani az összes eljárást, mert főképp a genetikus algoritmus és a hangyakolónia algoritmus a beállított paraméterektől függ. Azonban adott esetben elvégezhető az összehasonlítás.

A Dijkstra, az A* és a genetikus algoritmusok összehasonlítását végezte el néhány kutató [20]. A konkrét esetre a következő értékeket kapták (2. táblázat).

Látható, hogy a gráf méretének növelése milyen drasztikus mértékben növeli a futási időket. Összehasonlítva a három módszert, a genetikus algoritmus lényegesen gyorsabban szolgáltat eredményt.

Más kutatók [21] elvégezték a genetikus és a hangyakolónia algoritmus összehasonlítását. Természetesen ezek az adatok nem összevethetők az előző adatokkal, de mégis viszonyítási pontot jelenthetnek az algoritmusok értékeléséhez és a közöttük való döntéshez. A mérési eredményeiket a 3. táblázat tartalmazza.

Az adatok alapján a hangyakolónia algoritmus ebben az esetben lényegesen gyorsabban ad megoldást a genetikus algoritmusnál.

3. táblázat A hangyakolónia- és a genetikus algoritmus összehasonlítása

A terület komplexitása	Relatív egyszerű		Relatív komplex		Komplex	
Algoritmus	Hangyakolónia algoritmus	Genetikus algoritmus	Hangyakolónia algoritmus	Genetikus algoritmus	Hangyakolónia algoritmus	Genetikus algoritmus
idő (s)	477,21	1522,08	648,69	1869,28	972,64	2560,42
iterációk száma	30	50	45	80	50	100
populáció	20	20	25	35	30	50

3. MULTIMODÁLIS RENDSZEREK

A közlekedéshez a mindennapok során általában többféle közlekedési eszközt veszünk igénybe. Többször kombináljuk az egyéni és közösségi közlekedést, illetve a helyi és a távolsági közlekedést. Emiatt kell az útvonaltervezésben is a különböző közlekedési módok kombinációjával foglalkozni, és nem célszerű elválasztani egymástól az egyéni és a közösségi közlekedést sem, mivel az utazók több esetben is valamilyen kombinációt használnak. A különböző közbringa rendszerek terjedésével pedig egyre inkább felértékelődik a két közlekedési filozófia együtt kezelése. A hivatásforgalom tekintetében is megjelenik ez a kérdéskör a P+R és B+R rendszerek terjedése miatt. Ennek eredményeképpen egy rengeteg csomópontot tartalmazó gráfot kell felépíteni, amelyben szerepel a közúthálózat és az ehhez szervesen kapcsolódó közösségi hálózat. Az ezen való keresés azonban igen erőforrásigényes, ezért hatványozottan fontos, hogy a lehető leggyorsabb és a leginkább erőforrás-kímélő algoritmust alkalmazzuk.

Nagyobb méretű hálózatok esetén az előbbiekben megemlített módszerek közül egyik sem képes elfogadhatóan gyors megoldást szolgáltatni. Emiatt további kutatások szükségesek, hogy a lehető leggyorsabb és a leghatékonyabb módszer kifejlesztésre kerüljön. A legújabb kutatásokban a fentebb említett rendszerek valamilyen kombinációjával próbálják a problémát feloldani. Ebbe az irányba sorolható be három ázsiai származású kutató által készített tanulmány [12]. Ez a munka a kikötői daruk dinamikus ütemezésének problémáját kezeli a kikötőhelyek elosztásának tervezésében egy heurisztikus algoritmussal kiegészített genetikus algoritmussal [12].

A hatékonyság azonban nem csak a módszerek kombinációjával növelhető, hanem a hálózat bizonyos szempontú felosztásával, szétdarabolásával is. Ezt úgy lenne célszerű elvégezni, hogy a keveset változó, nagy csomópontokat összekötő hálózatokat, amelyeket az emberek gyakran keresnek/használnak, azokat letárolnánk, és adott időközönként olyan szempont szerint megvizsgálnánk, hogy még mindig az az útvonal a legjobb? Erre a célra azok a városok lennének alkalmasak, ahol jelentős forgalmú repülőterek, vasúti csomópontok, illetve távolsági autóbusz-pályaudvarok vannak. Magyarországon belül ilyen lehetne Budapest, Debrecen esetleg Szeged. Első lépésben meg kellene határozni az ilyen városok főbb kilépési pontjait, Budapest esetén példának okáért a fejpályaudvarok, a Kelenföldi pályaudvar, a Népligeti autóbusz-pályaudvar, a repülőtér és a fő közlekedési útvonalak (autópályák, főutak…) lehetnének ilyenek (7. ábra). A működési területen ki kell majd jelölni az összes ilyen nagyobb csomópontot, amelyek között létre kell hozni különböző szempontok szerint az eljutási lehetőségeket leíró táblát. Mivel ez a keresés előre elvégezhető és minden lehetséges kapcsolat felépítése szükséges, alkalmazható a Dijkstra algoritmus is.

7. ábra Budapest főbb tömegközlekedési csomópontjai [22]

Azon hálózati elemeknek, amelyek az előző listába nem kerültek be, dinamikusan kell kapcsolódni ehhez a törzshálózathoz, azaz a kiindulási és érkezési cím környezetében meg kell keresni a legközelebb eső nagy csomópontokat, és valamilyen útvonalkeresési eljárással, a megadott szempontok szerint fel kell építeni az utazást.

Erre a célra olyan algoritmus szükséges, ami gyors, akár párhuzamosítható is, mivel ebben az esetben a kiindulási pontok számossága a végtelenhez közelít, a rendelkezésre álló idő pedig a felhasználói elvárások miatt szűkös. Ebből kifolyólag a jelenlegi algoritmusok közül a hangya- és a genetikus algoritmus az, amely alkalmas lehet. A hangyakolónia algoritmus nagy előnye, hogy az egy időben kereső hangyák száma növelhető, így lényegesen csökkenthető a kereséshez szükséges idő, azonban a teljesítményigény jelentősen növekszik. A genetikus algoritmusok előnye abban áll, hogy egyszerre vizsgálják az összes lehetséges útvonalat, azonban hátrányuk, hogy előzetesen ismerni kell a lehetséges útvonalakat, és az optimumkeresést meg kell támogatni egyéb heurisztikus eljárásokkal.

ÖSSZEFOGLALÁS

Az útvonaltervezés tekintetében a világ egyre inkább afelé halad, hogy a „lehető legjobb" útvonalat sikerüljön megtalálni a legrövidebb idő alatt, így csökkentve a „költségeket" (pénzbeli, időbeli…). A jelenlegi megoldások általában kisebb, illetve jól lehatárolt területekre koncentrálnak. Ennek eredményeképpen kisebb, jobban kezelhető hálózatokkal dolgoznak, így a futási idő és a komplexitás csak kisebb mértékben kerül előtérbe. Azonban az egyre inkább globalizálódó világban, illetve a járművek minél jobb kihasználásának (Car sharing, Telekocsi, Bubi…) igénye mellett egyre hangsúlyosabbá válik annak az igénye, hogy egy integrált rendszerben lehessen a közlekedést tervezni. A cikkben bemutattuk a jelenleg legelterjedtebb útvonaltervező algoritmusokat, leírtuk a működésük lényegét, és összehasonlítottuk azokat a gyorsaság szempontjából. Ez alapján arra a következtetésre jutottunk, hogy a genetikus és a hangyakolónia algoritmus a leginkább alkalmas arra, hogy a multimodális útvonaltervezésre alkalmazzák, így a jövőbeli kutatások erre irányulnak

FELHASZNÁLT IRODALOM

[1] **Dijkstra, Edsger Wybe**: *Oral history interview with Edsger W. Dijkstra*, Charles Babbage Institute, University of Minnesota, Minneapolis, 2001.08.02

[2] **Dijkstra, Edsger Wybe**: *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, 1959, 269-271

[3] **Podobni Katalin:** *Legrövidebb útkereső algoritmusok diplomamunka,* ELTE TTK , 2009

[4] **Hernáth Zoltán:** *Valós idejű adaptív A* útkeresési algoritmus***,** MSc önálló labor 2 összefoglaló, BME VIK, Méréstechnika és Információs Rendszerek Tanszék Intelligens Rendszerek Kutatócsoport, 2012,

[5] **R. J. Lipton and R. E. Tarjan:** *A separator theorem for planar graphs***,** SIAM J. Appl. Math. 36, 1979, 177-189

[6] **Monika R. Henzinger, Philip Klein, Satish Rao, Sairam Subramanian:** *Faster Shortest-Path Algorithms for Planar Graphs***,** journal of computer and system sciences 55, 3-23 oldal, 1997, article no. SS971493

[7] **Greg N. Frederickson:** *Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications***,** Purdue University, Computer Science Technical Reports, Department of Computer Science, West Lafayette, 1984, report no. 83-449

[8] **Zsolnay Károly:** *Genetikuis algoritmusok,* BME VIK, 2012/2013 I. félév

[9] **Danielle Venton:** *Feature - Evolving towards the future of science: genetic algorithms and grid computing***,** iSGTW, 2008.02.27

[10] **Youfang Huanga, Chengji Lianga, Yang Yangb:** *The optimum route problem by genetic algorithm for loading/unloading of yard crane***,** Computers & Industrial Engineering 56, Intelligent Manufacturing and Logistics, 993-1001 oldal, 2009

[11] **Gen, M., & Cheng, R.:** *Genetic algorithms and engineering optimization***,** New York: John Wiley & Sons, 2000

[12] **Chengji Lianga, Youfang Huanga, Yang Yang:** *A quay crane dynamic scheduling problem by hybrid evolutionary algorithm for berth allocation planning***,** Computers & Industrial Engineering 56, Intelligent Manufacturing and Logistics, 1021-1028 oldal, 2009

[13] **Marco Dorigo:** *Optimization, learning and natural algorithms (in Italian)***,** Ph.D. Thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[14] **Kovács Gábor:** *Elektronikus fuvar- és raktárbörze rendszermodellje***,** Ph.D. értekezés, Budapesti Műszaki és Gazdaságtudományi

Egyetem, **Közlekedésmérnöki és Járműmérnöki kar, 2011.**
[15] **John E. Bella,Patrick R. McMullenb:** *Ant colony optimization techniques for the vehicle routing problem*, Advanced Engineering Informatics, 2004. Január, 41-48 oldal
[16] **Bonabeau, E., Dorigo, M., Theraulaz, G.:** *Swarm intelligence: From Natural to Artificial Systems*, Oxford University Press, 1999, ISBN 0-19-513159-2
[17] **Dorigo, M., Gambardella, L. M.:** *Ant colony system: A cooperative learning approach to the travelling salesman problem,* IEEE Transactions on Evolutionary Computation, 1, 1997, 53-66 oldal, DOI: 10.1109/4235.585892
[18] **Dorigo, M., Stützle, T.:** *Ant Colony Optimization,* Ant Colony Optimization, MIT Press, 2004, ISBN 0-262-04219-3
[19] **Russell, Stuart J., Norvig, Peter, Canny, John F.:** *Mesterséges intelligencia modern megközelítésben,* Panem kiadó, Budapest, 2005, ISBN 963-545-411-2
[20] **A.R. Soltani, H. Tawfik, J.Y. Goulermas, T. Fernando:** *Path planning in construction sites: performance evaluation of the Dijkstra, A, and GA search algorithms,* Advanced Engineering Informatics, 291-303 oldal, 2002
[21] **Fatemeh Khosravi Purian, Fardad Farokhi, Reza Sabbaghi Nadooshan:** *Comparing the Performance of Genetic Algorithm and Ant Colony Optimization Algorithm for Mobile Robot Path Planning in the Dynamic Environments with Different Complexities,* Journal of Academic and Applied Studies, 29-44. oldal, 2013 Február, ISSN1925-931X
[22] **BKK Utastájékoztatás:** *Budapest és környékének vasúti hálózata,* BKK, 2014.03.27.
[23] **Wikipedia (user:Subh83):** *Illustration of Dijkstra's algorithm,* Wikipedia, 2011. 04.13
[24] **Wikipedia (user:Subh83):** *Illustration of A* search algorithm,* Wikipedia, 2011. 04.13
[25] **Johann Dréo:** *Shortest path find by an ant colony,* Wikipedia, 2006. 05.27

Route planning algorithms

Route planning is one of the most common problems of everyday life. During the transport in our daily lives – as in all other areas – we strive to use minimum energy, so we seek to find the "best" possible route. This area has become increasingly important over the 20th century as the world became "smaller", and travel and the transportation of goods became more and more frequent and part of our everday life, taking into account both short and long distances.

The goal of the research is to create a route planning algorithm which is able to work effectively with large networks spreading over several countries, and which offers an integrated approach to individual and public transport.

As the first step of the research, routing algorithms which are currently widely used have been mapped. The results are introduced in the article. The summary lists those algorithms that may be useful in larger networks to provide fast, effective and optimal solutions.

Routenplanungsalgorithmen

Routenplanung ist eine der häufigsten Probleme des Alltags, weil wir in unserem täglichen Leben während des Transports - wie in allen anderen Bereichen - uns bemühen, minimale Energie zu ver-wenden, so dass wir versuchen, die "besten" von den möglichen Routen zu finden. Dieser Bereich wurde wirklich im 20. Jahrhundert aufgewertet, da die Welt „kleiner" wurde, und die Reisen und die Warentransporte – sowohl auf kurzen als auch auf langen Strecken - immer häufiger und zum Teil unseres Alltags geworden sind.

Das Ziel der Forschung war, einen Routenplanungsalgorithmus zu erstellen, der auch in über mehrere Länder verbreiteten großen Netzwerken effizient benutzt werden kann, wobei auch einen integrierten Ansatz zu den individuellen und öffentlichen Verkehrsmitteln angeboten wird. Als erste Stufe der Forschung wurden die derzeit am häufigsten verwendeten Routenplaner-Algorithmen aufgelistet und bewertet. Im Artikel wurden die Ergebnisse vorgestellt. Als Zusammenfassung es wurden die Algorithmen ausgewählt, die geeignet sind, in großen Netzwerken eine schnelle, effektive und optimale Lösungen zu bieten.