**Legrövidebb utak keresése**

**ALT algoritmussal**

*Kiss Bence*

*Kinyik Bence Tamás*

# **Bevezetés**

## **Az ALT algoritmus**

Az ALT algoritmus Landmarkokat és a háromszög egyenlőtlenséget használja fel annak érdekében, hogy gyorsan lefusson, és optimális javaslatot adjon. A Landmarkok egyfajta tájékozódási pontok, amelyek valamilyen információt nyújtanak a gráfban található többi pontról. Ebben az esetben minden Landmark-hoz tároljuk az összes többi csúcsponthoz mért légvonalbeli távolságát. Ahogy láthatjuk az ALT algoritmus abban tér el az A\* algoritmustól, hogy segédinformációkat tárol a gráfról. Ahhoz, hogy ezek a Landmarkok rendelkezésünkre álljanak, a gráfnak előfeldolgozáson kell átesnie. A Landmarkok megtalálásához az úgynevezett Landmark kereső algoritmusok nyújtanak segítséget. Több féle Landmark kereső algoritmus létezik, amelyek eltérő keresési eredményekhez vezetnek attól függően, hogy hogyan helyezkednek el a keresett útvonalhoz viszonyítva. A jó Landmark olyan, hogy vagy a keresett útvonal kezdőpontja előtt van, vagy a végpontja után[[1]](#footnote-1). Emiatt a jó Landmark kereső algoritmus a Landmarkokat a gráf szélein helyezi el. Tőbb Landmark kereső algoritmust használtunk és a különböző algoritmusokkal kapott eredményeket hasonlítottuk össze.

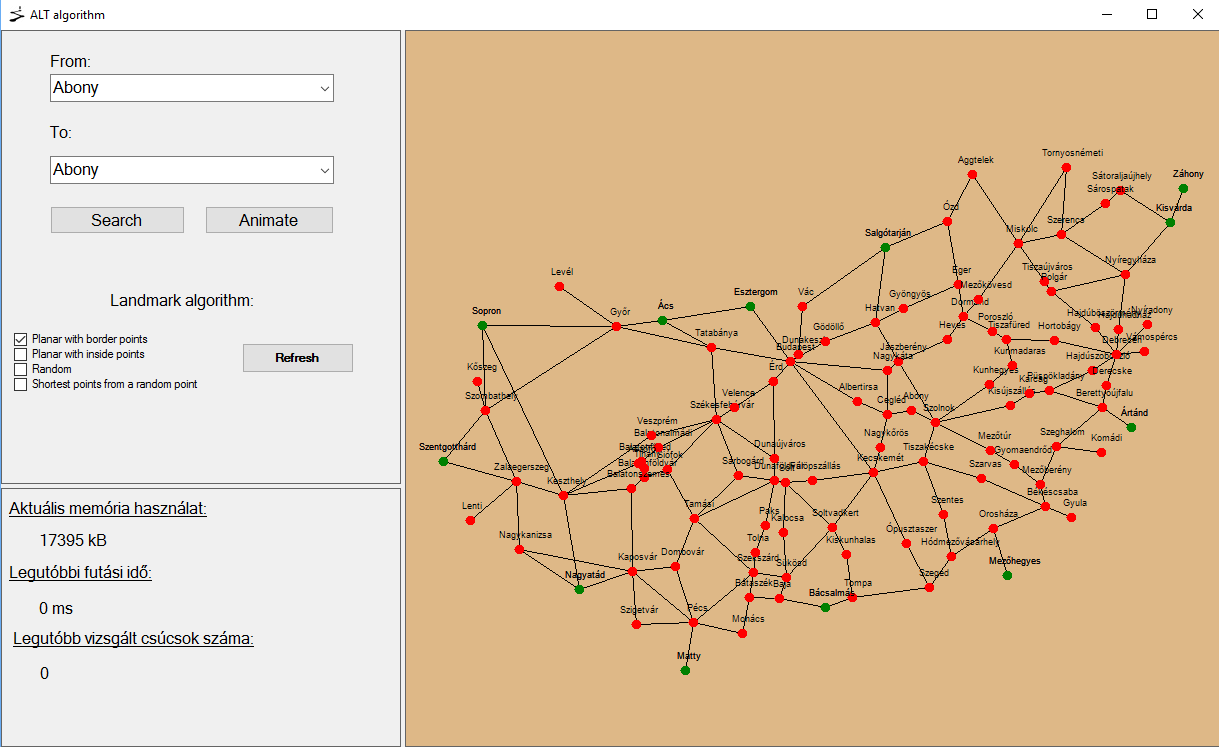
## **Landmark kereső algoritmusok (Bence ezt te írd meg, ide csak az elméleti háttér, az implementáció később)**

* 1
* 2
* 3
* 4

# **Implementáció**

Az implementációhoz c# nyelvet használtunk, a grafikus felhasználói felülethez pedig Windows Forms-os megoldást választottunk.

A programban egy Magyarország térképet közelítő gráfon tudunk utakat keresni települések között. Az adatokat úgy vettük fel, hogy azok közelítsék a valós értékeket. Az egyes településekhez tartozó csúcspontok elhelyezkedését egy Magyarország térkép segítségével adtuk meg, míg a szomszédos városok közötti távolságok meghatározásához egy távolságkereső weboldalt[[2]](#footnote-2) használtunk fel.

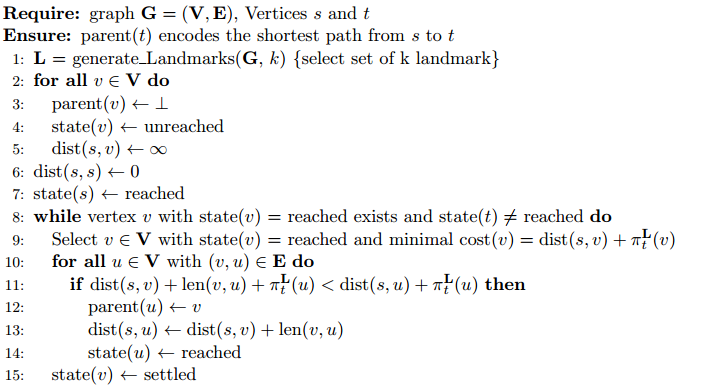
1. ábra

Az első ábrán láthatunk egy képet a programról. A baloldal felső részén látható a kezelő panel. Itt tudjuk kiválasztani a két várost, amelyek között keressük az utat. A „Search” feliratú gomb megnyomásával indíthatjuk el a keresést. Miután befejeződött a keresés a program kirajzolja a kapott útvonalat kék színnel, a két végpontot pedig kiemeli. Az „Animate” feliratú gomb megnyomásával animálni tudjuk a keresés folyamatát. Végig tudjuk követni, hogy az algoritmus milyen pontokat vizsgál meg, merre keresi az utat. A gombok alatt tudjuk kiválasztani a használt Landmark kereső algoritmust. A megfelelő checkbox-ot kiválasztjuk, majd megnyomjuk a refresh gombot. Ekkor a program előfeldolgozza a gráfot, és megkeresi a megadott Landmark kereső algoritmussal a Landmarkokat. Miután megkereste a Landmarkokat, kirajzolja azokat a térképre zöld színnel. A kezelő panel alatt található néhány információ a program futásáról. Láthatjuk a program által használt memória méretét, a legutóbbi útvonal keresés időtartamát (ez az információ csak a Search gomb megnyomása esetén íródik ki, mert animáció esetén az animáció sok időt vesz igénybe és meghamisítja a mérési adatot) és a legutóbbi futtatás során vizsgált csúcspontok számát. A képernyő jobb oldalán található a kirajzolt térkép. Piros pontokkal a csúcspontok, fekete vonalakkal a köztük futó élek vannak jelölve. Az egér görgője segítségével tudunk zoom-olni a kirajzolási képernyőn és bal egérgombot lenyomva tudjuk áthelyezni a gráfot.

## **Gráfok implementációja**

A gráfokat a Vertex és az Edge osztályok segítségével írjuk le. A Vertex osztály a csúcspontokat, míg az Edge osztály az éleket reprezentálja. Mindkét osztály a GraphBuilder.cs fájlban található. A Graphbuilder osztály felelős a gráf létrehozásáért és annak kezeléséért. Tartalmaz egy listát a csúcspontokról, az élekről és tárol néhány információt a kereséshez. A Draw függvény meghívásával tudjuk kirajzolni a gráfot, amely meghívja Vertex és az Edge saját Draw függvényét és a megfelelő alakzatot kirajzolja. Az Edge osztály két Vertex-et tárol (kezdő és végpont). Az egyik végpont megadásával az Edge objektumtól le tudjuk kérdezni a szomszédját. A gráfok adatait csv fájlokból tudjuk beolvasni, ezek a GraphDatas mappában található Edges.csv és Vertes.csv fájlok.

## **Algoritmus implementáció**

Az algoritmust az alábbi pszeudokód[[3]](#footnote-3) írja le:

A parentNodes Dictionary tárolja az egyes Vertexekhez azt a Vertexet, ahonnan oda jutottunk az algoritmus futtatása során. Ez alapján tudjuk később kirajzolni a kapott útvonalat. ReachedNodes tárolja azokat a csúcsokat, amiket már elértünk az algoritmus során, ezek közül választunk a következő lépésben. Tároljuk hozzájuk a Landmarkok alapján számolt költségeket, hogy ki tudjuk választani a minimális költségűt.

## **Landmarkok implementációja**

A Landmarkokat tároljuk a GraphBuilder osztályban. A hozzájuk tartozó légvonalbeli távolságot a Vertexekhez tárolt koordináták alapján számoljuk úgy, hogy beszorozzuk egy váltószámmal (GraphBuilder osztály \_zoom változója). Ezt a váltószámot úgy kapjuk meg, hogy megkeressük a legrövidebb élt, majd ennek az értékét elosztjuk a két végpontjának légvonalbeli távolságával.

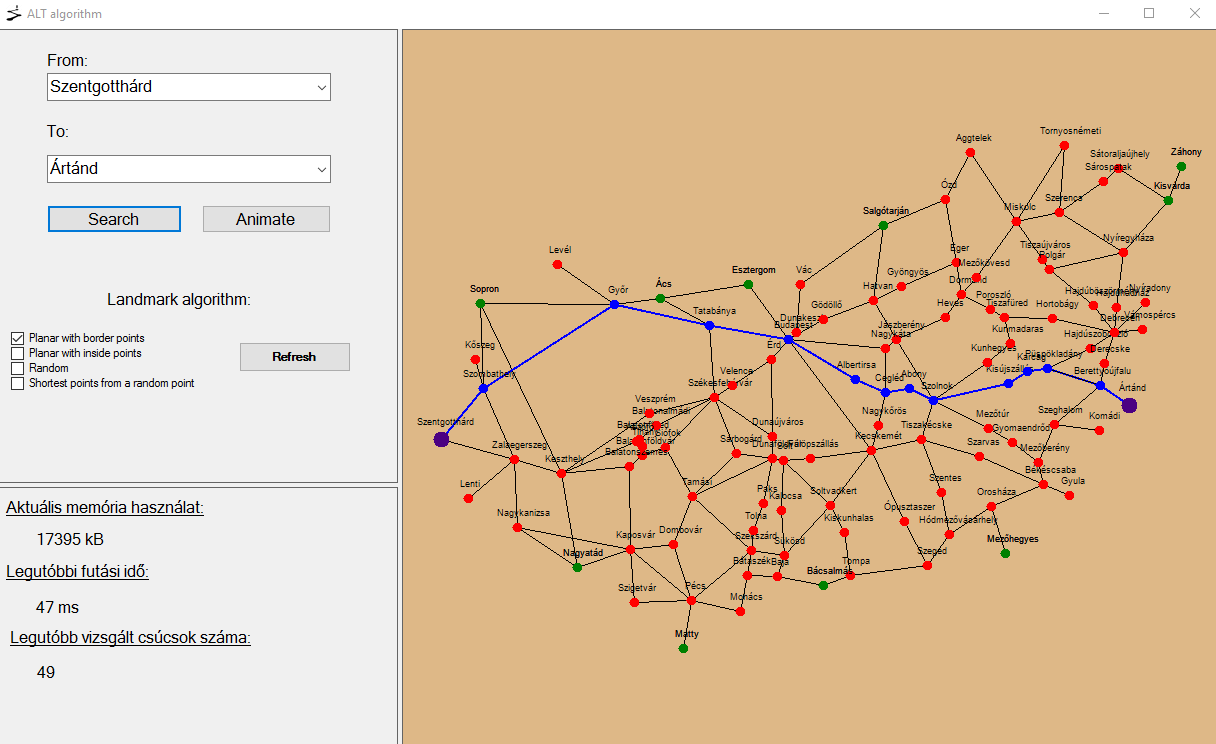
### **Algoritmusok**

Bence itt írd le az algoritmus implementációját

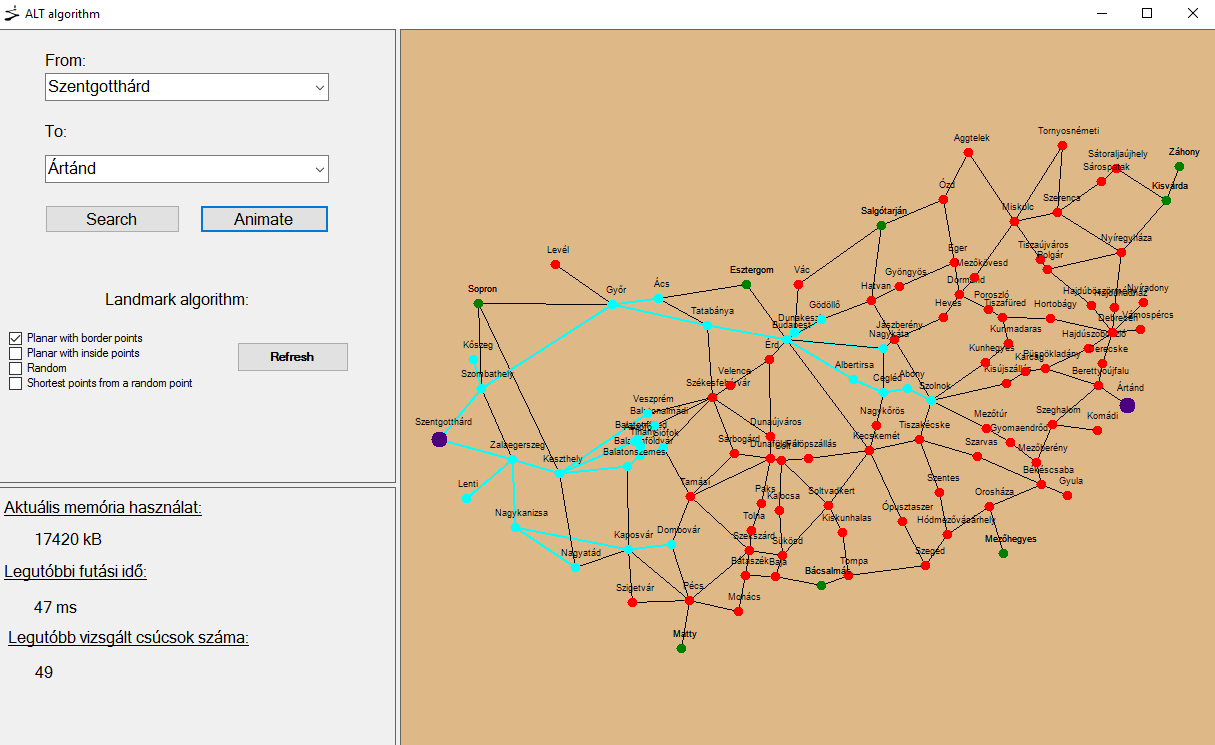
csinálj képeket is az egyes landmark elhelyezkedésekről

# **Program futtatása**

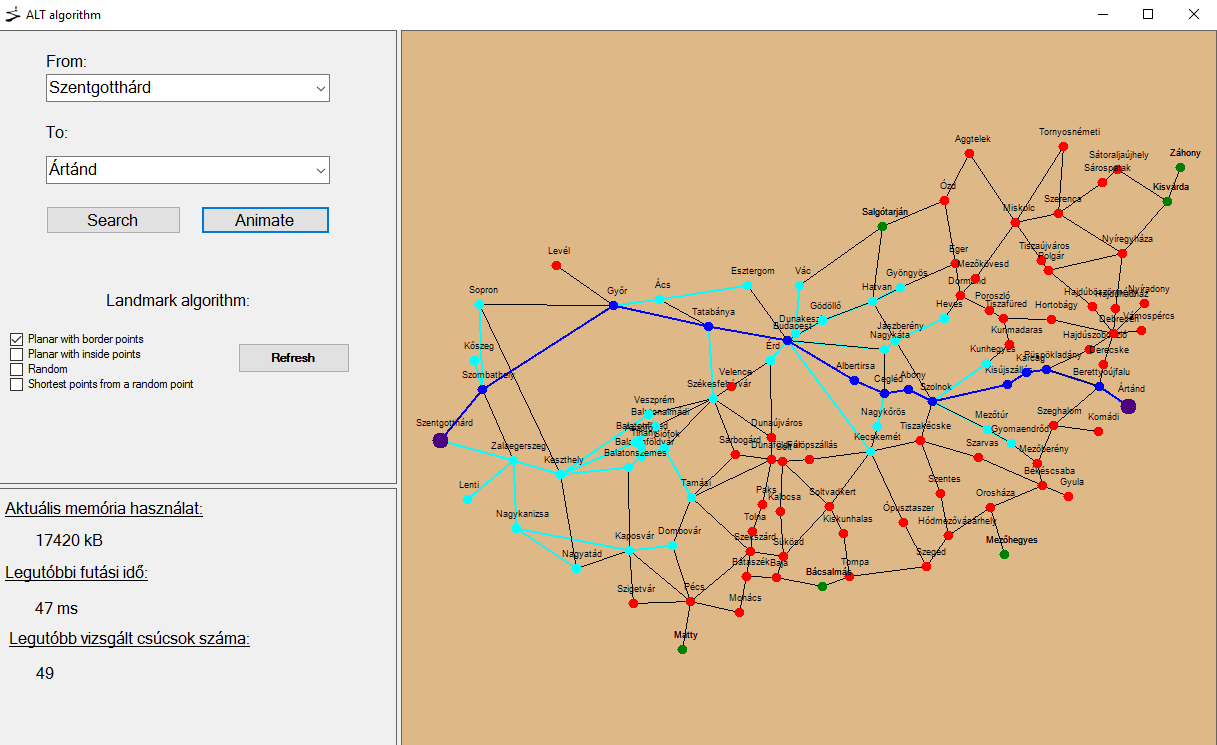
Szentgotthárd és Ártánd közötti legrövidebb út meghatározása:



Láthatjuk, hogy kiválasztottuk Szentgotthárdot és Ártándot a kezelő panelen. Miután megnyomtuk a Search gombot a képernyő jobb oldalán megjelenik kék színnel kiemelve az útvonal.

Keresés animálása:

Az Animate gomb megnyomása után a program a képernyő jobb oldalán elkezdi animálni az útkeresést. Világoskék színnel rajzolja a gráf azon részét, amit már megvizsgált az algoritmus. 100 milliszekundumonként új csúcsot vesz be a vizsgált csúcsok közé és újrarajzolja a képernyőt.

Végeredményként a program megjeleníti a kapott útvonalat:

# **Kísérletezés**

A keresési eredményekhez csak a kirajzolásokhoz készítettünk képeket, a kapott mérési eredményeket táblázatba gyűjtöttük ki.

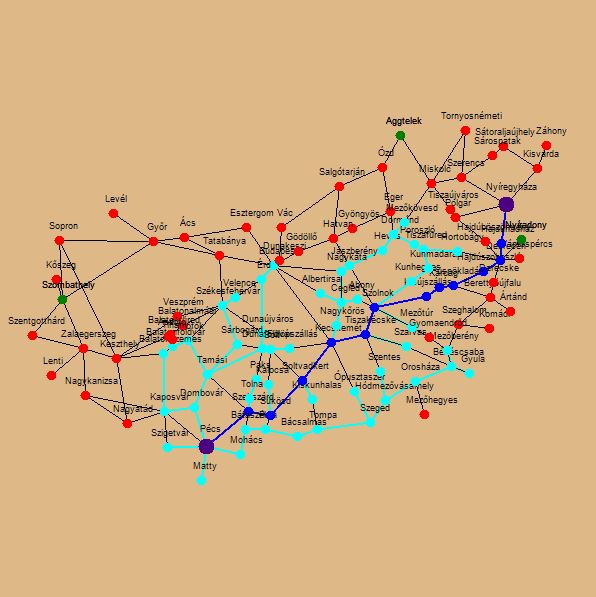
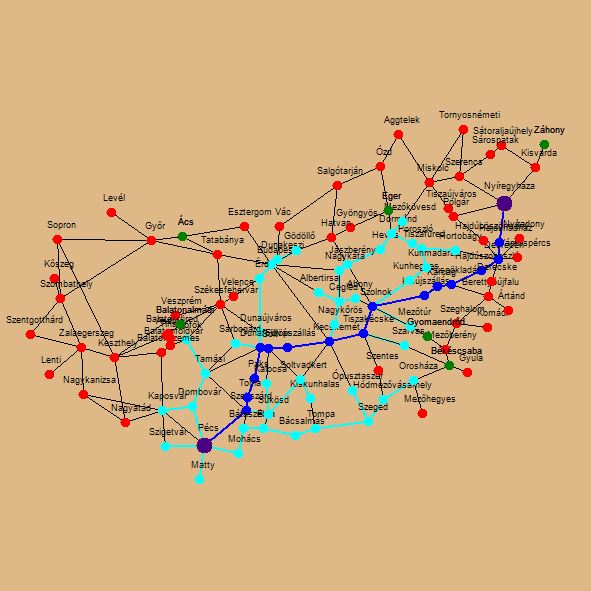
Az futási idő a vizsgált csúcspontok számától függ, azonban a legutóbbi futási időben lehetnek eltérések a számítógép terheltsége miatt, így a mérésekhez a vizsgált csúcspontok számát elemezzünk ki.

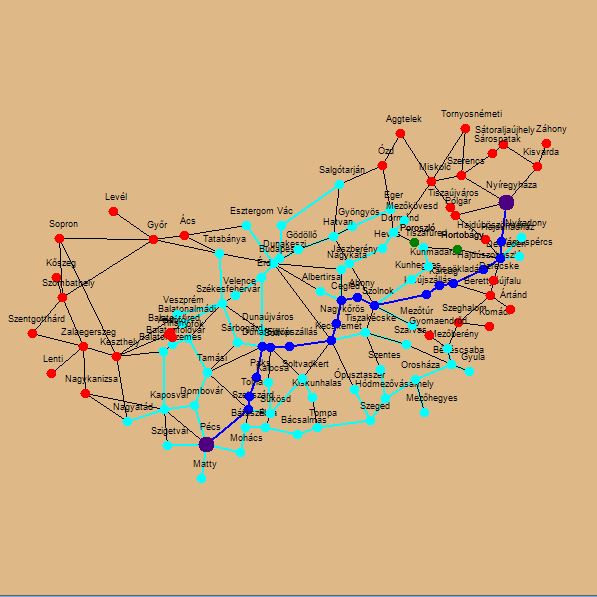
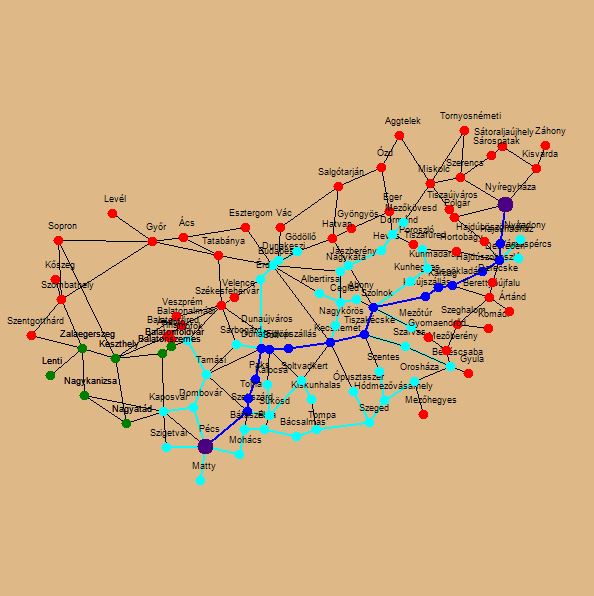
6 féle mérést végeztünk különböző elhelyezkedésű Landmarkokkal:

* Planar width border points
* Planar width inside points
* Random
* Random (jobb elhelyezkedés)
* Shortest points from a random point
* Shortest points from a random point (Kezdőpontnál)

3 útvonalat vizsgáltunk meg:

## Pécs – Nyíregyháza közötti útvonal

 *border points inside points*

 random random(jobb elhelyezkedés)

shortest shortest(kezdő pontnál landmarkok)

1. http://logic.pdmi.ras.ru/midas/sites/default/files/midas-werneck.pdf [↑](#footnote-ref-1)
2. http://www.tavolsag.hu/ [↑](#footnote-ref-2)
3. http://i11www.iti.uni-karlsruhe.de/\_media/teaching/theses/studienarbeitfabianfuchs.pdf [↑](#footnote-ref-3)