TDK

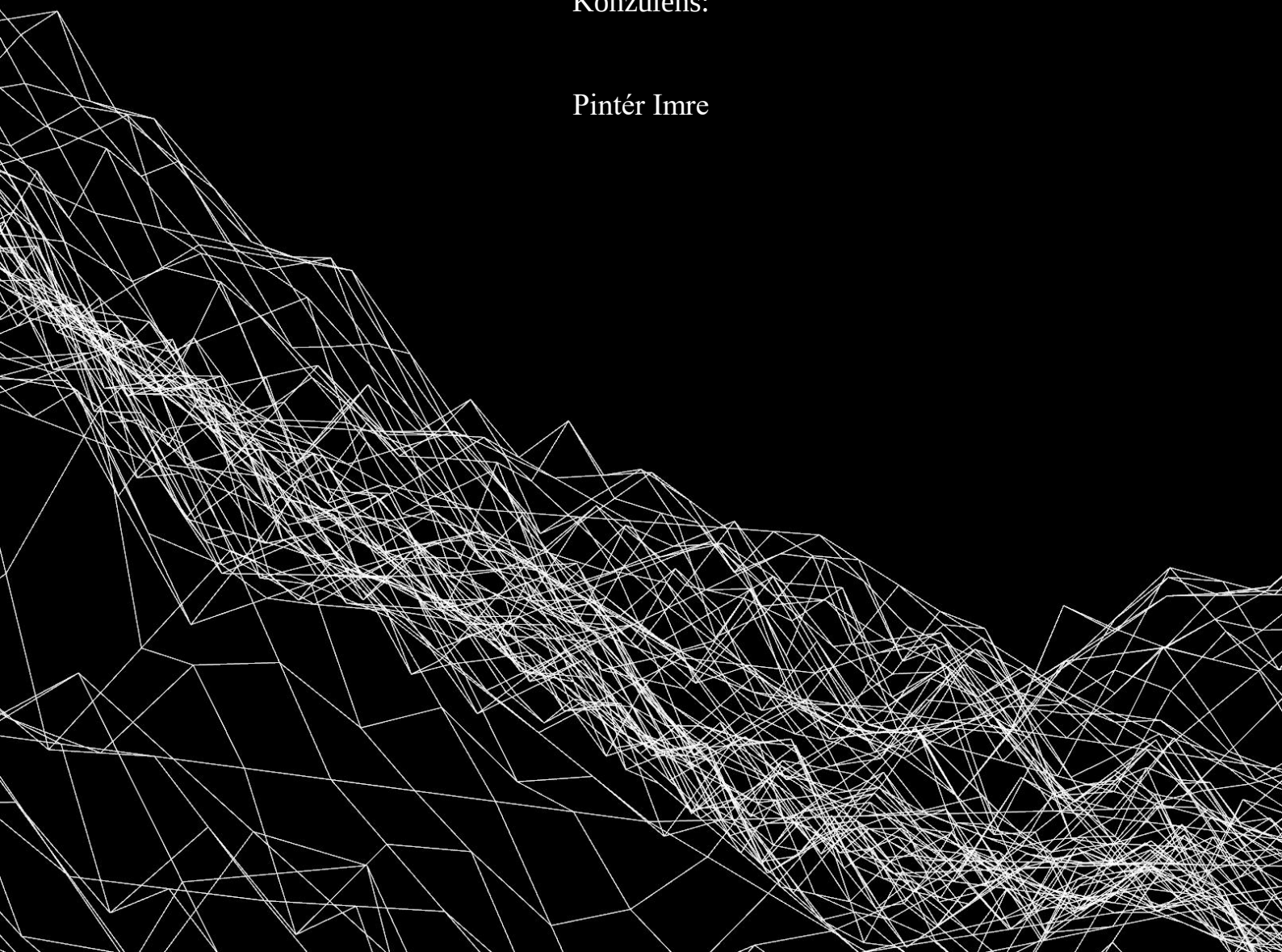
2018

Boronkay Gábor

Tartószerkezet tervezés evolúciós eljárásokkal

Konzulens:

Pintér Imre



Tartalom

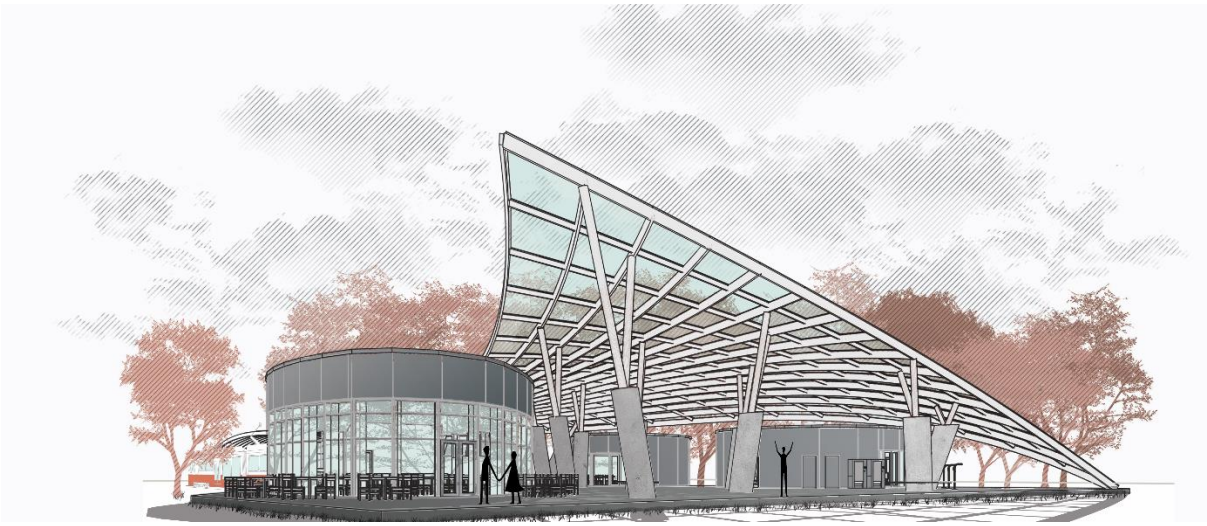
Boronkay Gábor BJYYHA
Konzulens: Pintér Imre

Bevezetés

A mai világban egyre nagyobb reflektorfényt kapnak olyan témák, mint a mesterséges intelligencia, tanuló algoritmusok. Óriás és kicsi tech cégek fektetnek mély tanuló algoritmusokba és életünket is egyre több helyzetben nem látható algoritmusok segítik. A kérdés jogosan merülhet fel, vagyis az építészet az algoritmusok által irányított világban hogyan alakulhat át.

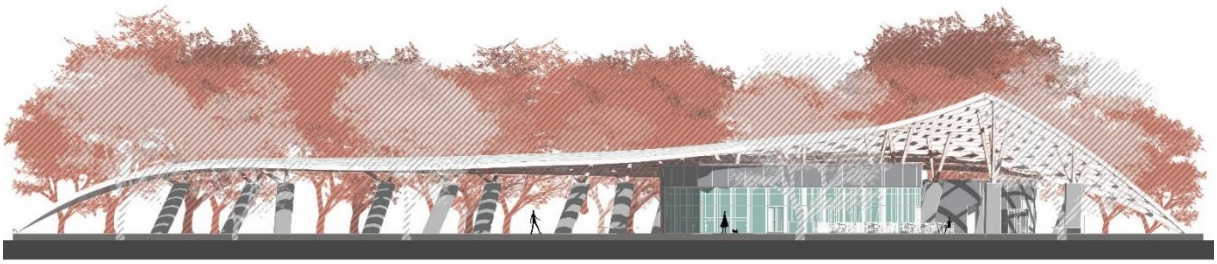
Ez a folyamat nem most kezdődött az építész szakmában sem és helyét a mai napig keresi. Viszont tendenciaként megfigyelhető, hogy egyre több tervezőiroda egyre több energiát és pénzt fektet az építészet algoritmikus megközelítésébe. Az építészek is elkezdték a programozást úgy használni, mint egy tervezési eszközt; legyen szó koncepció alkotásról vagy akár kiviteli tervek készítéséről. Sokszor nem is szükséges egy adott programozási nyelvet - mint például Python-t vagy C#-ot - megtanulni, az építészeti vizuális gondolkodáshoz közel álló vizuális programozási felületek is elérhetők már. Erre jó példa a Grasshopper 3D alkalmazás is. Ezt, a McNeel Rhinoceros CAD szoftverhez tartozó kiegészítőt használják olyan irodák is, mint a Zaha Hadid, Norman Foster, Bjarke Ingels, Thomas Heatherwick stúdió is. A vizuális programozás segítségével könnyen elérhetővé és használhatóvá válnak olyan misztikusnak tűnő algoritmusok, mint az evolúciós vagy mély tanuló programok. Sokszor hallhatjuk ezeket a kifejezéseket, de hogy valójában miről van szó és milyen módon hasznosíthatjuk lehetőségeiket, már ritkán. Ebben az írásomban egy gyakorlati példán keresztül mutatom be milyen lehetőséget nyújthatnak az evolúciós eljárások szerkezettervezés szempontjából, mikor lehet és mikor érdemes használni azokat és milyen szempontokat kell figyelembe venni alkalmazásuk közben.

[1]



A téma körüljárásához a Komplex tervezési feladatomat választottam, mely a XII. kerületbe tervezett fogaskerekű végállomására ad építészeti javaslatot. A tervezési téma (hely, funkció stb.) választásánál szempont volt az is, hogy egy olyan építészeti program és elképzelés álljon elő, mely teret ad a kísérletezésre. A tervezett végállomás fejépületének lényeges elemei a pavilonszerűen elrendezett épületek és a peront, valamint az átmeneti tereket lefedő, átlátszó – légies - tető. A pavilonok funkciói: kávézó és kiállító tér, személyzeti pihenő és gépészeti helyiség vizesblokk-kal, továbbá bicikli kölcsönző és javító állomás. A peron és átmeneti tér lefedését egy kétszergörbült felülettel oldottam meg. A tető formáját a fő közlekedési irányok is alakították; a peron végén vonalmentén ér a földhöz, onnan pedig a Normafa felé szétnyílva ad védelmet az eső ellen. A tetőszerkezet a Normafával ellentétes, az autóút felöli oldalon még egy pontban letámaszkodik, az erdő felé pedig gesztusszerűen felemelkedik. Az algoritmus ebből a felületből képzi a tető hierarchikus szerkezetét és a véletlenszerűnek tűnő oszlopos alátámasztását. A szelvényméretek és oszlopok pozícióját és dőlésszögét már evolúciós és végeselem módszerekkel definiáltam. A pontos eljárást, hogy hogyan is épült fel a script, azonban csak az evolúciós folyamatok bemutatása után ismertetem. Előtte röviden a használt alkalmazásokat és fontosabb paramétereiket, továbbá a fontosabb definíciókat tárgyalom, így, a későbbiekben egyszerűbben értelmezhetőek lesznek az eljárás lépései olyanoknak is, akik még nem használtak evolúciós megoldó motort.

[2]

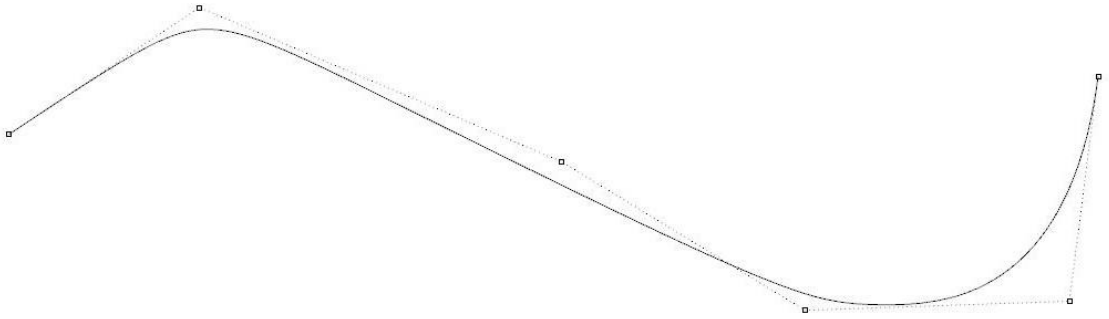


McNeel Rhino 3D, mint CAD szoftver

A Rhino 3D szoftver egyre inkább elterjedt modellező szoftver, használják autógyártásban, termékdesignban, cipőgyártásban, ékszeriparban és az építészetben is egyre több iroda alkalmazza, sokszor csak koncepció fázisában, de akár kiviteli tervekhez is megfelelő eszköz lehet. A szoftvert a McNeel cég fejleszti, az első verziót 1998-ban adták ki. A legjelentősebb eltérés a megszokotthoz képest, hogy a modellezés alapját NURBS azaz Non-Uniform Rational B-Spline-ok adják. Ezek matematikai görbék, melyeket a következő paraméterek írnak le:[1]

- **Fokszám:** Egy egyenes 1-es fokszámmal rendelkezik, egy kör 2-es fokszámú. A legtöbb szabad formálású görbe 3 és 5 közötti fokszámmal rendelkezik. Ezt a számot akár fel lehet fogni úgy, mintha a görbe szabadsági foka lenne. Az értéket, ami a fokszám + 1-gyel egyenlő, hívjuk **rangnak**.
- **Vezérlő-pontok:** A görbe görbületét ezekkel a pontokkal tudjuk definiálni. 1 fölött a pont magához húzza a pontot, 0 és 1 között taszítja. Egy görbéhez tartozó vezérlő pontok száma nem lehet kisebb, mint a görbe rangja.
- **Csomók:** Ez egy számlista, ami a görbe simaságát határozza meg (fokszám +N-1)
- **Értékelési szabály:** Ez a formula, mely meghatározza magát a görbét a bemenő paraméterek alapján.

[3]



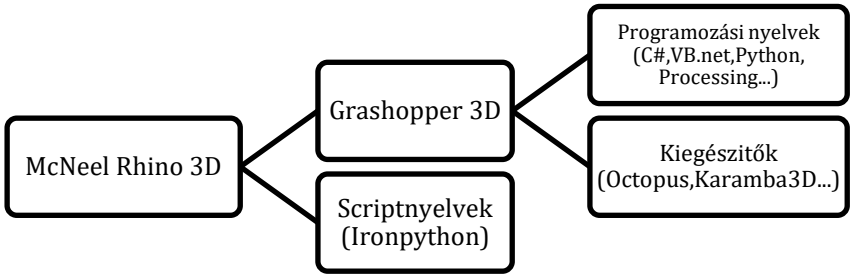
A Rhino egyik erősségét ezek a szabadon formálható görbék és az ebből származtatható felületek adják. Megfelelő mértékű precizitással lehet benne létrehozni felületeket. Ezeket a felületeket leginkább úgy lehet elképzelni, mint egy gumilap.

[1] (Tedeschi, 2014)

Ezen felül alkalmas a szoftver a megszokott Bezier görbe, illetve poligonháló alapú modellezésre is. Sokat használják 3D nyomtáshoz is és különböző CNC megmunkáló gépek vezérléséhez is. A szoftver tudása kiegészítőkkel bővíthető, ezek közül érdemes kiemelni a Grasshopper 3D-t. Ezzel a bővítménnyel képesek vagyunk magát a szoftvert parametrikusan is használni.

Grasshopper 3D

[4]



Az egyik legjelentősebbnek mondható kiegészítője a Rhino-nak a Grashopper 3D, mely lehetővé teszi a program egyszerű parametrikus irányítását. Ezt a kiegészítőt a McNeelnél hobbi projektként kezdték el fejleszteni és 2007-ben jelent meg belőle az első verzió. Az 1.0-s verzió számot még a mai napig nem érte el, 2017-ben belekezdtek az újraírásába, mivel eddig egyszerre volt VB.net-en és C#-ban írva. Az új verzió teljes egészében C#-ban íródik, de továbbra is támogatni fogja a Pythont, VB.net-et, Processinget és szinte bármelyik programozási nyelvet. A program alapvetően egy rajzlapból épül fel, melyre az előre elkészített komponenseket tudjuk lepakolni és köztük kapcsolatokat létesíteni. Az eredményt a Rhino-ban háromdimenzióban láthatjuk. A vizuális programozás egyik kötöttsége, hogy az adat csak balról jobbra haladhat. Elméletben nem lehetségesek rekurzív folyamatok az ilyen típusú adatáramlás miatt, de ezek is megoldhatók kiegészítőkkel vagy programozással. Magának a szoftvernek a lényege, hogy nem a megszokott módon modellezünk, hanem egy szabályrendszert állítunk fel, bemenő és kimenő paraméterekkel. Így, ha egy geometriát definiáltunk és változtatni akarunk tegyük fel a magasságán, nem szükséges újra modellezni, elég csupán a magassági paramétert változtatni. Az algoritmusunk újraépíti azt a neki lefektetett szabályok alapján. Szinte bármilyen bonyolult struktúrát képesek vagyunk így definiálni. Mivel valójában nem is modellezünk, hanem egy szabályrendszert definiálunk, azt bármikor módosíthatjuk, felülvizsgálhatjuk. A fontosság vagy az egyedisége a gondolkodásban rejlik: a programozási logika, a programozói gondolkodás a vizuális scriptelés által bekerült-bekerül az építészek

„eszköztárába”. De ez nem csak egy eszköz, hanem inkább módszertan. Következménye pedig, hogy valami új dolog születhet belőle. Más nagy cégek is elkezdték beintegrálni a programcsomagjaikba ezt a fajta megközelítést, például az Autodesk a Dynamo nevű kiegészítőjével. A Grasshopper 3D erejét a tudása mellett a mögötte álló közösség is adja. Rendkívül gyorsan lehet információt keresni és rendkívül sok hasznosabbnál hasznosabb kiegészítők készülnek az alapcsomaghoz. A Grasshopperen belüli parametrikus szerkezettervezéshez, méretezéshez például a Karamba 3D-t érdemes használni, A Karamba 3D egy végeselem alapú statikai program. Honlapjukon a www.karamba3d.com -on rengeteg példa projekt van, a legtöbb a Bollinger + Grohmann cég nevéhez fűződik.

Optimalizációs motorok[1]

Optimalizációs motorok alatt értem az összes olyan megoldó algoritmust, ami valamilyen peremfeltételek felállítása mellett egy optimális megoldást keres és ad vissza. Későbbiekben láthatjuk majd, hogy nem mindig csak egy válasz létezik az általunk definiált kérdésre. Ebben a dolgozatban többnyire az evolúciós megoldó motorokkal foglalkozom, de a teljesség kedvéért ismertetem a többi lehetőséget is, amit a Grasshopper 3D környezete kínál.

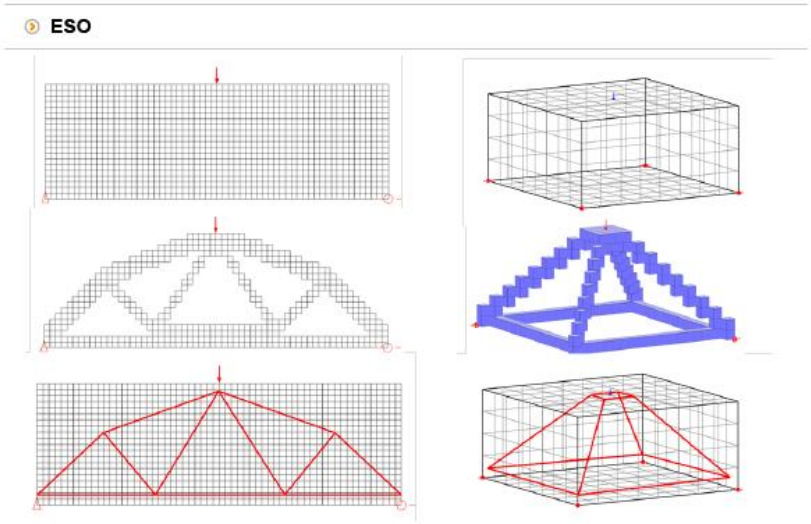
Topológia optimalizáció

Ez alatt értjük azt a fajta geometriai optimalizációs eljárást, amikor a formát a topológia szabályai alapján változtatjuk. Az első ilyen eljárás az ESO [5] (Evolutionary Structural Optimization), melyben definiálnunk kell a “játszóteret”, azt a környezetet, amin belül az algoritmus módosíthat a geometrián. Egy VEM modell felállítását követően az algoritmus az általunk megadott Rejection Ratio (RR) alapján és a testben ébredő Von Mises igénybevételek alapján anyagot kezd kivonni, folyamatosan vizsgálva a test állékonyságát. Az anyagelvételt a Rejection Criterion (RC) alapján teszi. Például, ha az RR-t 50%-ra állítjuk, az algoritmus mindenhol elvesz anyagot, ahol az RC értéke kisebb, mint 50%. Miután a szerkezet elért egy stabil állapotot, további optimalizáláson esik át, amiért az Evolutionary Rate (ER) felelős.
A XESO (Extended ESO) eljárás alapjaiban hasonlóan működik, mint az ESO, viszont több irányítási lehetőséget hoz az eljárásba. Kontúrvonalakat határoz meg a testet érő erőhatások és a Von Mise-i erők alapján, ahol nem teljesül a $\sigma_d < \sigma_{lim}$, ott anyagot von el, ahol teljesül, ott

[1] (Tedeschi, 2014)
[5] http://astruttie.aroad.co.kr/index.php/2016/03/17/evolutionary-structural-optimization-method-e-s-o/

anyagot ad hozzá a geometriához. A XESO alapú algoritmus működését dupla irányú evolúciós eljárás segíti. Efféle eljárásokat a Grasshopperen belül is képesek vagyunk létrehozni, a Millipede kiegészítő segítségével, melyet Kajima Sawaka és Michalatos Panagiotis készített lehetőséget adva VEM analízisre is. Továbbá létezik ún. BESO (Bidirectional Evolutionary Structural Optimization) eljárás is, mellyel szintén anyag elvételével és hozzáadásával tudjuk optimalizálni a szerkezetünk geometriáját. Előnye a többi eljárással szemben, hogy gyorsabb, optimálisabban használja ki a számítógép erőforrásait. Grasshopper 3D-n belül is léteznek kétirányú evolúciós számítások, a legújabb kiadású Karamba 3D-vel már gerendákat és héjszerkezeteket is tudunk optimalizálni.

[5]



Exact solvers:

Ezek az algoritmusok egy matematikai formula alapján működnek és mindig ugyanazt az eredményt adják vissza. Ilyenre képes például a Goat nevű kiegészítő, melyet Simon Flöry jegyez. Az Exact Solver algoritmusok csak egy fitness értéket tudnak figyelni. Fitness alatt értjük azt, hogy egy adott állapotban az algoritmus eredménye mennyire felel meg elvárásainknak. Ez bármilyen számérték lehet, például, ha egy görbén levő pont távolságát keressük egy adott egyenestől, ez a fitness értékünk ebben az esetben. Ezt az értéket optimalizálhatjuk, úgy, hogy a maximális vagy minimális távolságot szeretnénk eredményként kapni. A fitness érték pontosabb működését később ismertetem. A Goat több számítású metódust is ismer attól függően, hogy az alaphelyzetben közel állunk-e a megoldáshoz vagy távol. Ettől függően fog véletlenszerű próbálkozások alapján egy, és mindig ugyanarra a megoldásra jutni.

Evolúciós eljárások:

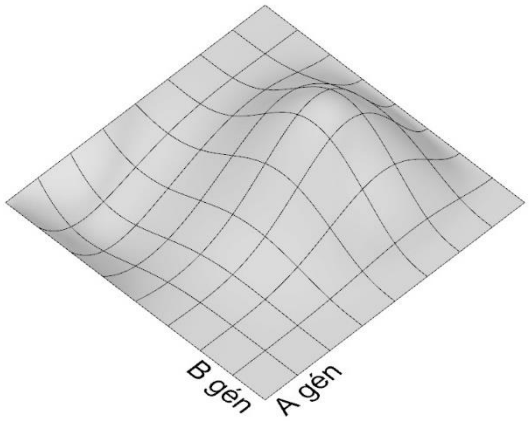
Evolúciós eljárások alatt azokat a megoldó motorokat értjük, amik a természetben tapasztalható szabályszerűség szerint járnak el. Ezek az algoritmusok már a 60-as évek óta léteznek, de sokáig nem tulajdonítottak nekik jelentőséget. Először Lowrence J. Fogel írt róluk "Az Értelem Szervezéséről" című publikációjában. A terület fontos alakjai még Ingo Rechenberg és John Henry Holland[2]. Egyik előnyük az exact solverekkel szemben, hogy rendkívül összetett problémákat is képesek lehetnek megoldani, több paraméter irányításával is megbirkóznak, viszont rendkívül időigényesek. Továbbá nem garantált az eredmény használatukkor. Erősségüket viszont bizonyítja, hogy egyre több szabadalom létezik, mely működésükhöz köthető, például sok nyomtatott áramkör kiosztást már ilyen algoritmusokkal oldották meg. Grasshopper 3D-n belül általában ők is egy fitness értéket tudnak figyelembe venni, ilyen például a Grasshopper 3D-be beépített Galapagos, de vannak, amik akár több fitness értéket is figyelembe vesznek, ilyen például az Octopus. Ez utóbbi azonban több optimalizációs problémát is felvet. Fontos tudni ugyanakkor az evolúciós eljárásokról, hogy többszöri futtatásra rendkívül kis eséllyel fogják ugyanazt az értéket visszaadni. A hatékony használatuk miatt lényeges az elvi működésük alapos ismerete. Megfelelő alkalmazással multifunkcionális eszközök lehetnek különböző kérdések megválaszolásában. Előnyük ezenfelül, hogy progresszív futásuk közben munkaközi állapotok is kimenthetők és ezek akár eredményként elkönyvelhetők. A rendkívül sok beállításnak köszönhetően finomhangolhatók és interaktívak. Galapagoson belül ezek például lehetnek a populáció száma, örökítési százalék, párzás értéke, a maximális futási idő, maximális generáció szám.

[2] (Domingos, 2015)

Működésük:[3]

Ebben a részben alapvetően a Galapagos működését mutatom be, de a rá érvényes szabályok nem sokban térnek el más evolúciós motorok esetében, például az Octopus esetében sem.
Maga a komponens két csatlakozási lehetőséggel bír, az egyik a fitness, vagyis az az érték, amit vagy maximalizálni vagy minimalizálni szeretnénk. A másik pedig az úgynevezett genoms (gének). Ezek azok a paraméterek, amiket a komponens irányíthat. Számuk szinte végtelen nagy lehet, ám minél nagyobb a számuk, annál inkább lassul a folyamat.

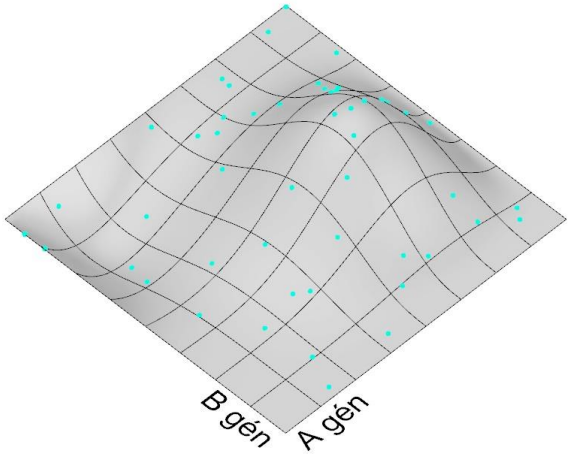
[6]



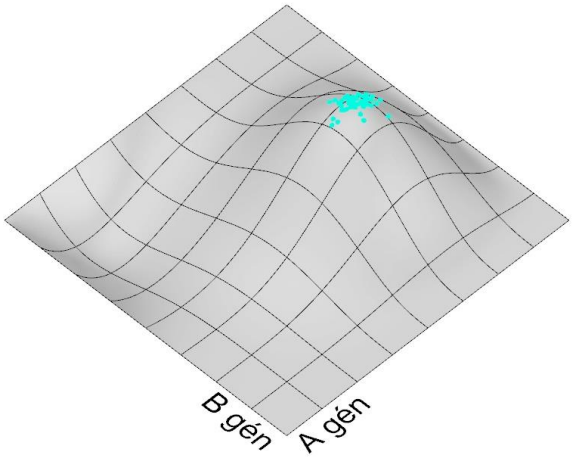
Attól függően, hogy hány Genomunk (változtatható érték) van, változik az úgy nevezett Fitness Landscape. 2 Genom esetén ez egy olyan négyzet alapú 3 dimenziós felület, aminek az egyik oldala megfeleltethető az egyik gén értéknek, a rá merőleges pedig a másiknak. Magának a „tájnak” a magas pontjai lehetnek lokális vagy globális maximumok, ugyanígy a völgyekkel, melyek lokális vagy globális minimumok, ez az n+1-dik kiterjedés valójában a Fitness [6]. Míg 2 gén esetén egyszerű ezt a Fitness Landscapet ábrázolni, 3 Génnél ez már 4 dimenziós.
Az algoritmus benépesíti a tájat különböző random értékkel. Tegyük fel, hogy az A és B gén értéke 0-tól 1-ig terjedhet. Ebben az esetben egy gén a következőképpen néz ki: {0.2;0.7}, mely megfeleltethető egy pontnak a tájunkon. Miután megvan a populációnk, ha maximumra törekszünk csupán ki kell írtani a legrosszabbul szereplő tagokat, melyek a völgyekben és domb alján tartózkodó egyedek. Majd a következő generációt létrehozni a legjobban szereplőkből, így közelítve a globális maximumhoz.

[3] (Rutten, 2011)

[7]



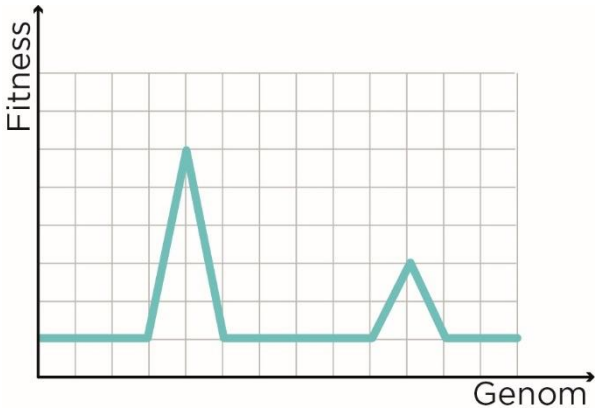
[8]



Fitness:

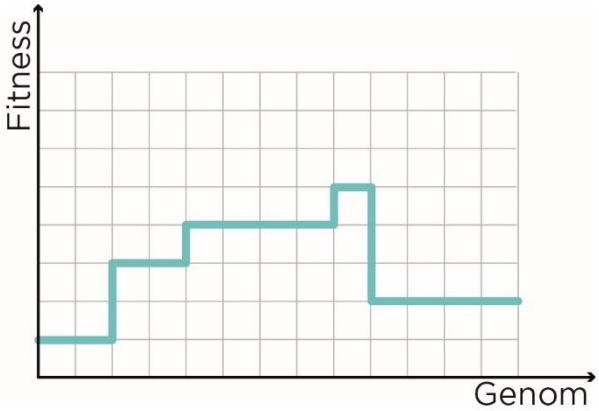
Ez az az érték, ami alapján az algoritmus el tudja dönteni, hogy jó irányba halad-e. A fentebb említett fitness tájunknak a 3. kiterjedését is ez az érték határozza meg. A gének feladata a dombok legmagasabb pontjának megtalálása. Ám ez a gén szempontjából olyan feladat, mint bekötött szemmel megmászni a Himaláját. Sosem lehet biztos benne, hogy egy adott ponton épp a maximumon van, vagy csak lokális maximumot talált és egy völggyel odébb rejtőzik-e az igazi csúcs. Minden probléma saját fitness tájjal rendelkezik, és ennek a tájnak a rajzolata determinálja mennyire effektív az algoritmus. Minden tájhoz más megközelítést érdemes alkalmazni.

[9]



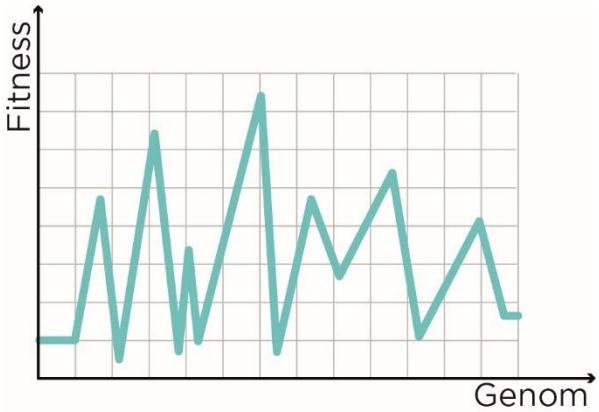
A fenti [9] példában nagy valószínűsége van, hogy az első generáció nagyon nehezen fog tudni kilépni a sík területről és minél több generáció telik el, annál jobban benépesítik a sík részt. Ám ha 1 tag megtalálja a csúcsot, gyorsan megnő az ő populációja és a többi kihal.

[10]



Ez [10] egy még rosszabb eset, mivel az algoritmusnak ötlete sincs, mit tegyen, csupán a szerencsén múlik, hogy megtalál-e egy fentebbi lépcsőfokot, de utána ugyanabba a szituációba kerül, hiába lép jobbra vagy balra, a fitness értéke nem változik. Ez tovább lassíthatja az eljárást.

[11]



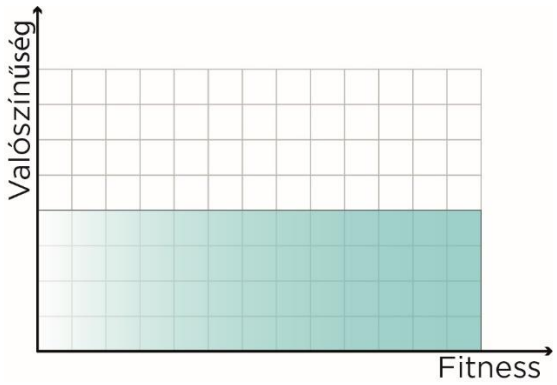
Legrosszabb eset [11], ha a tájunk túl zajossá válik. Ilyen esetekben szinte lehetetlen megállapítani a globális maximumot. Továbbá megtörténhet az, hogy két gén utódja egy másik völgybe kerül, mint a szülei, így nehezítve a konzisztens szaporodást és navigációt.

Szelekció

A populáció tagjainak párosíthatóságát a szelekció alapján választjuk ki. A Galapagoson belül erre 3 különböző lehetőség van.

Izotropikus Szelekció [12]

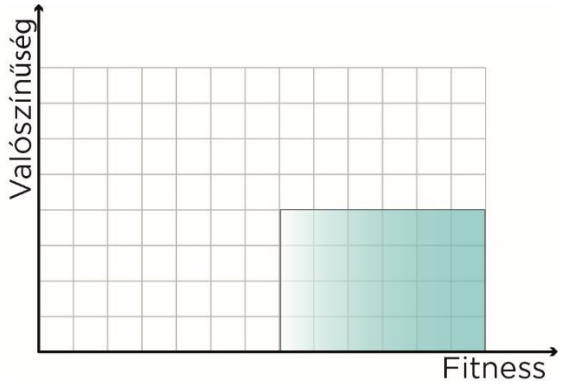
[12]



Ebben a forgatókönyvben mindenkinek van esélye a párzásra. Párzás alatt értjük azt a folyamatot amikor két egyed génállományát egyesíti, így létrehozva egy új egyedet. Elsőre értelmetlennek tűnik, de például felgyorsíthatja az algoritmust és el lehet vele kerülni lokális maximumok idő előtti benépesítését.

Exkluzív szelekció [13]

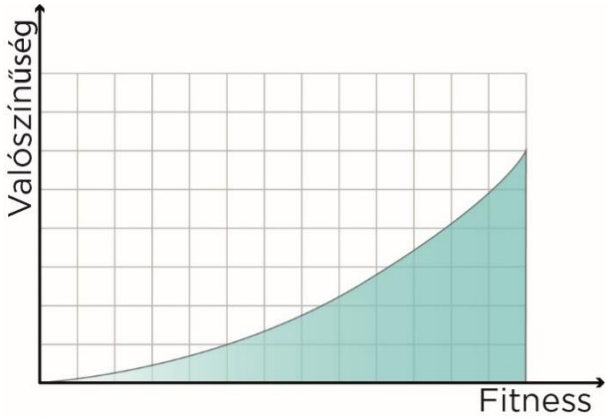
[13]



Ilyenkor csupán a populáció legfittebb részét engedjük pározni, ez viszont növeli a lokális zsákutcába kerülés esélyét.

Baised szelekció [14]

[14]

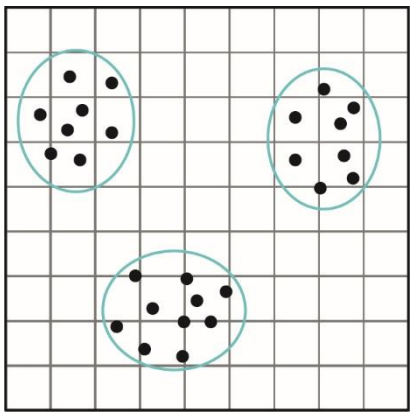


Ebben az esetben mindenkinek van esélye a párzásra, ám a fittebb egyedeknek nagyobb, mint a nem fitt társaiknak. Ezzel képesek vagyunk egy konzisztens előrehaladást biztosítani, közben csökkentjük a zsákutcában ragadás lehetőségét. Továbbá az evolúciós eljárásokban létezik az örök élet fogalma, így, ha egy tagnál nem tud az algoritmus jobbat pároztatni, az a tag addig maradhat életben, amíg ez nem sikerül az eljárásnak.

Párosítás

A párosítás az az eljárás, amikor a populációból kiválasztjuk, melyik 2 vagy több egyed genetikai állományát kombináljuk össze. Ezt tehetjük úgy, hogy egy adott egyedhez legközelebb elhelyezkedő egyedekkel párosítjuk, így a hozzá leghasonlóbbakkal. Ilyenkor viszont azt kockáztatjuk, hogy a populációnk egy lokális optimumon fog ragadni. Egy másik megközelítés az úgy nevezett zoophilic mating, ilyenkor kizárjuk az adott egyedhez túl közel álló egyedeket, és ezzel növeljük a genetikai sokszínűséget. Ezt az eljárást több egyedre kivetítve jól lehatárolható fajokat kapunk, ahol mindegyik populáció csoport a saját optimumát keresi és ezzel egy nagyobb szórásban vizsgáljuk az adott problémát. A legoptimálisabbnak tűnő megoldás, ha megpróbáljuk elkerülni a beltenyészetet, de kontrollban tartjuk a túlzott távoli párosodást is, így elkerülve a túl véletlenszerű zajos eredményeket. A Galapagoson belül ezt százalékos értékkel szabályozhatjuk, ahol
-100% a totális távoli párosodás, +100% a teljes beltenyészet.

[15]



Zoophilic mating

Egyesülés

Ha megtaláltunk két megfelelő értéket a párosításra, már csak azt kell eldöntenünk, milyen módszerrel vonjuk össze a genetikai állományukat. Az igazság az, hogy ez a számítógépes környezetben valójában nehezebb, mint a valóságban. Számítógépek világában gén alatt egy racionális számot értünk, melynek van maximum és minimum értéke. Egy egyed abból rengeteget tárolhat magában, attól függően, hogy hány Genomra, változtatható paraméterre van kihatása az algoritmusnak. A párosítást a következő módokon oldhatjuk meg:

Keresztezés:

Ilyenkor az új egyed fele arányban megkapja az anya génállományát és a másik felét pedig apjától örököli.

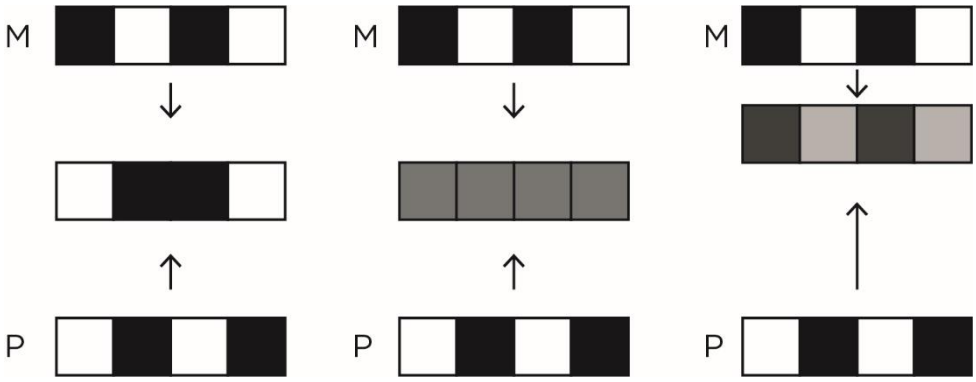
Átlagolás:

Ilyenkor az új egyed a szülői génállomány értékeinek az átlagát kapja.

Preferált-átlagolás:

Ebben az esetben van a legtöbb lehetőségünk a kontrollra, mivel ilyenkor eldönthető, melyik szülőnek a génjei legyenek a dominánsabbak és ennek megfelelően kapjuk az új egyedet, a két szülő génjeinek súlyozott átlagából.

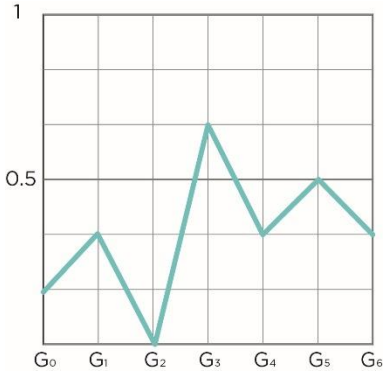
[16]



Mutáció:

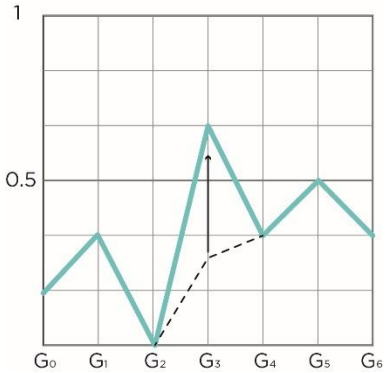
Az eddig beszélt eljárások mind csökkentik a biológiai sokféleséget. A legjobb fegyverünk ez ellen a mutáció. Ha felveszünk egy koordináta rendszert, ahol az x tengely a különböző géneket, az y tengely pedig a fitness értéküket mutatja, egyszerűen ábrázolhatunk multidimenzionális pontokat egy kétdimenziós térben. Erről könnyen leolvasható az egyes gének megfelelősége [16].

[16]



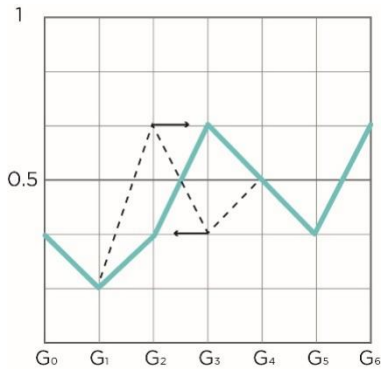
Az egyik legelterjedtebb mutáció a Pont mutáció [17], ilyenkor az egyik gén értékét változtatva vagyunk kihatással a populációra.

[17]



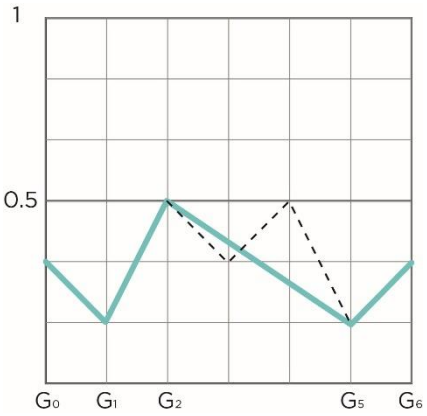
Az inverziós mutáció [18] egy másik lehetőségünk, ilyenkor két szoros kapcsolatban lévő gén cserél értéket. Ám ezt általában csak specifikus esetekben érdemes használni, mivel drasztikusan változtatja a fitness értékünket. Ez sokszor hátrányos lehet.

[18]

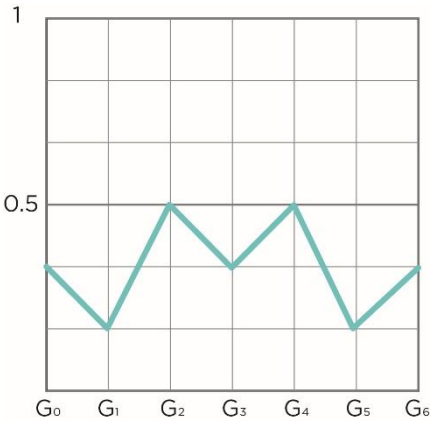


Továbbá létezik gén hozzáadására és elvételére is lehetőség [19], ezt viszont csak fix számú populáció esetén tehetjük meg.

[19]



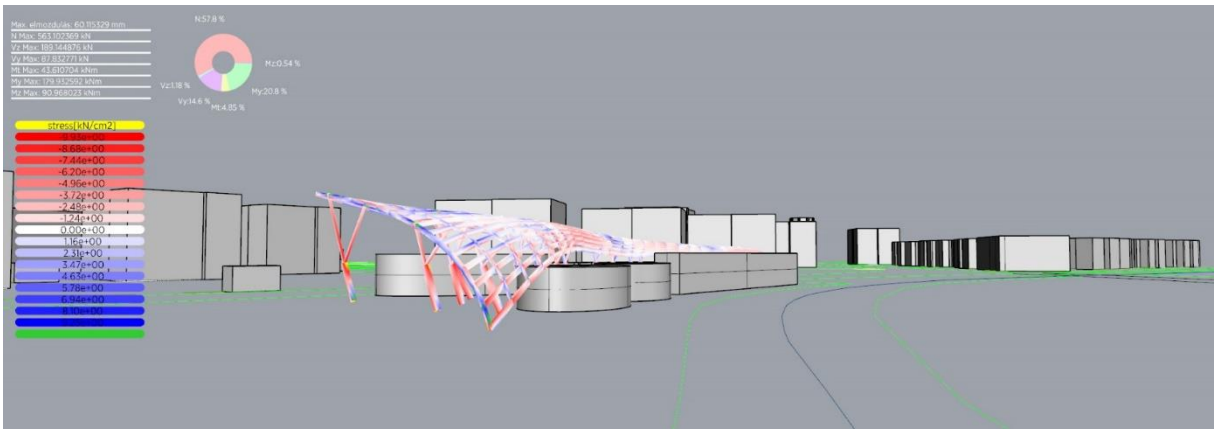
[20]



A script működése:

Az alábbi kép az algoritmus „kimenetét” (szöveg, érték és 3D-s geometria) mutatja a 3D-s modellező eszközben [21]:

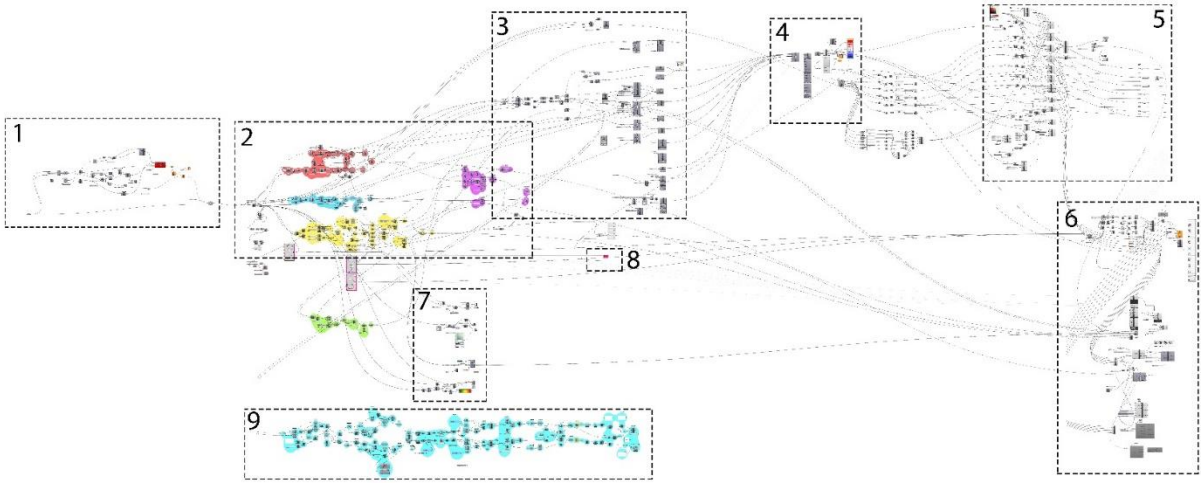
[21]



Grasshopper 3D-n belül különböző modulok közötti kapcsolatokat tudunk definiálni, minden modul rendelkezik egy vagy több kimeneti vagy bemeneti lehetőséggel, ettől csak kevés modul tér el.

Ezen a képen pedig maga az algoritmus [22]:

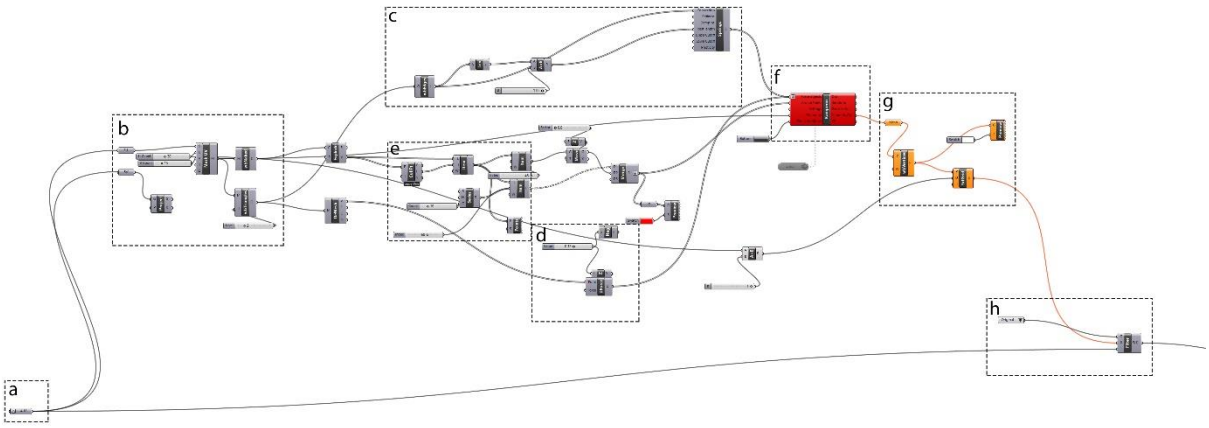
[22]



A különböző csoportok a következőkért felelősek:

1. Tető alapgeometriájának formaoptimalizálása

[23]



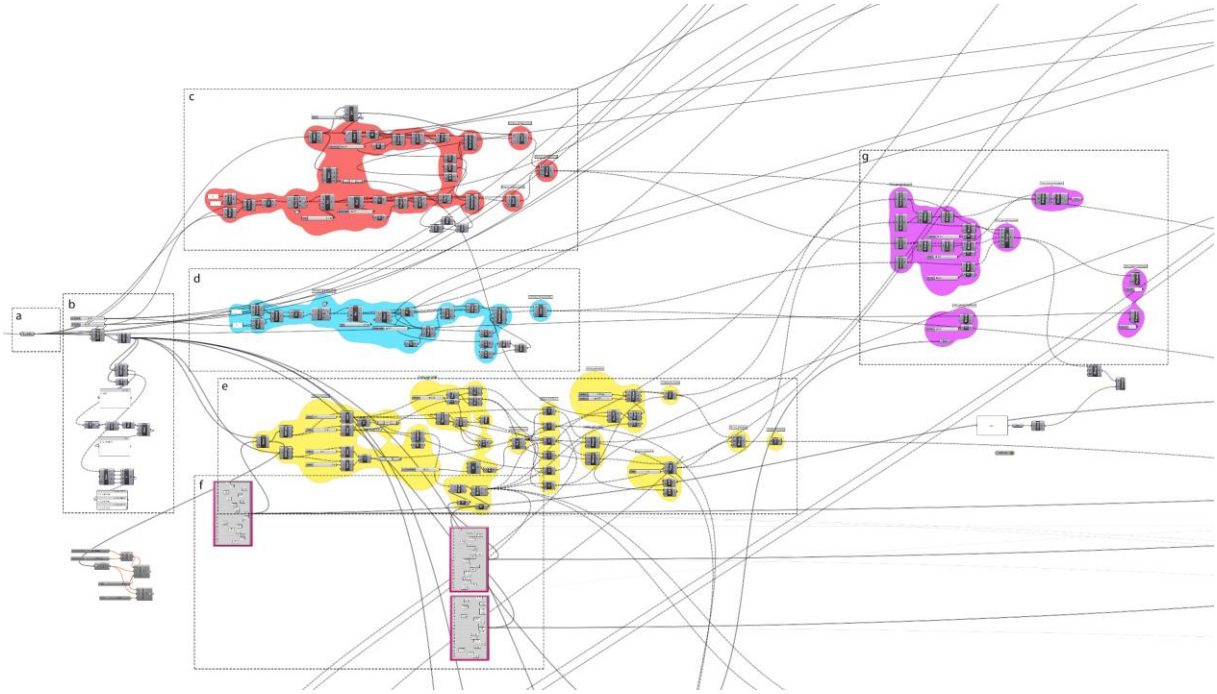
Az [a] pontban tudjuk definiálni az általunk, a Rhinoban megrajzolt felületet. A kód ekkor két opcióra bomlik, az egyik ágon a felület módosítása nélkül tovább küldhetjük a geometriát, vagy a másik ágon forma optimalizációt hajthatunk rajta végre a Kangaroo* segítségével. Ekkor a [b] pontban a NURBS felületből poligonhálót képzünk, mivel ez a fajta geometria szükséges a Kangaroonak. Majd a [c] pontban képesek vagyunk szabályozni, hogy mekkora megnyúlást engedélyezünk a felületnek, a [d] pontban a felületre ható erőket, az [e] pontban pedig a fix, nem elmozduló pontokat. A Kangaroo motorja valójában egy modul, az [f] pontban található.

* A Kangaroo kiegészítőről bővebben a Forma optimalizálás fejezetben lesz szó.

A [g] pontra azért van szükség, hogy a poligonhálót újra NURBS felületté konvertáljuk, mivel tető geometriáját ilyen formában jobban tudjuk kezelni. A [h] pont pedig ahol eldönthetjük, hogy melyik ágat küldjük tovább az algoritmusban, a forma optimalizáltat, vagy az általunk szerkesztettet.

2. Könnyen kezelhető, kis polygon számú modell, exportálásra pl.: Archicad-be

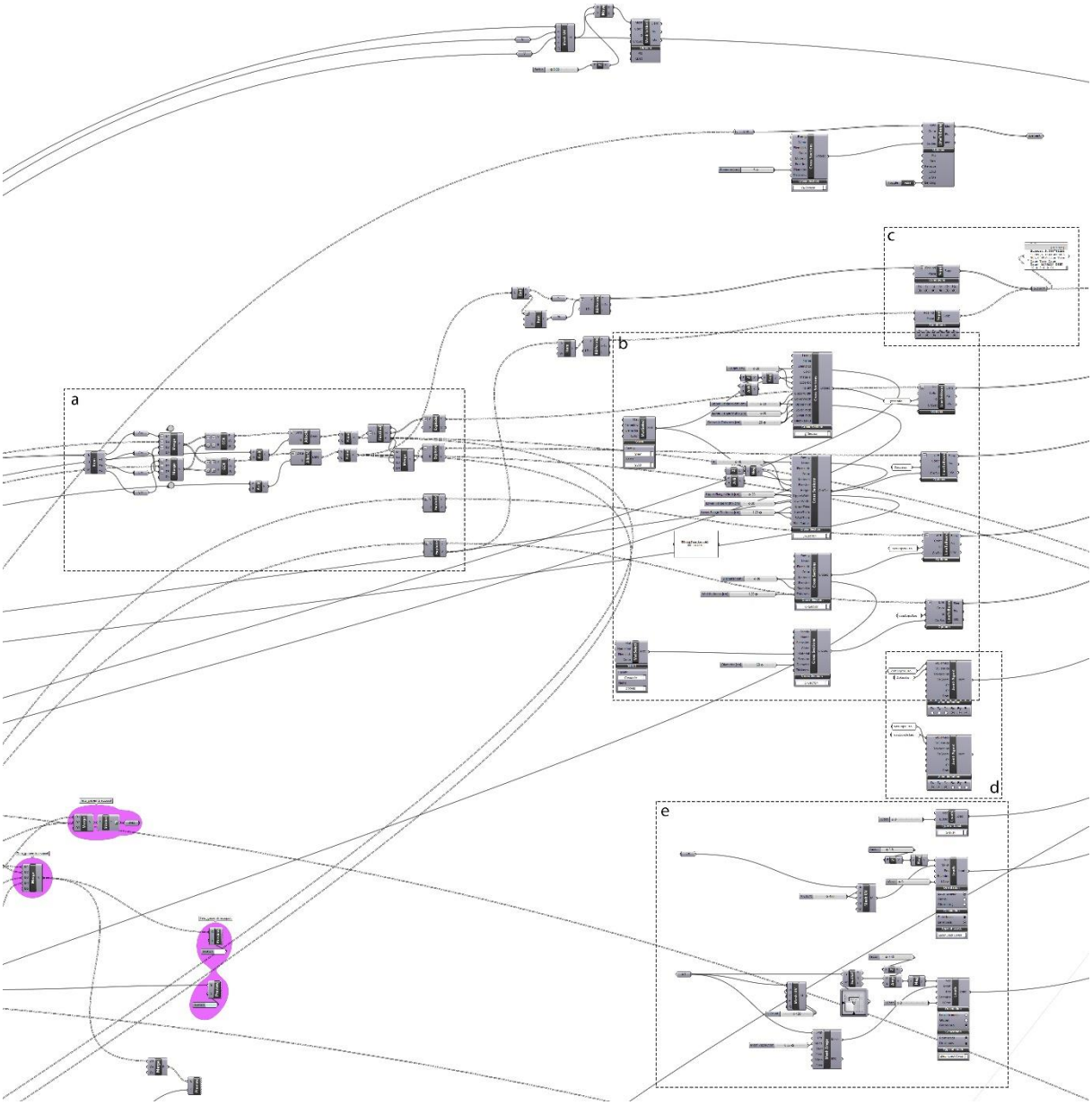
[24]



Az [a] pontban kapjuk meg a továbbított geometriát. A Grasshopper 3D-ben sokszor egyszerű lépések is rendkívül sok modult igényelhetnek, köszönhetően annak, hogy legelemibb geometriai lépesekre vannak lebontva. Egy sík meghatározásához is legalább három modul kell, egy origó pont, egy normál vektor és egy modul, ami létrehozza a síkot. Továbbá egy „fonalon”, ami a modulokat összeköti, több fa struktúrába rendezett adat is folyhat, így a modulok számát az ilyen fajta adatkezelés is megnöveli. Ennek köszönhető, hogy olyan egyszerűnek mondható geometria, mint a főtartók [c] és melléktartók [d] sok helyet foglalhat. Ezért is szükséges a [b] halmaz, ahol csupán csak az adatstruktúrát ellenőrizzük. Az [e] pontban állítjuk elő az oszlopok geometriáját az [f] pontban található paraméterek alapján, ezek azok a paraméterek, amikre később az evolúciós algoritmusnak kihatása lesz, továbbá a szelvények méretét a végeselem modellnél megadott szelvényméretek befolyásolják. A [g] pontban csupán összesítjük a geometriákat, ha esetleg exportálni szeretnénk más programokba.

3. Szerkezeti elemek definiálása, anyagminőség, szelvényméretek, kapcsolatok

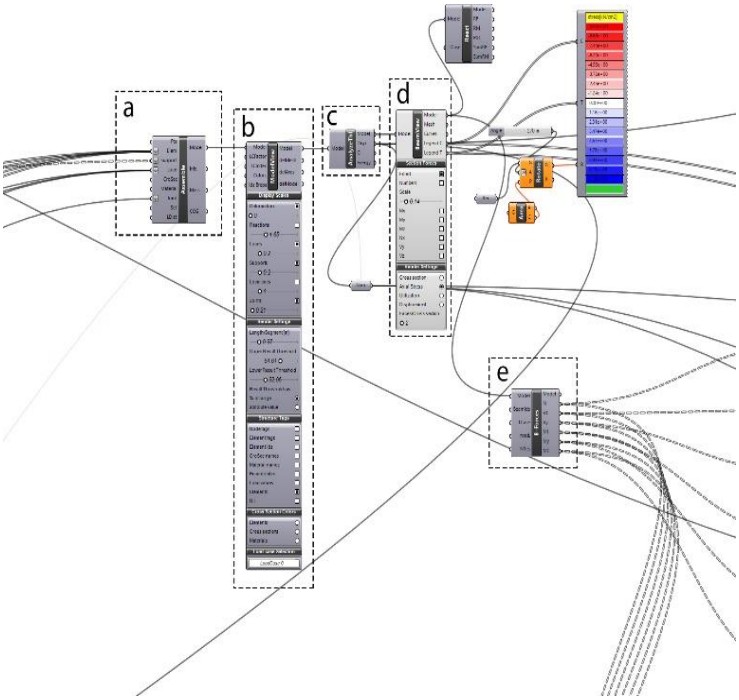
[25]



Az [a] pontban a NURBS geometriát újra polygonhálóvá alakítjuk a végeselem modell felállításához. A modell átalakítása az előző 2-es pontban található alapfelületből történik, így valójában kettő modellt kapunk, egy a végeselem motornak megfelelőt és egy a végeselem paraméterei alapján képzett, részletesebb modellt. A [b] pontban vagyunk képesek a különböző anyagminőségeket, geometria méreteket beállítani. A [c] pontban tudjuk a támaszok helyét és típusát beállítani. A [d] pontban állíthatjuk a kapcsolatok típusait. Az [e] pontban pedig képesek vagyunk definiálni a tetőt érő erőket.

4. VEM modell kezelőfelület

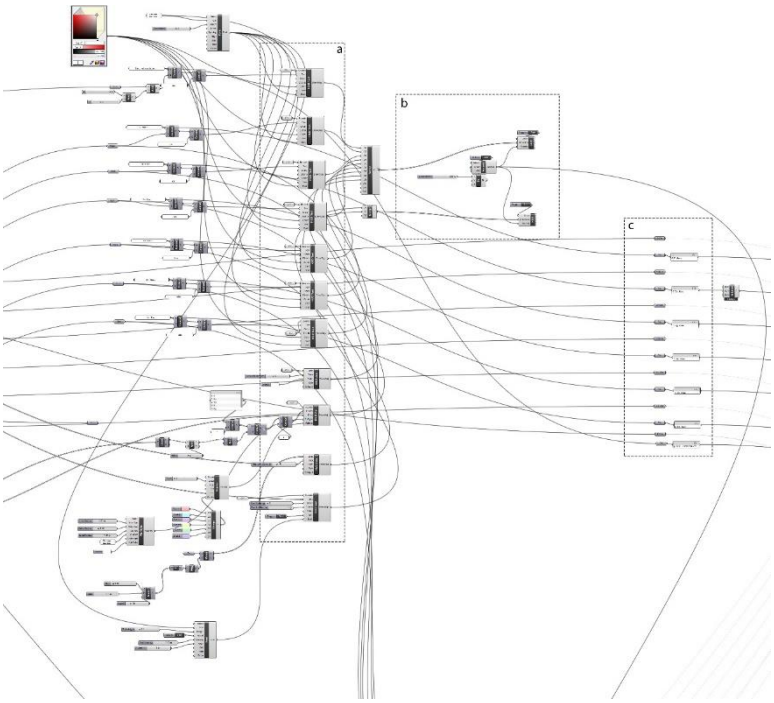
[26]



Miután felállítottuk a modellt, az [a] pontban összegük elemeit, a [b] pontban felállítunk egy vizuális reprezentációt. Majd a [c] pontban elvégezzük az analízist, a [d] komponens a kiértékeléshez szükséges. Így tudjuk megjeleníteni a belső erő ábrákat (M_x, M_y, M_z, N_x, V_y, V_z), a szerkezet kihasználtságát és a maximális elmozdulás eredményét, amit későbbiekben fitness értékként fogunk definiálni az evolúciós motornak. Az [e] pontra az értékek szétbontása miatt van szükség.

5. Felhasználói felületen látható adatok megjelenítése definiálása

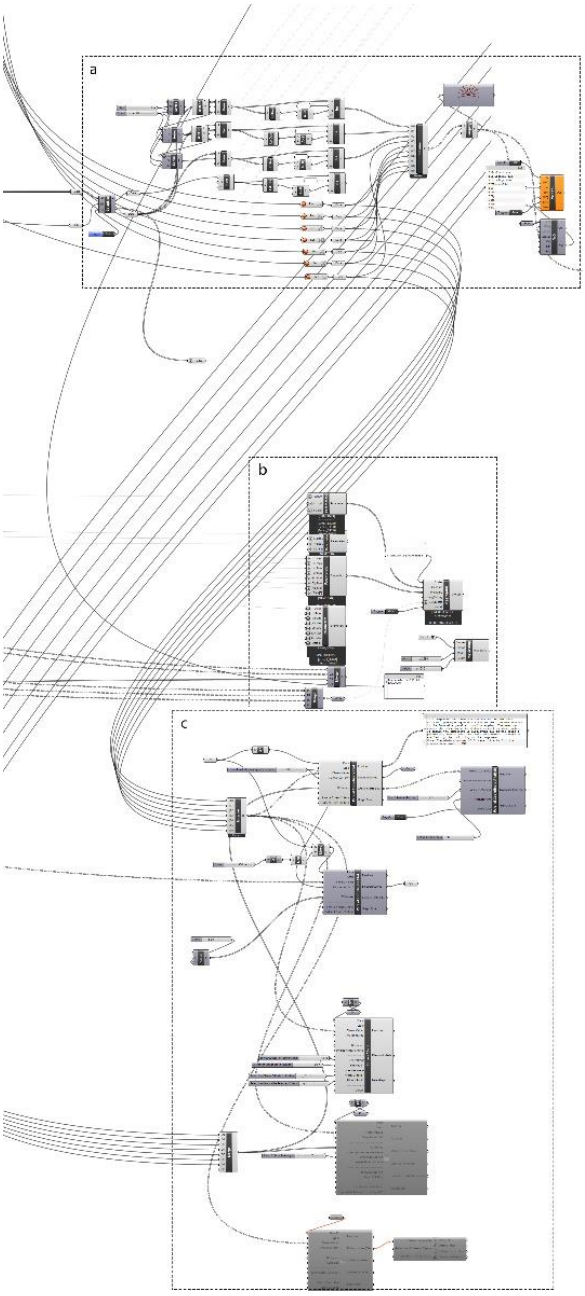
[27]



Az [a] pontban tudjuk definiálni a különböző értékek kinézetét, azaz, hogy szeretnénk megjeleníteni a felhasználói felületen. A [b] pontban tudjuk a felhasználói felületre vetített értékeket ki- és bekapcsolni. A [c] pontban pedig összegezzük ezeket az értékeket, hogy későbbiekben azokat tárolni, értékelni tudjuk.

6. Adatok külső exportálása, pillanatképek készítése, grafikus kimutatások, Excel táblába való exportálás

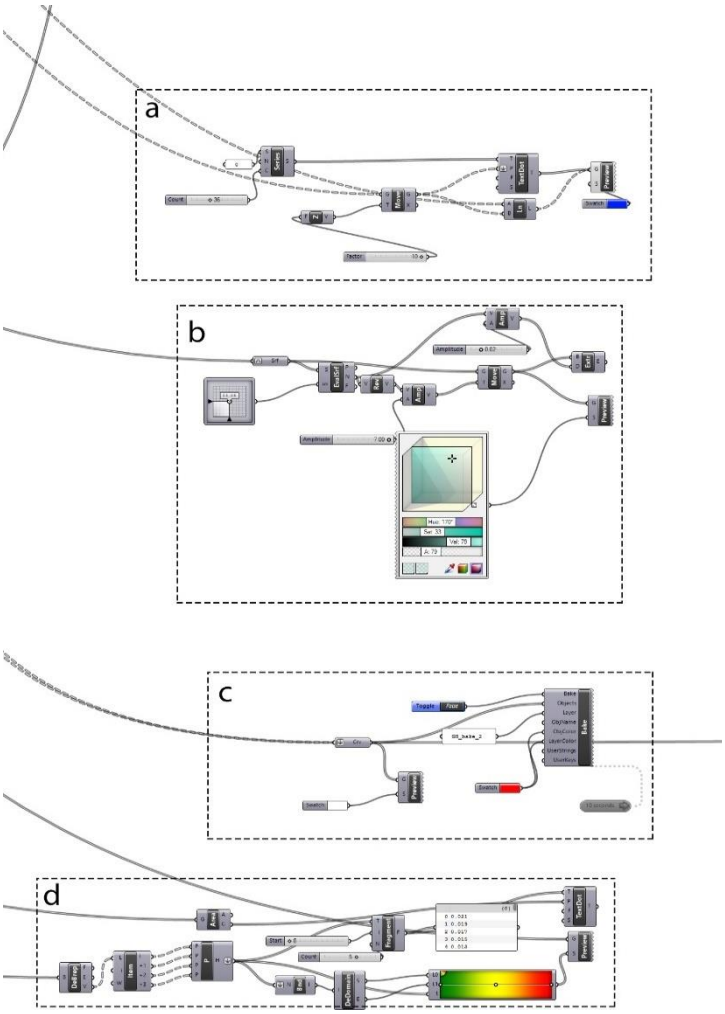
[28]



Az [a] pontban történik az Excelbe való exportálás, és az adatok rögzítése. A [b] pont feladata minden generációról pillanatfelvétel készítése és annak a felhőmappába való mentése, későbbi értékelésére. A [c] pontban pedig az adatok alapján különböző grafikonokat vagyunk képesek kinyerni.

7. Ellenőrző modulok, konszignáció, tetőpanelek sík ellenőrzése

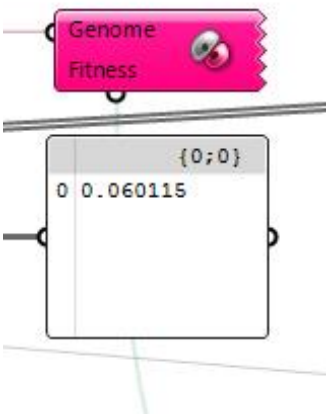
[29]



Az [a] pont felelős az oszlopok konszignálásáért a felhasználói felületen. A [b] pont a tetőt fedő panelek megjelenítéséért felel. A [c] pontban történik az összes kipróbált oszlop helyének rögzítése. A [d] pont pedig a tető panelek görbületének nagyságát vizsgálja.

8. Evolúciós motor

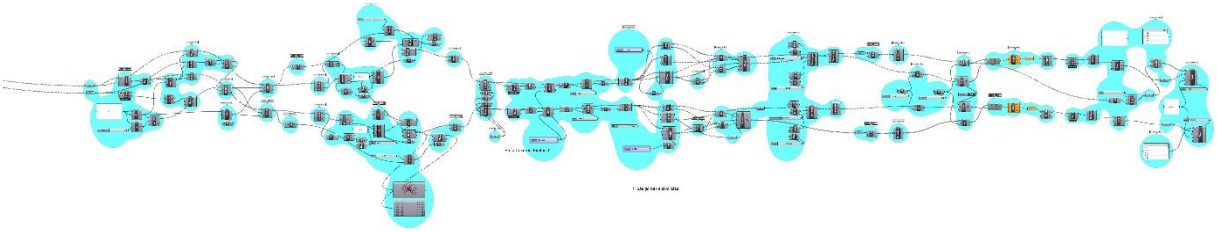
[30]



Az evolúciós motor valójában csak egy modulból áll. Meg kell viszont neki határozni, hogy mi legyen a fitness értéke, és mely paramétereket módosíthatja. További beállításokat a modul megnyitása után kapunk egy felhasználói felületen.

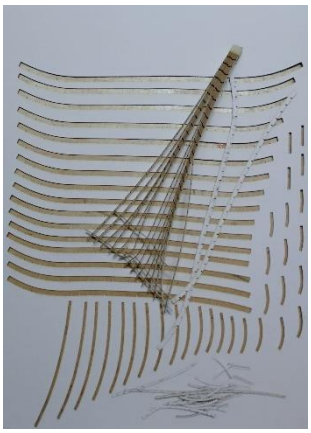
9. Modell kiterített képe, lézervágáshoz előkészítése.

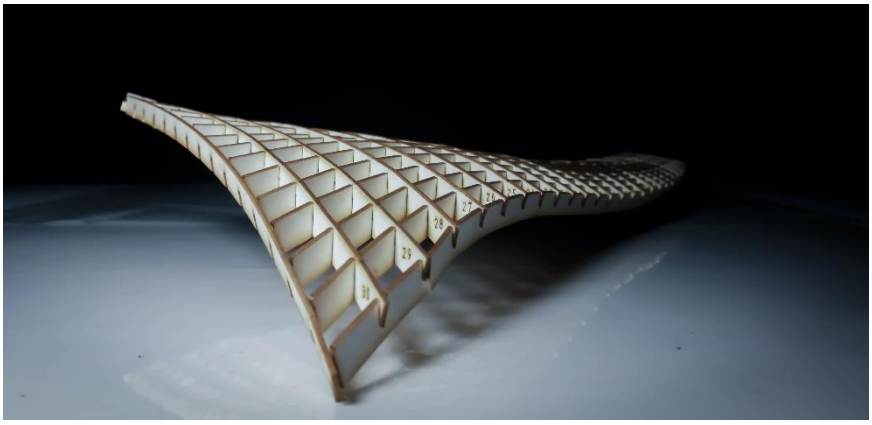
[31]



A lézervágáshoz a tartószerkezeti elemeket kiterítettem, és az elemeket lapra rendezve konszignáltam.

[32]





[33]

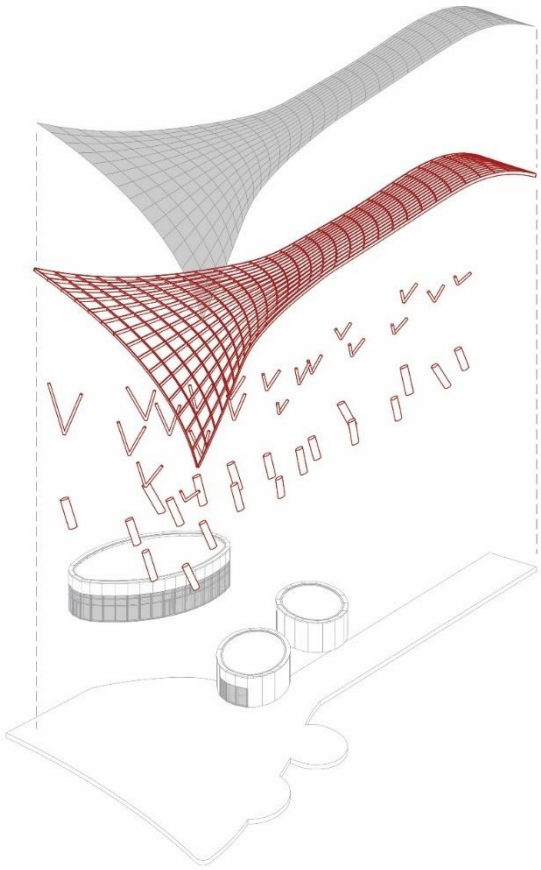
A fenti felsorolásból jól látható, hogy egy jól felépített algoritmus segítségével rengeteg részfeladatot képesek vagyunk lefedni, kezelni. Akár egy sor klasszikus program kód írása nélkül is képesek vagyunk bonyolult kapcsolatokat definiálni. Az algoritmus a sok konvertálás miatt nem a legoptimálisabban fut, sok számítási kapacitást igényelhetnek ezek az eljárások. A geometria kezelhetőségének szempontjából egyszerűbb a NURBS felületeket használni, míg szimulációkhoz a poligonhálók az optimálisak, ezeknél viszont számításba kell venni a felület felbontási minőségét, részletességét is.

A tetőszerkezet fitness landscape-e a következők szerint tevődik össze: 20 oszlopnak keressük a helyét, ez egy általunk meghatározott szám. Ez a 20 oszlop a 15 · 30 részre osztott tetőfelületen 450 lehetséges helyen lehet. Továbbá minden oszlop dőlését x és y tengelyen pozitív vagy negatív eltolásával szabályozzuk -2.00 és +2.00 érték tartományban. Ez körülbelül $9{,}75 \cdot 10^{52}$ lehetséges állás. Így a fitness landscape-ünk az adott algoritmusban 61 dimenziós, mivel 60 változtatható értékünk van és plusz egy maga a fitness. Egy ekkora génállománynál a fitness landscape rendkívül zajos lehet, sok lokális minimummal.

Az evolúciós eljárást ebben a példában a következők szerint kell elképzelni. Ha például 50 fős populációt állítunk be akkor a program választ véletlenszerűen a 60 állítható érték közül 60-at mivel minden oszlop három állítható értékkel rendelkezik, (egy az oszlop helye, egy az x irányú eltolása egy pedig az y irányú eltolása) és összesen 20 oszlopunk van (3 · 20). Ezt ismétli meg

50-szer, majd a legjobban teljesítő egyedeket, akiknek a legkisebb a fitness értéke, tovább pároztatja, azaz azokat, akik hatására a legkisebb lehajlás keletkezik a tetőben. A pároztatás pedig a megfelelő finomhangoló paraméterek szerint alakul, mint az öröklési százalék, pároztatás típusa.

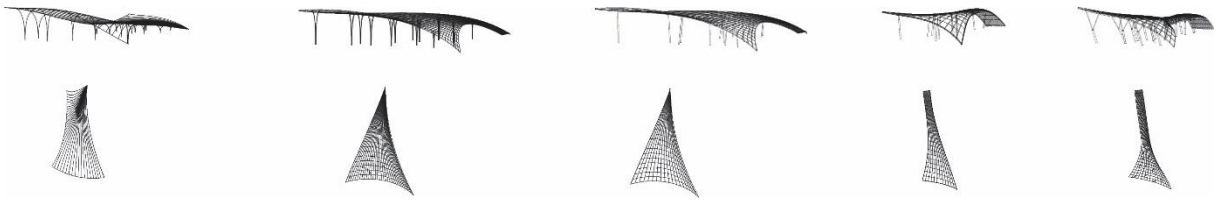
Evolúciós eljárások a gyakorlatban



[34]

Miután megismertük magának az algoritmusnak a működését, biztosabban alkalmazzuk a gyakorlatban. Az könnyen belátható, hogy egy evolúción alapuló megoldó motor sokféle szituációban alkalmazható. Grasshopper 3D-n belül lehetőségünk nyílik akár benapozási, épületszerkezeti, tartószerkezeti vizsgálatokra is. Ezeknek az eredményeit pedig fitness kondíciónak használhatjuk és az evolúciós motorra bízhatjuk az optimum megtalálást. Így irreguláris, ám jó megoldásokat is találunk. Jelen esetben szerkezettervezésére használtam a komplex tervem keretén belül.
Maga a felület kétszer görbült, a peron felőli oldalon vonalmentén, a másik végén egy pontban ér le a földhöz. A tervhez készült algoritmus a felületből képezi le a tetőszerkezetet. Az egyetlen bemeneti geometria az általam létrehozott kétszer görbült felület. Ennek a felületnek további bontásából részfelületek jönnek létre, ezek közel sík geometriák, ezt az algoritmus ellenőrzi. Ezekből alakulnak ki az akrillal fedett mezők. A négyzetes felületek egyik oldali, és azzal párhuzamos alkotóiból alakulnak ki a szerkezet főtartói és erre merőleges alkotókból a melléktartók. Azt, hogy a felületet az U és V oldala mentén hanyadrészére daraboljuk, szabadon állítható, így szabályozható a tartószerkezetek sűrűsége. A félév során több verzió is készült a tető geometriájára, ezek a változások mind a funkcióból és tartószerkezetei szempontból következtek be. Mindegyik geometria értékelésre került a Karamba 3D segítségével, törekedve az optimum megtalálására.

Ekkor még az algoritmus nem tartalmazta a forma optimalizációs részt. A különböző verziók geometriája az alábbi képen látható [35].



[35]

A geometriából fakadóan esett a választás fémszerkezetre, annak könnyebb megmunkálhatósága és szilárdsági tulajdonságai miatt. Fa tartóként is realizálható a szerkezet, ám sok egyéb problémát vet fel, mint a például a csomópontok kialakításának nehézségei és a kúszás jelensége. Az oszlopok egy négyzet felület átlós pontjait támasztják alá és változó szögekben állnak, így képesek felvenni több irányú terhelést is és merevítik a szerkezetet. Az oszlopok alsó három és fél méteres vasbeton technológiával készül elliptikus formában, majd annak a tetején készül el a fogadó szerkezet, ahonnan két irányba elindulnak a tetőt támasztó körszelvények. A tetőszerkezet hierarchikus szerkezetként tevődik össze. Oszlop-főtartó-melléktartó-fedés kapcsolati sorrendben. A fő- és melléktartók zártszelvényből készülnek, geometriájukat a Karamba 3D nevű végeselem program segítségével határoztam meg. A program lehetőséget nyújt a szelvények méretezésére is. Képes szelvénytáblázatok adatai alapján a szerkezet igénybevételeit és a szerkezet alakváltozását figyelembevéve szelvényt választani. Az oszlopok számát én határoztam meg, ám elhelyezkedésüket és dőlésszögüket már a Galapagos nevű evolúciós motor segítségével jelöltem ki. Az evolúciós eljárás a szerkezet végeselem modelljéből számított elmozdulást figyelte a szerkezetet érő terhek alapján, melyek a következők voltak: önsúly, szélteher, hóteher. Az oszlopok helyének és dőlésszögének változtatásával próbálta elérni azok minimumát.

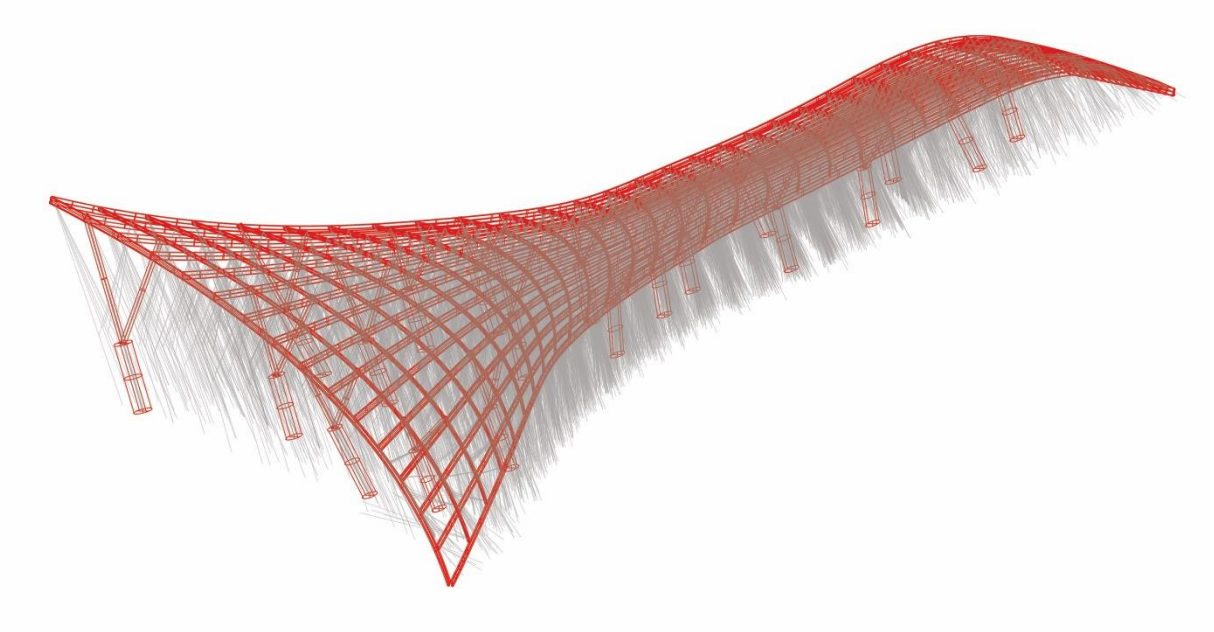
Az első félév során futtatott eljárás egy délutánon keresztül futott, 50 fős populációval és a 100. generációig hagytam futni, mivel ott már csak ezred milliméteres változások következtek.

A szelvény méretek a következők szerint alakultak:

- Főtartó: S355, 400*200*12 hidegen hengerelt zártszelvény Königfrankstahl EN 10219
- Melléktartó: S355, 200*200*12 hidegen hengerelt zártszelvény Königfrankstahl EN 10219
- Oszlop körszelvények: S355, Φ300*12 cm falvastagságú hidegen hengerelt zártszelvény Königfrankstahl EN 10219

- Elliptikus beton oszlopok C50-es betonminőséggel, hossztengelye: 65 cm, rövidebb tengelye: 25 cm

Az alábbi képen látható az összes opció, amit az algoritmus kipróbált az oszlopok helyének és dőlésszögének meghatározására [36]:



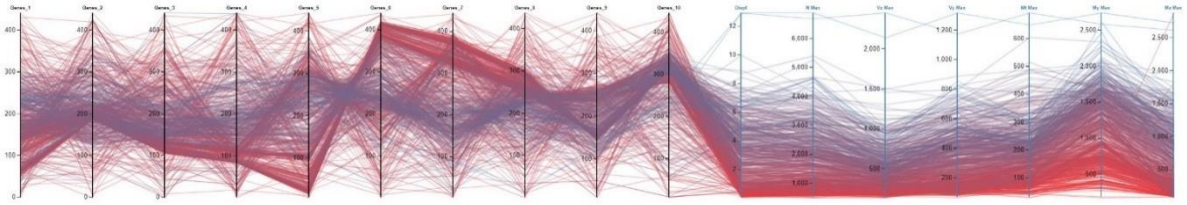
[36]

Itt a Galapagos beállításai a következők voltak:

Egy generációra vett populáció:	50 fő
Kezdő populáció kezdő szorzója:	2
Generációk közötti genom átviteli százalék:	2%
Párosítás százaléka:	+75%

[37]

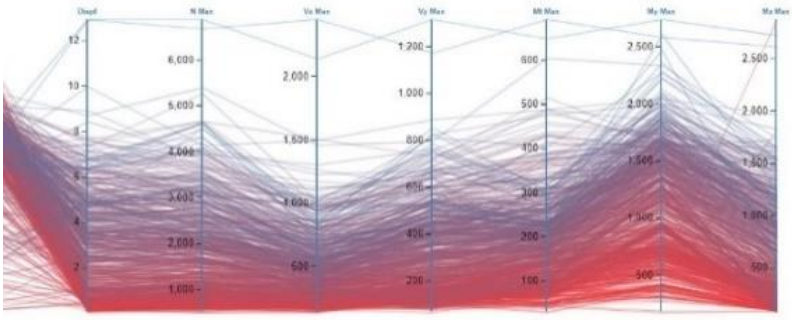
Ezeken a beállításokon, az evolúciós eljárás futása után a következő értékeket kapjuk a beállítások módosítása nélkül. Ez az eljárás is az oszlopok helyének változásával kereste a szerkezet minimális lehajlását. Az algoritmus futása 2 és fél óra volt, új optimumot viszont az előző 100 generációs eljáráshoz képest nem volt képes elérni, így csak 20 generációig futott [37]. A legjobb fitness érték továbbra is a 0,06 m-es lehajlás maradt. Az evolúciós eljárások egyik előnye, hogy a munkaközi állapotok is lehetnek érdekesek, akár végeredményként is felfoghatók. Ezeknek az eredményeknek a kiértékelését segíti a Colibri nevű kiegészítő, mellyel képesek vagyunk minden egyes genomot rögzíteni és azt későbbiekben szűrésekkel és értékeléssel ellátni. Az előző 20 generációs eljárás a következőképpen alakult [38]:



[38]

Az ábrán egy vonal reprezentál egy egyedet a populációból, minél vörösebb a vonal, annál frisseb, annál inkább az eljárás vége felé keletkezett. Fekete tengelyen láthatjuk a gének értékeit, azaz az adott oszlopot hol próbálta elhelyezni az algoritmus, milyen kombinációban. Kék tengellyel jelölt értékek pedig sorban a következők:

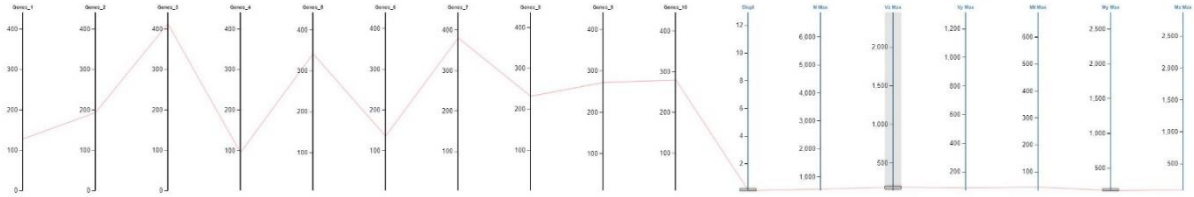
- Elmozdulás
- N max
- Vz max
- Vy max
- Mt max
- My max
- Mz max



[39]

Amennyiben szűrést állítunk be arra vonatkozóan, hogy mekkora tartományban szeretnénk látni az elmozdulás, Vz max és My max értékét, meg is kaphatjuk a legmegfelelőbb egyedet a populációból.

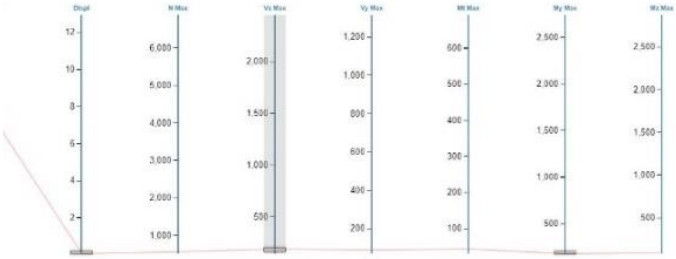
Ez az egyed pedig a következő [40-41]:



[40]

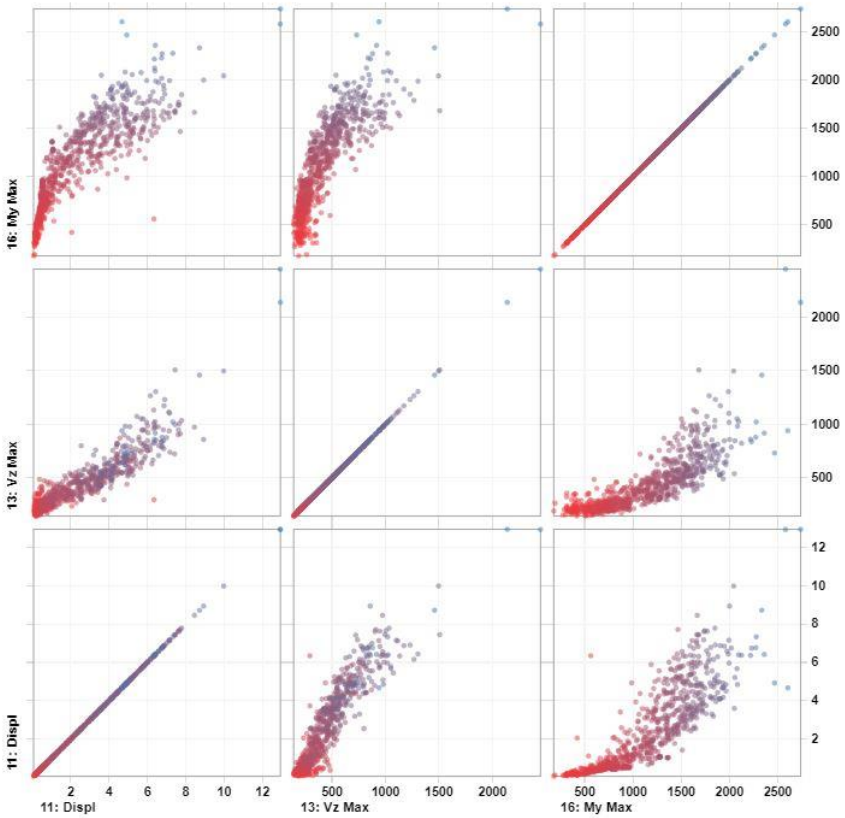
m / kN / kNm

Elmozdulás	0,0601 m
N max	563,10 kN
Vz max	189,14 kN
Vy max	87,83 kN
Mt max	43,61 kNm
My max	179,93 kNm
Mz max	90,96 kNm



[41]

Az értékek szórását is vizsgálhatjuk [42]:



[42]

Ezen a képen láthatjuk az eljárás szórását elmozdulás, Vz max és My max szempontjából.

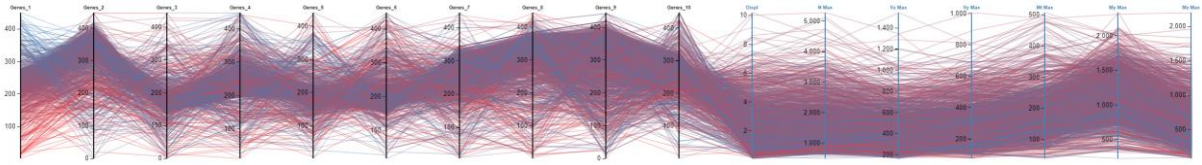
Az eljárás online adatbázisa ezen a linken elérhető: https://goo.gl/kW6G2F

Optimalizációs tesztek

Ezek után a szerkezet véletlenszerű újra rendezése vált szükségessé újabb tesztek elvégzése végett, így egy friss, lehetőleg egy lokális minimumtól távol tudjuk újrakezdeni a tervezést.
Mivel az eljárás során rendkívül sok gén szabályozását bízzuk az algoritmusra, érdemes lehet a populáció számát alapértelmezett 50-ről 100-ra növelni, így egy adott generációban sokkal nagyobb szórásból kapjuk meg a később tovább örökítendő gének értékét, ám lassítjuk magát a folyamatot. Ennek megfelelően a következő beállításokkal indítottam el az eljárást:

Egy generációra vett populáció:	100 fő
Kezdő populáció kezdő szorzója:	1
Generációk közötti genom átviteli százalék:	5%
Párosítás százaléka:	+40%

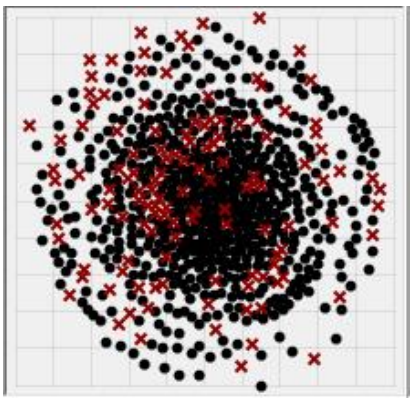
A párosítást kis mértékben eltoltam a zootophilic irányba, megpróbálva ezzel tovább csökkenteni a lokális minimumban maradás kockázatát.
Az algoritmus 5 és fél órán keresztül futott, a 22. generációt érte el, de nem sikerült elérni új lokális minimumot.



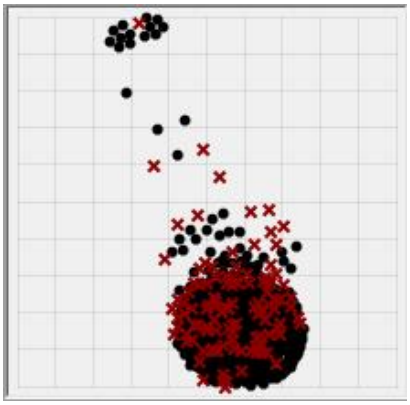
[43]

Ez köszönhető annak is, hogy ilyen pároztatási százaléknál lekorlátozódhat a minimumok közötti vándorlás, mivel az algoritmus se kimozdulni, se ott maradni nem akar egy állapotból.

[44]



Az eljárás elején vett populáció



Az eljárás végén vett populáció

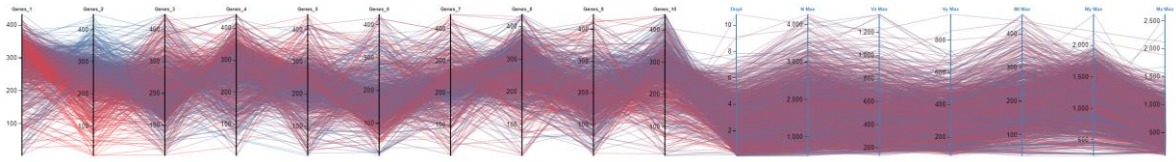
[45]

A teljes kimutatás elérhető ezen a linken: https://goo.gl/PwA2BK

Ezek után a negatívját vettem a párosításnak, majd a véletlenszerű újrarendezés után a következő beállításokkal indítottam az eljárást:

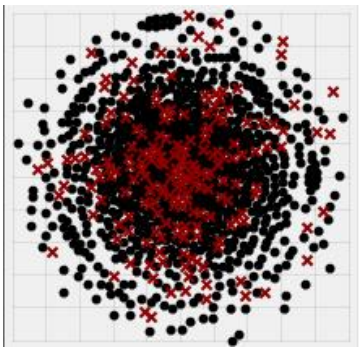
Egy generációra vett populáció:	100 fő
Kezdő populáció kezdő szorzója:	1
Generációk közötti genom átviteli százalék:	5%
Párosítás százaléka:	-40%

Ilyenkor már az algoritmus a vártaknak megfelelően viselkedik, sok lokális minimumra rátalál, közelít a globális minimumhoz, de mivel a párosítási érték továbbra is viszonylag közel áll 0%-hoz, nem képes nagyobb lokális minimumot megtalálni.
Ez az eljárás 14 órán keresztül futott, a 31. generációt érte el. [46]

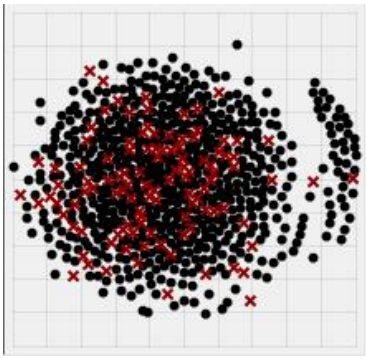


A populációs eloszlások pedig a következők az eljárás elején és végén [47-48]:

[47]



[48]

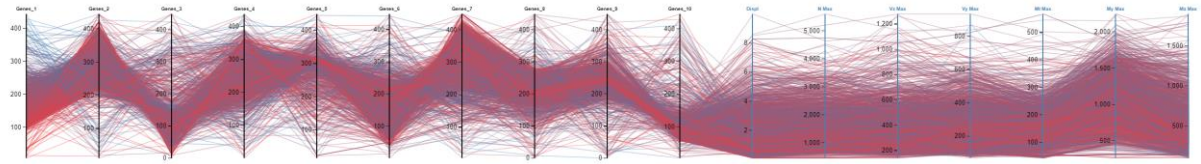


A képen jól látható, hogy sokkal nagyobb a szórásunk, és az algoritmus főleg a populáció közepén levő tagokat törli. Részben már megfigyelhető a zootophilic-féle viselkedés.
A teljes kimutatás elérhető ezen a linken: https://goo.gl/ojbZ5p.

Ezek után a szerkezet újra rendezését követően egy valós faj szerinti pároztatást futtattam, alacsonyabb populációval, a következő beállítások szerint:

Egy generációra vett populáció:	50 fő
Kezdő populáció kezdő szorzója:	2
Generációk közötti genom átviteli százalék:	5%
Párosítás százaléka:	-75%

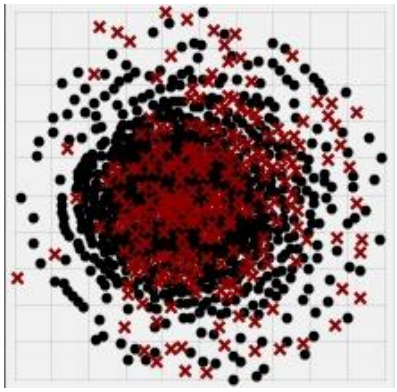
Az algoritmus szintén sok lokális optimumot megtalált, és nagyobb hatékonysággal, mint elődje, ám a várható globális minimumot továbbra se sikerült neki megtalálnia, csupán csak megközelítenie. Az eljárás 7 órát futott és az 50. generációig jutott.

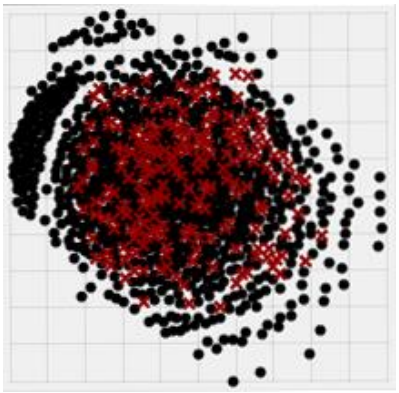


[49]

A populációs eloszlások pedig a következők az eljárás elején és végén [50-51]:

[50]





[51]

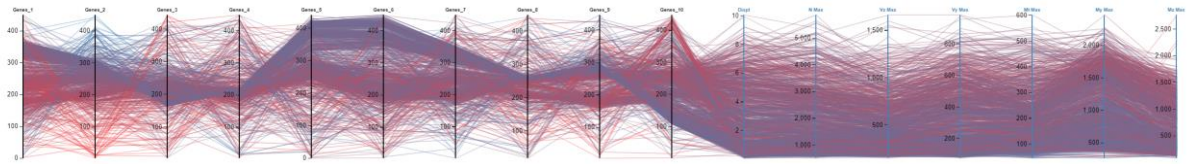
A teljes kimutatás elérhető ezen a linken: https://goo.gl/XaHczq

A következő eljárást már belső pároztatással indítottam, mivel ezzel garantáltjuk, hogy egy adott minimum körül a lehető legalaposabban megvizsgáljuk a lehetséges eredményeket és csökkentjük az eredmények zajosságát. Ekkor már nem rendeztem újra a szerkezetet, így kihasználva az előző futtatások nagyobb eredmény szórását. Az algoritmus az előző eljárás legjobb megoldásától kezdi az optimalizálást.

A beállítások a következők voltak:

Egy generációra vett populáció:	50 fő
Kezdő populáció kezdő szorzója:	2
Generációk közötti genom átviteli százalék:	2%
Párosítás százaléka:	+75%

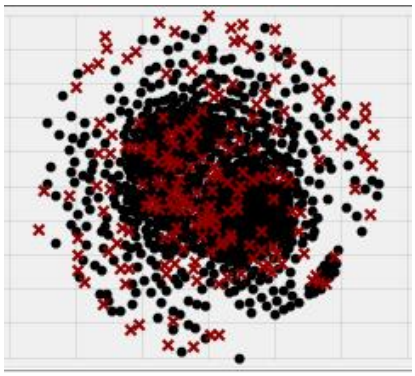
Az eljárás viszont nem járt eredménnyel, nem talált újabb optimumot. 50. generációig futott 10 órán keresztül, majd eredmény hiányában leállította magát.

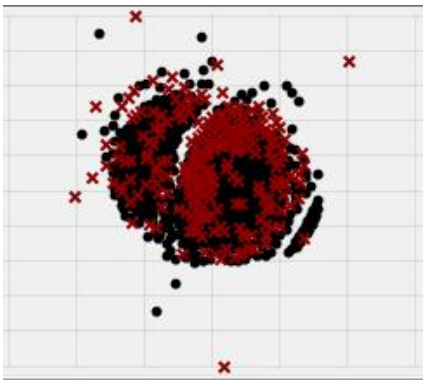


[52]

Példaképpen megfigyelhetjük, hogy az eljárás valóban belső pároztatást mutatott.

[53]





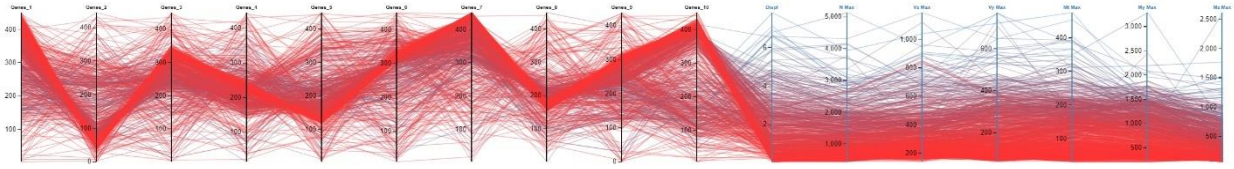
[54]

A teljes kimutatás elérhető ezen a linken: https://goo.gl/wuaSXJ

Ezek után szintén újra rendezés nélkül újabb eljárást indítottam, ezúttal viszont növelve a genom átviteli százalékot, hogy konzisztensebb eredményt kaphassunk, a jól produkáló egyedek nagyobb eséllyel éljenek túl. A beállítások a következők:

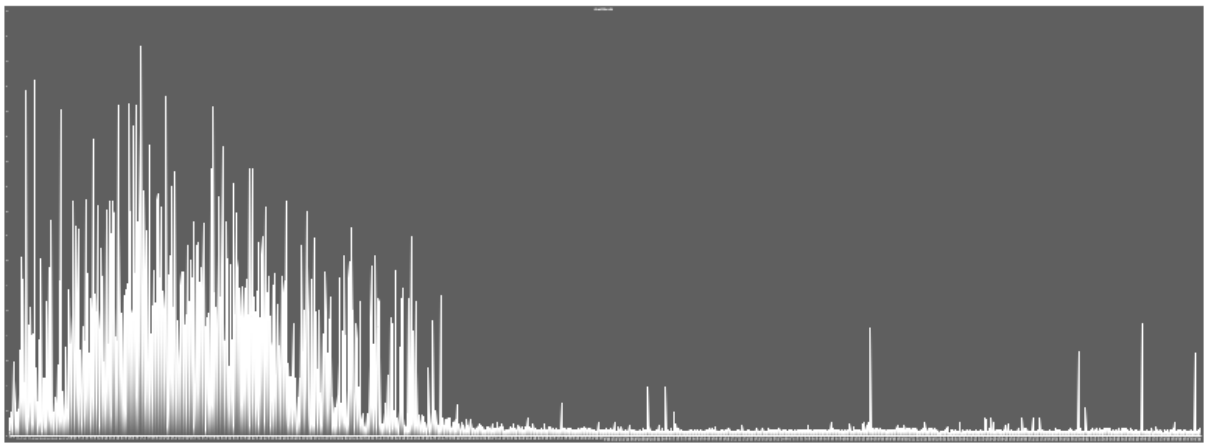
Egy generációra vett populáció:	50 fő
Kezdő populáció kezdő szorzója:	2
Generációk közötti genom átviteli százalék:	5%
Párosítás százaléka:	+75%

Az algoritmus több mint 2 napig futott, a 112. generációig jutott el és rendkívül sok lokális minimumot megtalált. A hosszú futási idejének köszönhetően és vélhetőleg az erősebb tovább öröklésnek, jobb eredményeket ért el, mint a Komplex 1 félév során alkalmazott eljárás.
A különböző generációk teljesítménye így tevődött össze [55]:



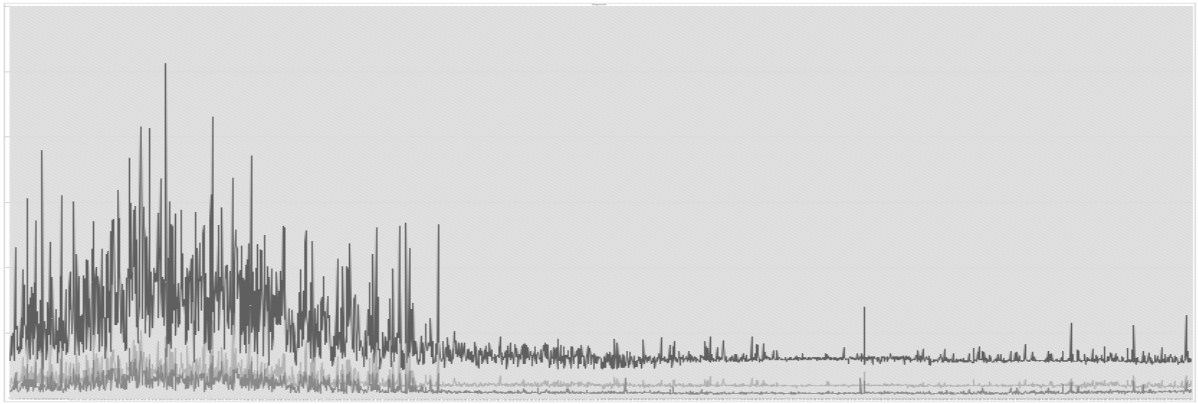
[55]

Az eljárás során az elmozdulás értékei a következők szerint alakultak [56]:



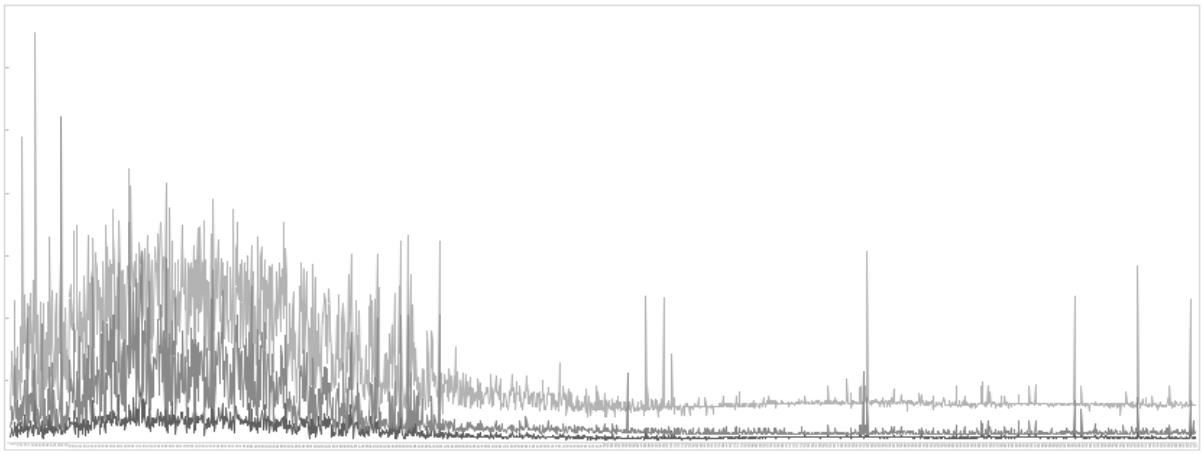
[56]

Jól megfigyelhető a diagramon, hogy ahogy haladunk a vége felé, egyre kisebb kiugrások vannak az elmozdulás értékében, az algoritmus egyre jobban közelít a hegycsúcshoz, a globális minimumhoz. Egyúttal megfigyelhető egy ugrásszerű javulás körülbelül a 30. generáció környékén. Ezt a tendenciát szintén megfigyelhetjük a normál erők, és nyíró erők szempontjából is [57].



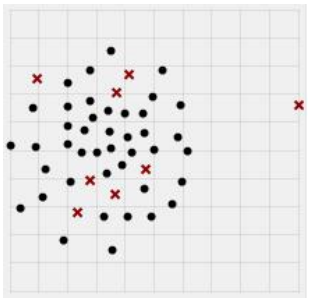
[57]

Illetve a hajlító nyomatékoknál is [58].

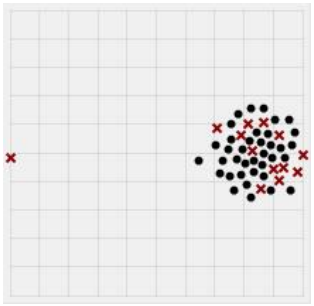


[58]

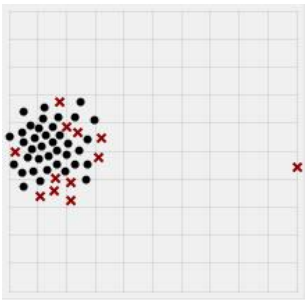
Az algoritmus az eljárás során több minimum pontot is benépesített, ez figyelhető meg a különböző generációk összpontosulásából:



[59]

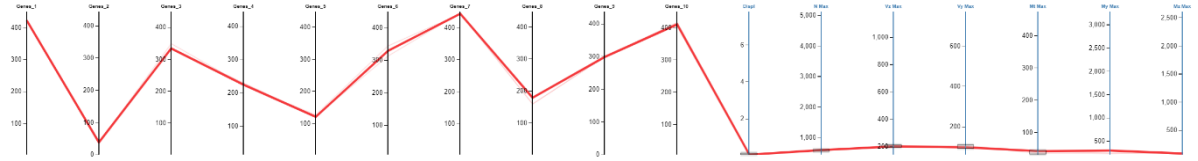


[60]



[61]

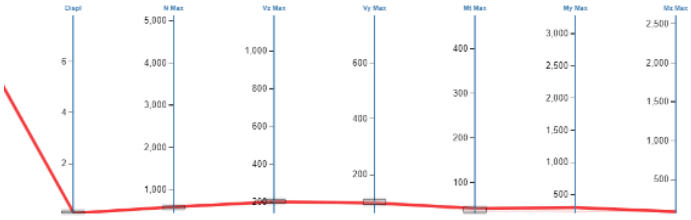
A lehajlás szempontjából legjobban szereplő egyed a következő volt:
Értékei pedig a következők [62]:



[62]

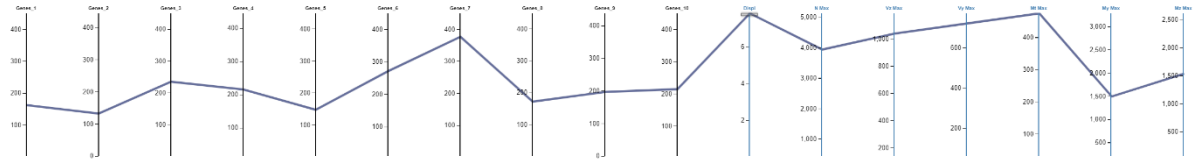
m / kN / kNm

Elmozdulás	0,0595 m
N max	592,03 kN
Vz max	199,93 kN
Vy max	98,36 kN
Mt max	42,17 kNm
My max	298,14 kNm
Mz max	86,43 kNm



[63]

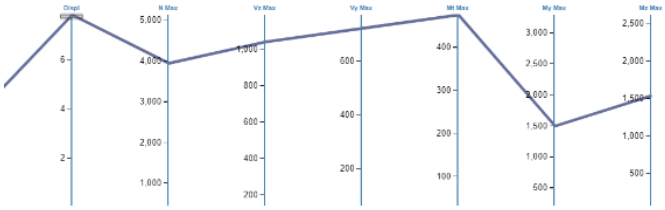
A referencia kedvéért a lehajlás szempontjából legrosszabbul szereplő tag [64]:



[64]

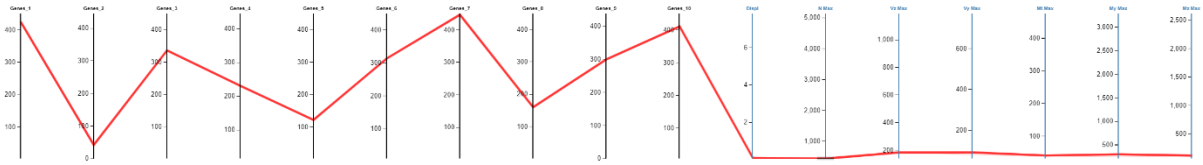
Értékei:

Elmozdulás	7,817 m
N max	3942,54 kN
Vz max	1041,34 kN
Vy max	720,56 kN
Mt max	475,55 kNm
My max	1497,35 kNm
Mz max	1536,39 kNm



[65]

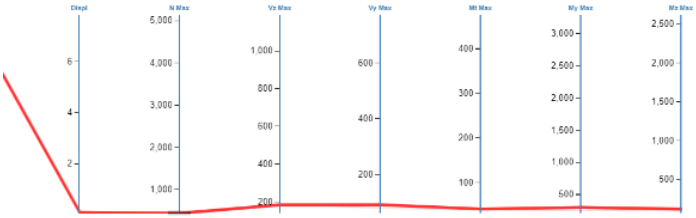
Normál erő szempontjából viszont a következő [66]:



[66]

Értékei:

Elmozdulás	0,0941 m
N max	439,85 kN
Vz max	184,01 kN
Vy max	92,63 kN
Mt max	40,61 kNm
My max	300,81 kNm
Mz max	119,71 kNm



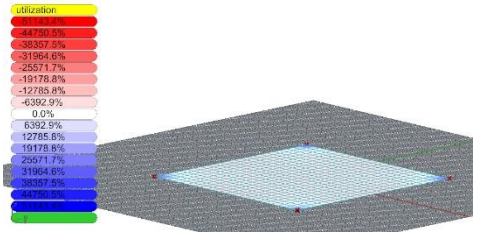
[67]

A teljes dokumentáció ezen a linken elérhető: https://goo.gl/UqMdhR

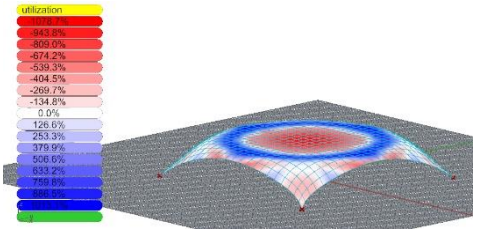
A tesztek alapján észrevehető, hogy 30 generáció környékén elvárható az algoritmustól, hogy egy lokális minimumot megtaláljon, vagy legalább annak közelében tartózkodjon a populációnk. A sikertelen generációk száma a beállításainktól is függ. Ám a 30. generáció elérése több órába is telhet, és így megfontolandóvá válik, hogy más beállításokkal próbálkozzunk tovább.

Forma optimalizálás:

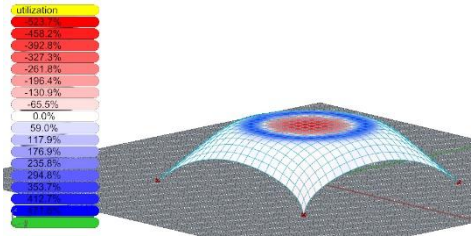
A program lehetőséget ad forma optimalizáló eljárásokra is, amennyiben képesek vagyunk jól definiálni az alapgeometriánkat és annak bemenő paramétereit. A létrejövő szerkezetet ugyanúgy képesek vagyunk vizsgálni a VEM modell alapján. Erre jó eszközt jelenthet például a Kangaroo[1] nevű kiegészítő, mellyel képesek vagyunk a Hook törvény alapján elasztikus, kötél és ponyva szerkezeteteket szimulálni. Így gyorsan és egyszerűen előállíthatók nyomásvonal alakú, vagy csak nyomott szerkezetek csökkentve vagy megszüntetve a szerkezetben fellépő hajlító nyomatékokat. Ezekkel a szimulált függesztett szerkezetekkel, könnyen előállíthatók akár Frei Otto vagy Gaudi szerkezetei is a szoftveren belül, hiszen ők is felfüggesztett modelleket vizsgáltak. Képesek vagyunk fix pontokat, megnyúlási százalékot és erőket is definiálni a szerkezeten. Másik forma optimalizációs eljárás lehet, ha a geometriánk alapparamétereinek a változtatását is egy megoldó motorra bízzuk egy, vagy több fitness értéktől függően. Hasonló függesztett szerkezeteket szimulálhatunk Karamba 3D-vel is.[4]



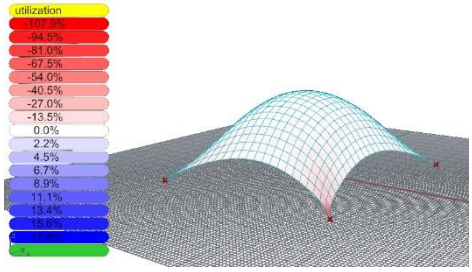
[68]



[69]



[70]

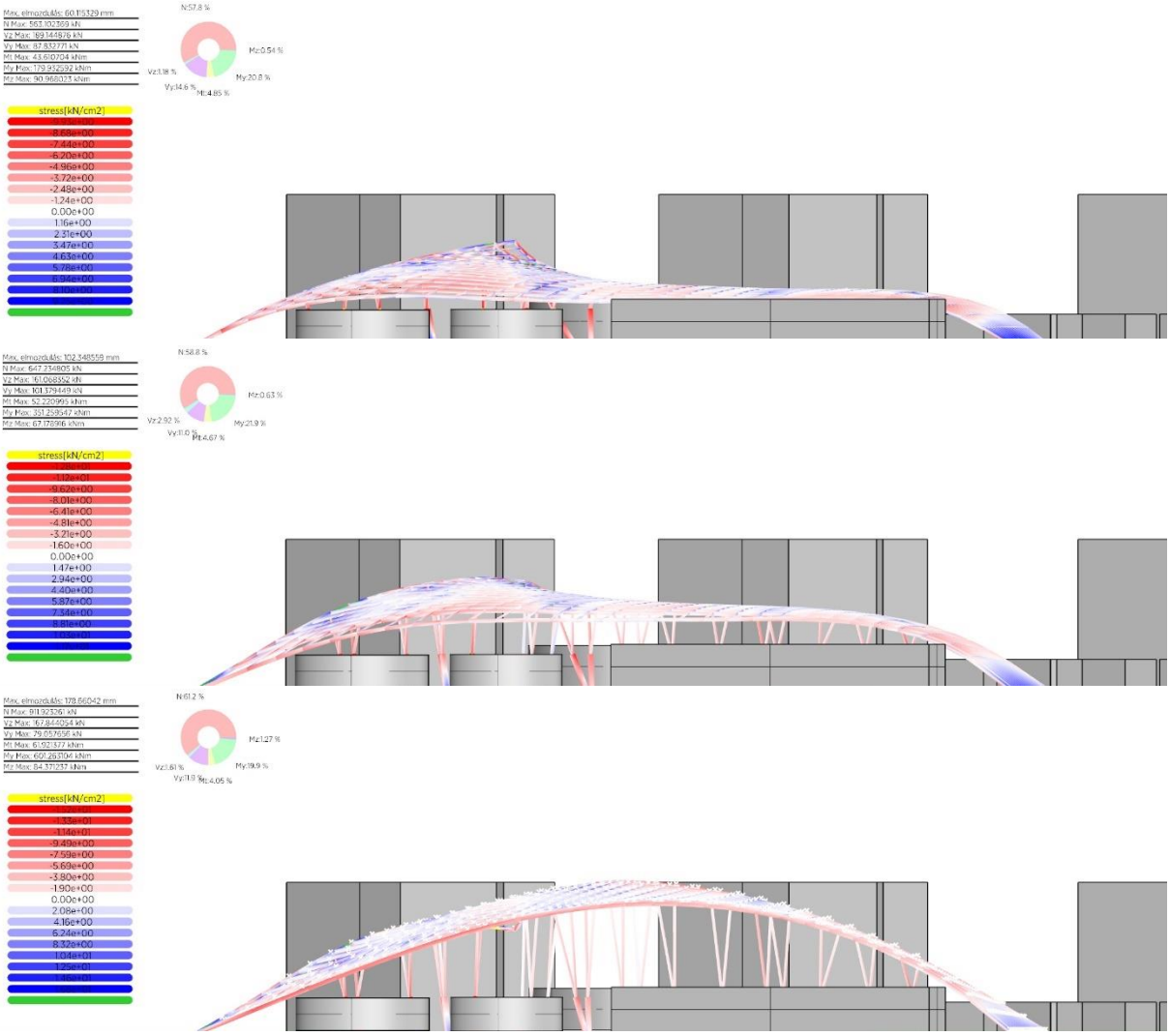


[71]

Az alábbi példában egy 4 sarkánál megfogott ponyva szerkezetet szimuláltam [68-71] inverz alakban. Az algoritmusban könnyen nyomon tudjuk követni a szerkezet igénybevételeit, és ezeknek megfelelően módosítani az alapgeometriát. Változtathatjuk a felfelé ható erők nagyságát, mely a ponyvát felfújja, vagy a megtámasztások helyét.

[1] (Tedeschi, 2014)
[4] (Pugnale, 2006-2018)

Ezt az eljárást megismételhetjük a tetőszerkezettel is, formaoptimalizáció nélkül a következő eredményeket kapjuk [74]:



[72]

[73]

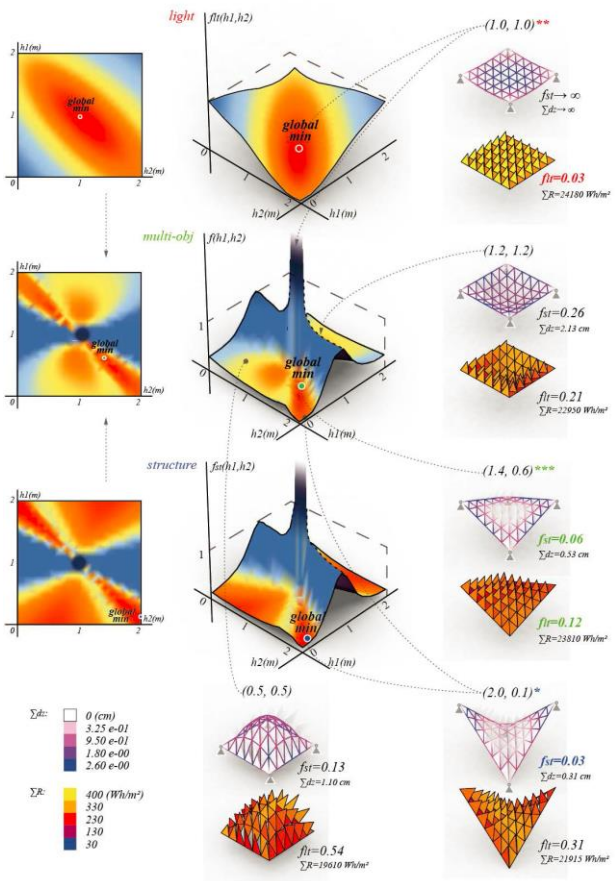
[74]

Ebből jól látható, hogy a szerkezet jobban felveszi az erővonal szerű alakot, így csökkentve benne a hajlító nyomatékok arányát és a szerkezetben ébredő többi feszültséget is, növelve a normál erők arányát, az eredeti állással szemben [72]. Ám ebben az esetben szükségessé válik az oszlopok helyét meghatározó evolúciós eljárás újrafuttatása, mivel megváltozott a szerkezet alapgeometriája. Viszont a kapott forma építészetileg megkérdőjelezhető, szerkezetileg optimálisabb, ám anyagköltségét tekintve a felület megnagyobbodása miatt drágább. Így fontos szem előtt tartani, hogy a tervezés melyik fázisában alkalmazzuk magát a forma optimalizációt, vagy milyen stratégiát építünk köré, hova teszünk fix pontokat és hol gátoljuk, hogy az algoritmus változtasson a formán. Jelen esetben csak a letámaszkodási pontok lettek fixálva. Másik szabályozási lehetőségünk, hogy a szerkezet „felfújását” melyik adott pontban állítjuk meg, mikor egyenlő az optimumok szintje mind építészeti, mind tartószerkezeti szempontból [73].

Több fitness érték optimalizálása:

Ma már egyre több megoldó motor elérhető a Grasshopper 3D-n belül, az utóbbi években elkezdtek megjelenni az evolúciós eljárásokon kívül neurális hálók is. Továbbá az evolúciós eljárások is fejlődtek. Erre jó példa az Octopus nevű kiegészítő, mely legnagyobb előnye az alap Galapagos motorral szemben, hogy egyszerre több fitness értéket is figyelembe tud venni, így, mint a fenti példában csak a tető elmozdulását vettük figyelembe, ennél az eljárásnál akár több peremfeltételt is megadhatunk. Ezzel képesek vagyunk bővíteni a választható jó megoldások számát. Sok múlik azon, hogy milyen peremfeltételeket választunk, hogy azok segítik egymást és létezhet egy közös optimumuk, vagy ellentétesek és emiatt a két szempontból leginkább megfelelő eredményt valahol a két átlagánál kell keresni. A több fitness értékű optimalizációk veszélye valójában ebben rejlik, mérlegelnünk kell, hogy melyik fitness értéket (vagy értékeket) részesítsük inkább előnyben, mivel minél többet definiálunk, annál több lehetséges optimumot választhatunk. Előállhatnak olyan eredmények, hogy az egyik fitnessnek rendkívül megfelel, ám a másik kárára. Amennyiben például egy szerkezetet egyszerre próbálunk optimalizálni bevilágítás és szerkezeti állékonyság szempontjából, az optimális megoldás a következő szerint alakul [75][5]: Jól látható, hogy a benapozás és állékonyság közös optimum valahol, a külön-külön vett optimumok átlagában helyezkedik el.

[75]



[5] (Computational Morphogenesis in architecture: structure and light as a multi-objective design/optimization problem, 2013)

Konklúzió:

A fenti példákon keresztül láthatjuk, hogy ezek az eljárások sokszor csak megfelelő ismeretekkel használhatók, fontos egy előzetes koncepció felállítása és az eszközkészletük ismerete, ellenkező esetben csupán szerencsén múlhat, hogy megkapjuk-e a várt eredményt. Továbbá az evolúciós eljárások sajátossága, hogy rendkívül időigényesek, ahhoz, hogy megfelelő eredményt kapjunk. Egy komplex feladatnál akár napokba is telhet, és megfontolandó, hogy saját fejlesztésű szoftvert használjunk rá, mivel a Grasshopper 3D számítási erőforrás kezelése nem mindig kielégítő. Ám a tárgyalt eljárások erejét és a bennük rejlő potenciált nem szabad elfelejteni. Egy vagy több fitness értékkel definiálható feladatokban nagyon strapabírók lehetnek, olyan megoldások felé mutathatnak, melyek elsőre nem lettek volna egyértelműek. Viszont eredményeik általában felülvizsgálatot igényelnek, szakértelem szükséges az adott problémához is, aminél alkalmazni kívánjuk őket. Cserébe viszont rendkívül sok megoldást kézhez kapunk, részmegoldásokat is elfogadhatunk véglegesnek és új irányokba terelheti tervünket. Koncepcionális fázisban is már rengeteg lehetőséget tesztelhetünk és felülbírálhatunk, mely segítheti a terv megfelelő irányának megtalálását. A terv flexibilitását mi szabályozhatjuk, annak megfelelően, hogy hogyan írjuk meg annak kódját, vagy biológiából vett hasonlattal élve, a DNS-ét. Kivitelezésben is segítségünkre válhat ez a metodika, irreguláris szerkezetek realizálást segítheti, nem látható optimumokra találhatunk rá. Időigényességük ellenére még mindig több időt spórolhatunk velük, mintha nélkülük szeretnénk az összes opciót végigpróbálni. Teljes tervezési feladatok ellátásra viszont még nem képesek. A mi feladatunk, hogy megfelelő peremfeltételeket és állítható paramétereket definiáljunk. Az informatika folyamatos, rohamos fejlődésével mindenképpen érdemes ezeken az eljárásokon tartanunk a szemünket. Az építészeten felül egyre többet kísérleteznek a különféle mesterséges intelligenciák összevonásával, tudásuk összekapcsolása és gyengeségeik kiegyenlítése végett, evolúciós eljárások irányítását és finomhangolását, eredményeik kiértékelését bízzák mélytanuló programokra. Ugyanakkor gyártástechnológiák folyamatos fejlődésének köszönhetően ma már egyre inkább realizálhatóvá válhatnak nem euklideszi geometrián alapuló szerkezetek is.

Saját tervemben is nagy segítséget jelentettek ezek az eljárások, realizálhatóvá tették a formai elképzeléseimet. Segítségükkel képes voltam egy irreguláris, ám megvalósítható szerkezetet tervezni.

Források:

Irodalomjegyzék:

1. **Tedeschi, Arturo. 2014.** *AAD_ Algorithms Aided Design.* 1. Olaszország : Le Penseur Publisher, 2014.
2. **Domingos, Pedro. 2015.** *The Master Algorithm.* 1. Anglia : Penguin Books, 2015.
3. **Rutten, David. 2011.** I Eat Bugs For Breakfest. [Online] 2011. https://ieatbugsforbreakfast.wordpress.com/.
4. **Pugnale, Alberto. 2006-2018.** albertopugnale. [Online] 2006-2018. http://www.albertopugnale.com/.
5. *Computational Morphogenesis in architecture: structure and light as a multi-objective design/optimization problem.* **Allesandro Liuti, Alberto Pugnale, Alessio Erioli. 2013.** 2013.