Debreceni Egyetem
Informatikai Kar

HÁROM KÖR A HÁROMSZÖGBEN

Konzulens:
dr. Aszalós László
egyetemi adjunktus

Készítette:
Király Péter
programtervező matematikus
szakos hallgató

DEBRECEN, 2008

Tartalomjegyzék

1. Bevezetés

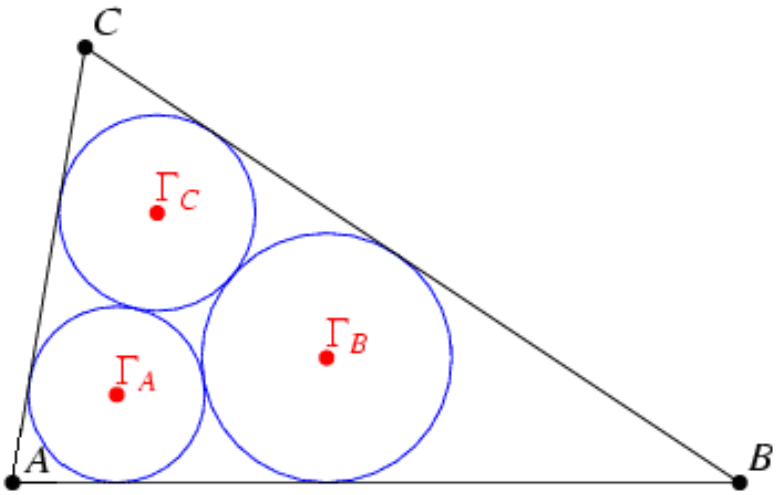
A matematikai problémák fontos kutatási területei a mesterséges intelligenciának, ahol a cél minél hatékonyabb keresési algoritmusok fejlesztése. Ezért a kutatási eredmények általában felhasználhatók más hasonló területen is. Mivel a hagyományos keresési algoritmusok idő- és tárbonyolultsága nagy lehet, így szükségessé vált a fejlettebb keresési eljárások alkalmazása.

A szimulált lehűtés algoritmus (SA) közel áll a genetikus algoritmusokhoz. Felfogható egy speciális genetikus algoritmusként. Robusztus struktúrájából kifolyólag képes olyan nehéz feladatokat is megoldani, ahol nem rendelkezünk terület-specifikus leírással, ezért az algoritmus probléma-független. A SA az adaptív keresési technikák osztályát képviselik, a hagyományos „gyenge módszerekhez" képest hatékony, terület-független keresési heurisztikát kínál. Mint sztochasztikus algoritmus jól alkalmazható NP-teljes feladatokra és nagy keresési térrel rendelkező problémákra.

A dolgozatban azzal fogunk foglalkozni, hogy ismertessünk néhány fejlett keresési módszert, és egy algoritmust fogunk szemléltetni egy konkrét példán keresztül. A feladat az, hogy egy adott háromszögben mely az a három, egymás nem átfedő kör, mely maximális területet fed le a háromszögből. Mivel máig megoldatlan ez a probléma a geometriában, így ezekkel az algoritmusokkal próbáljuk közelíteni az optimális megoldáshoz. Ezen kívül a dolgozat folyamán áttekintjük az SA kapcsolatát a genetikus algoritmusokkal, a probléma megoldásához szükséges matematikai hátteret és a Delphi 7 programnyelv felhasznált elemeit is ismertetem.

2. Matematikai háttér

Hasonló jellegű probléma a Malfatti-körök (Malfatti-circles) szerkesztésének megvalósítása. Annyiban tér el a dolgozatban bemutatott feladattól, hogy úgy kell felvenni a köröket, hogy egyenként érintsék a háromszög két oldalát és a másik két kört is, az alábbi ábrának megfelelően:



1803-ban vetette fel és oldotta meg először ezt a problémát Gian Francesco Malfatti (1731-1807) olasz matematikus, úgy hogy kiszámította a megszerkesztendő körök radiusait. Innen származik az úgynevezett „parkettázási probléma”. Lényegében egyszerűsíthető volt a feladat a szerkesztéstől eltekintve arra, hogy három kört írjunk egy háromszögbe maximális területtel és lényeges volt az is, hogy a körök érintsék egymást. A probléma nagyon felkapott lett a 19. század végén, születtek megoldások, de bizonyítani nem sikerült a maximális területlefedést. 1929-ben Lob és Richmond sejtése alapján a Malfatti-körök nem oldják meg a „pakettázási problémát”, amit később 1967-ben M. Goldberg egy szemléletes, grafikus bizonyítással megmutatatta. Ez alapján a Malfatti-körök soha nem oldják meg a lefedési problémát.

A program megírása során jelentős szerepet játszott a távolságmérés, illetve a koordináta-rendszerben való geometriai elhelyezkedése a pontoknak, alakzatoknak. A programban csak pozitív koordinátákat használok, 0 és 1 közé esőket. A kirajzoltatott háromszögem koordinátái (0;0), (1;0) a harmadik (X;Y) amelyek 0 és 1 között választható értékek.

A sík minden egyes egyeneséhez hozzá tudunk rendelni egy olyan kétismeretlenes egyenletet, amelyet az egyenes pontjainak koordinátái, és csak azok elégítenek ki. Egy egyenest a síkon egyértelműen meghatározza

(1) egy pontja és az iránya,

(2) két pontja.

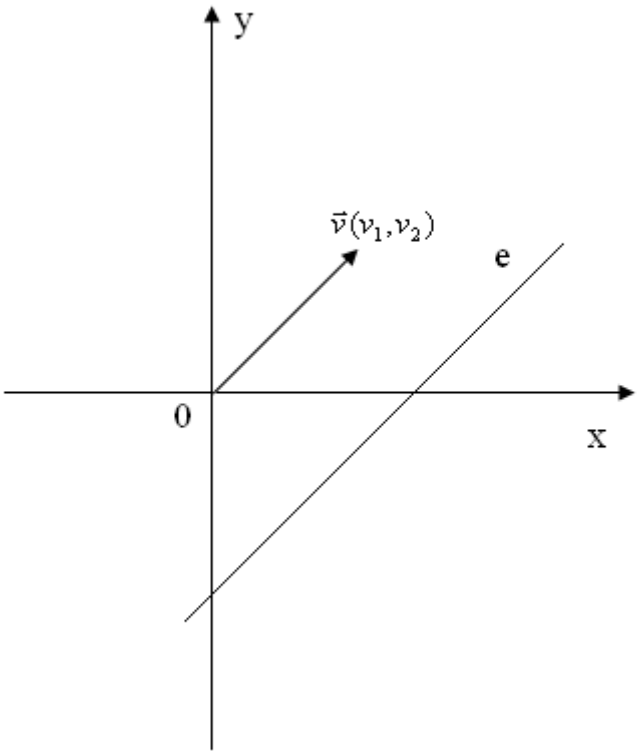
Az egyenes irányát a síkon többféleképpen megadhatjuk. Két természetesen adódó lehetőség:

(1) megadunk egy vele párhuzamos egyenest,

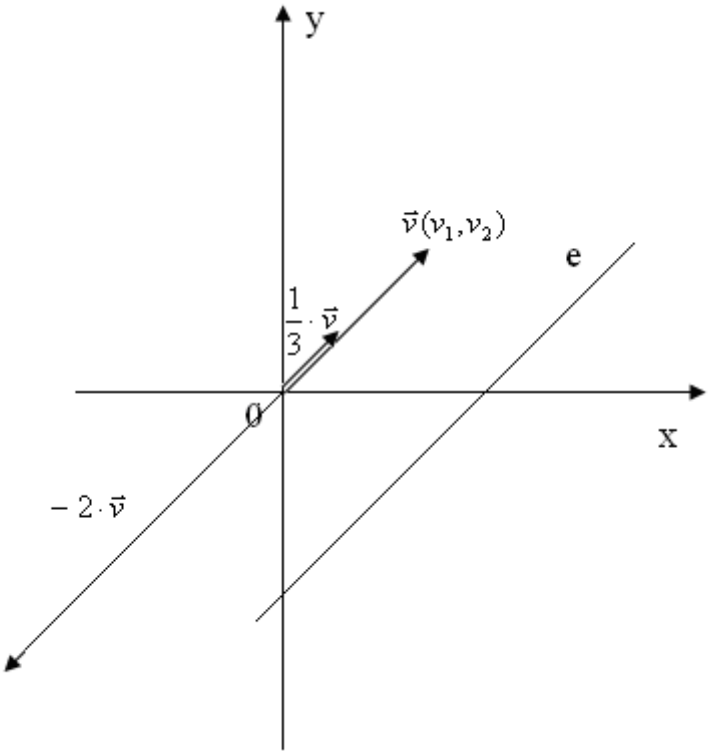
(2) megadunk egy rá merőleges egyenest.

A továbbiakban megmutatjuk, hogy az egyenes irányát milyen számszerű jellemzőkkel adhatjuk meg a koordináta-rendszerben.

Egy egyenes irányvektora bármely, az egyenessel párhuzamos, nullvektortól különböző vektor. Jele: $\vec{v}(v_1, v_2)$



Egy nullvektortól különböző vektor pontosan akkor párhuzamos egy egyenessel, ha a vektort reprezentáló irányított szakasz egyenese párhuzamos az egyenessel. Ha $\vec{v}$ az *e* egyenesnek irányvektora, akkor bármely 0-tól különböző λ valós szám esetén $\lambda \cdot \vec{v}$ is irányvektora *e*-nek.



Ha adott az e egyenes két különböző pontja $P_1(x_1;y_1)$ és $P_2(x_2;y_2)$, akkor e egy irányvektora $\vec{v}\,(x_2\text{-}x_1;\ y_2\text{-}y_1)$.

Ha konkrét számokkal végezzük a számításainkat, akkor érdemes olyan irányvektort választani, amelynek koordinátáival a lehető legegyszerűbb módon tudunk számolni. Pl.: $\vec{v}\,(4;6)$ akkor számolhatunk $\vec{v}' = \frac{\vec{v}}{2}$ is, így $\vec{v}' = (2;3)$.

Egy egyenes irányvektoros egyenletét fel tudjuk írni, ha adott a $\vec{v}(v_1, v_2)$ irányvektora és egy $P_0(x_0;y_0)$ pont, ami az egyenes egy pontja. Ekkor az irányvektoros egyenlet: $v_2 x - v_1 y = v_2 x_0 - v_1 y_0$.

A síkbeli derékszögű koordináta-rendszerben az egyenes egyenlete olyan

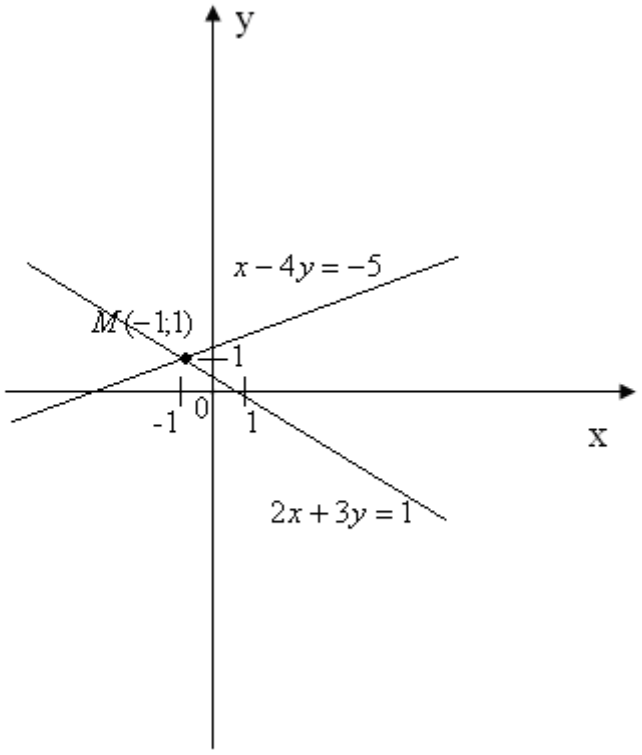
$$Ax + By + C = 0$$

alakú kétismeretlenes lineáris egyenlet, amelyben A és B közül legalább az egyik 0-tól különböző ($A^2 + B^2 > 0$)

Ha tudjuk az egyenes egyenletét, akkor azt is el tudjuk dönteni, hogy egy pont hol helyezkedik el az egyeneshez képest.

Ha egy tetszőleges pont koordinátáját behelyettesítjük az egyenletbe és az egyenlőség továbbra is fenn áll, akkor a pont az egyenesen van rajta. Ha baloldalra rendezzük az egyenletünket és behelyettesítjük a tetszőleges pontunk koordinátáit és az egyenlet baloldala nagyobb mint 0, akkor a pontunk az egyenes felett helyezkedik el, ha kisebb, mint 0, akkor az egyenes alatt.

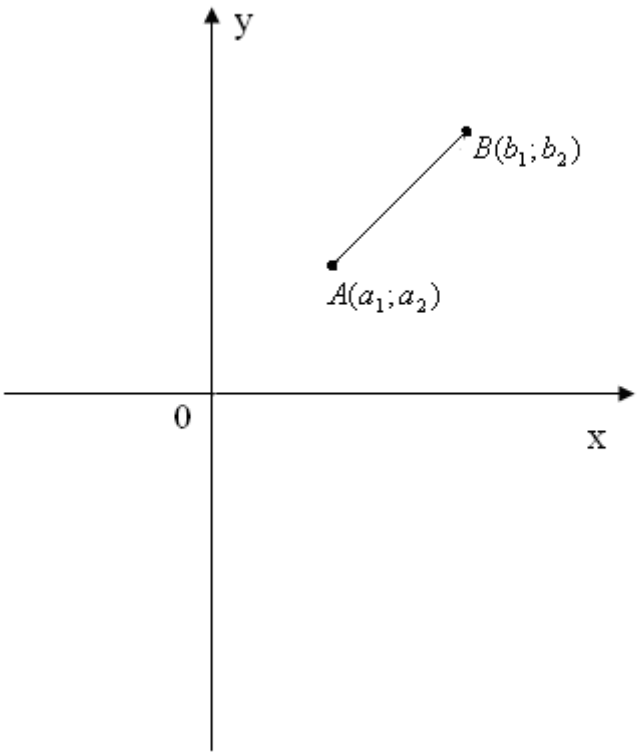
Ha két egyenesnek ismerjük az egyenletét, akkor ki tudjuk számolni a metszéspontjukat. Két síkbeli metsző egyenes metszéspontjának koordinátái a két egyenes egyenletéből álló egyenletrendszer megoldásai. Pl.:



Pont és egyenes távolságát is meg tudjuk határozni csak az egyenes egyenletére és egy tetszőleges pont koordinátájára van szükség. De először nézzük meg két pont távolságának a meghatározását,

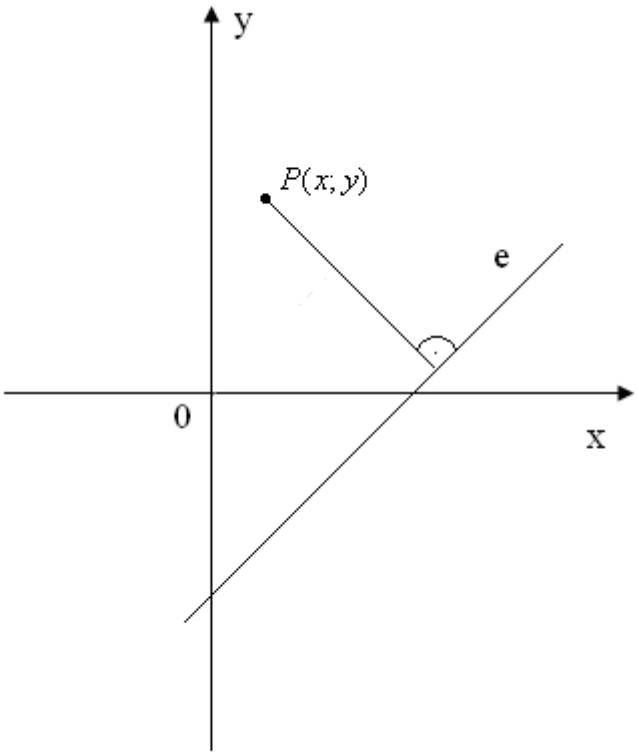
Az $A(a_1;a_2)$ és $B(b_1;b_2)$ pontok távolsága:

$$AB = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$$



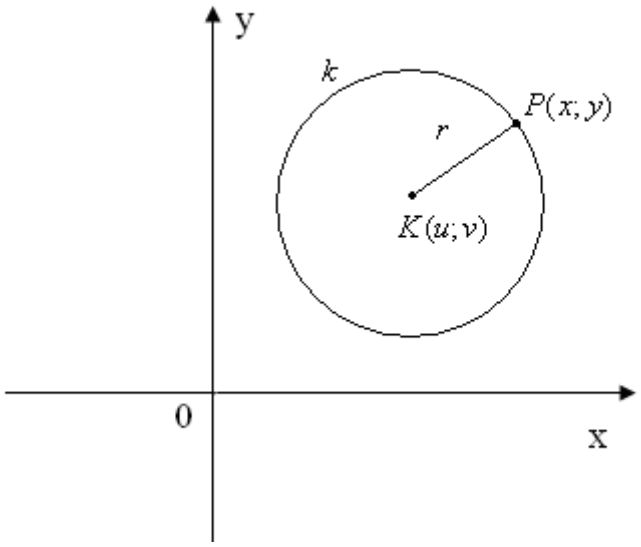
Pont és egyenes távolsága a pontból az egyenesre állított merőleges talppontjának és a tekintett pontnak a távolsága. Melyet a következő képletből ki lehet számolni:

$$\frac{|Ax + By + C|}{\sqrt{A^2 + B^2}}$$



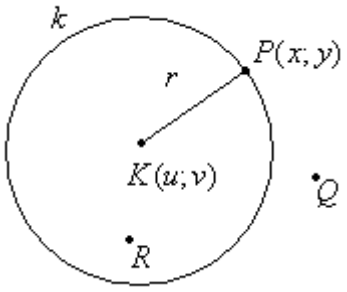
A sík egy másik nevezetes ponthalmazát, a kört is tudjuk egyértelműen jellemezni egy olyan egyenlettel, amelyet csak a kör pontjainak koordinátái elégítenek ki. A kört egyértelműen meghatározza a síkon a középpontja és a sugara. A K(u;v) középpontú r sugarú körre, akkor és csak akkor illeszkedik a P(x;y) pont ha a következő képletbe behelyettesítve az egyenlőség teljesül.

$$(x-u)^2+(y-v)^2=r^2$$



Egy tetszőleges pontról el tudjuk dönteni, hogy a körön kívül vagy a körön belül helyezkedik el.

Ha $(x-u)^2+(y-v)^2>r^2$, akkor Q pont a körön kívül helyezkedik el, ha $(x-u)^2+(y-v)^2<r^2$, akkor az R pont a körön belül helyezkedik el.



Egy kétismeretlenes másodfokú egyenlet akkor és csak akkor egyenlete egy körnek a síkbeli derékszögű koordináta-rendszerben, ha

$$x^2+y^2+Ax+By+C=0$$

Alakra hozható, ahol A, B, C olyan valós számok, amelyekre teljesül az $A^2 + B^2 - 4C > 0$ egyenlőtlenség, Ezzel az egyenlettel előállított kör középpontja $K\left(-\frac{A}{2};-\frac{B}{2}\right)$, sugara $r = \frac{\sqrt{A^2 + B^2 - 4C}}{2}$

3. Kereső algoritmusok

A hagyományos keresési algoritmusok idő- és tárbonyolultsága nagy lehet, így szükségessé vált a fejlettebb keresési eljárások alkalmazása. Nagyon sok olyan feladat van, amelyre még nem fejlesztettek ki elég gyors, hatékony algoritmusokat. A legtöbb ilyen feladat az optimalizációs és a keresési feladatok osztályába tartozik. A nehéz optimalizációs feladatoknál nem ismerjük az optimális megoldást, így közelítő megoldásokat keresünk a fejlett kereső algoritmusokkal.

A három kör a háromszögben problémánál nem ismerjük az optimális megoldást, ezért optimalizációs kereső algoritmusokat alkalmazunk.

Ilyen algoritmusok:

1. Genetikus algoritmusok (GA):
   Szelekció, reprodukció
   Mutáció
   Keresztezés, rekombináció
2. Szimulált (le)hűtés
3. Tabukeresés
4. A hegymászó algoritmus

Ezek közül a szimulált lehűtést választottam a probléma megoldására. Mivel a körök középpontjai és sugarai a racionális számok halmazából lettek választva, ezért a feladat folytonos. A genetikus algoritmusok DNS kódolásának megválasztása ezekre feladatok nagyon nehéz. A tabukeresést főleg diszkrét feladatoknál alkalmazzák. A hegymászó algoritmus pedig könnyen „leragadhat” egy lokális maximumnál.

3.1. Genetikus algoritmusok (GA):

A genetikus algoritmusok az utóbbi évtizedben széles körben elterjedt optimalizálási módszereknek bizonyultak. Robusztus struktúrájukból kifolyólag képesek olyan nehéz feladatokat is megoldani, ahol nem rendelkezünk terület-specifikus leírással, ezért az algoritmus probléma-független. A GA az adaptív keresési technikák osztályát képviselik, a hagyományos, gyenge módszerekhez képest egy sor hatékony, terület-független keresési heurisztikát kínál fel. A természetes szelekció elvén alapszik, számos biológiai folyamatot imitálva, szimulálva: kromoszóma reprezentáció, genetikus operátorok, egyed kiválasztás, rátermettségi függvény stb. Mint sztochasztikus algoritmus jól alkalmazható NP-teljes feladatokra és nagy keresési térrel rendelkező problémákra. A matematikai problémák nagy része pont ilyen típusú feladatokat képvisel: nagy a keresési tér és nincs konkrét terület-specifikus ismeret.

A természetes szelekció mechanizmusát szimulálják egy valószínűsített adatcsere segítségével, alkalmazva a darwini elvet: a legrátermettebb túlélése (*survival of the fittest*). A GA az evolúciós számítások egy részhalmaza, amelyek evolúciós technikákat alkalmaznak a számítási algoritmusokra. Mivel az utóbbi években nagy érdeklődés mutatkozott a genetikus algoritmusok iránt és széles körben alkalmazták különböző kutatási területeken, így nagyon sok változata létezik. Mi több, a használt genetikus operátorok típusa és száma is problémáról problémára változik, így a GA az adott feladatra testre szabható, biztosítva ily módon a jobb és gyorsabb konvergenciát. Számos könyv jelent meg a genetikus algoritmusokról és minden szerző más-más módon közelíti meg őket, bemutatva a sajátos alkalmazási területeket.

Mivel a genetikus algoritmusok a természetben lezajló folyamatokból inspirálódtak, így a terminológia egy részét a biológiából (elsősorban a genetikából) kölcsönözték, amit szükség esetén az informatikában ismert fogalmakkal bővítettek ki. A genetikus algoritmusok jobb megértése végett, ismételjük át a genetikában használt fogalmakat.

Az egy fajon belüli csoportot populációnak nevezzük. A populációt egyedek alkotják, amelyeket a kromoszómájuk határoz meg. A kromoszóma gének sorozatából áll, amely a tulajdonságok átörökítését (a szülőről az utódra) biztosítja. Egy egyedben levő gének összességét genotípusnak nevezzük. A fenotípus a genotípus megnyilvánulása az adott egyedben és tartalmazza a szerzett tulajdonságokat is. A keresztezés és a mutáció során az egyes egyedekből új egyedek alakulnak ki.

A 60-as években megpróbáltak a természetben lezajló szelekciós folyamatok mintájára számítógépes modelleket létrehozni. Ezek a modellek képesek mérnöki (elsősorban optimalizálási) feladatok megoldására.

Egymástól függetlenül több próbálkozás is született. Németországban 1973-ban Rechenberg vezette be az **evolúciós stratégiák**at (*Evolutionsstrategie*), amelyeket repülőgépszárnyak paramétereinek az optimalizálására használt. Az **evolúciós programozás**t (*evolutionary programming*) Fogel, Owens és Walsh dolgozta ki Amerikában, és egyszerű problémák megoldására szolgáló véges automaták automatikus kifejlesztésére használták. A **genetikus programozás** (*genetic programming*) egy még újabb terület és a genetikus algoritmus egy speciális alkalmazási területe, amikor is a cél meghatározott feladatokat végrehajtó számítógép programok (általában LISP nyelven) automatikus fejlesztése. Ez a próbálkozás Koza nevéhez fűződik (1992).

A **genetikus algoritmusok**at (*genetic algorithm*) Holland fejlesztette ki a michigani egyetemen és 1975-ben összefoglalta a kutatás eredményeit. Célja a szelekció és adaptáció számítógépes és matematikai modellezése volt.

A genetikus algoritmus populációk sorozatát állítja elő operátorok segítségével úgy, hogy egyszerre több megoldással (egyeddel) dolgozik. Ahhoz, hogy a megoldásokból egyszerű operátorok segítségével könnyen újabb megoldásokat lehessen szerkeszteni, a megoldásokat kódolni kell. Minden megoldáshoz, hozzárendelünk egy szintaktikailag jól és könnyen kezelhető betű- vagy számsorozatot egy **kódoló függvény** segítségével. Ezt nevezzük a megoldás **genotípus**ának, míg maga a megoldás a **fenotípus**. A

genotípus pozíciói (indexei) a **gén**ek, az ott levő értékek (számok vagy betűk) az **allél**ok.

A megoldások értékeit egy értékelő függvény, vagy **rátermettségi függvény** (*fitness function*) segítségével adjuk meg. Ennek a függvénynek kell megkeresni a globális optimumát. A rátermettségi függvény megválasztása lehet a legnehezebb feladat és egyben a legfontosabb is, hiszen ennek segítségével mérjük az egyedek teljesítményét, alkalmasságát, rátermettségét. Általában a rátermettség kiszámolása sok időt vesz igénybe, ezért sokat dob az algoritmuson, ha ezt jól választottuk meg.

A kódokon alkalmazott operátorokat három nagy csoportba lehet sorolni. Az első csoportba tartoznak a **szelekció**k, amelyek valamilyen módszer segítségével bizonyos számú egyedet (általában a legrátermettebbeket) választanak ki a populációból. Az így kiválasztott egyedekre alkalmazzuk a **rekombináció**t, amely során két szülő tulajdonságait ötvözve rátermettebb utódokat hozunk létre. A leszármazottak bizonyos százaléka mutálódhat is, alkalmazva a **mutáció**s operátort.

Szelekció, reprodukció

A szelekciós operátor probléma-független és a rátermettséget veszi figyelembe. Ez kiválaszt a populációból egy egyedet (megoldást), és ezt annyiszor kell elvégezni, ahány szülőre szükség van az új populáció előállításához.

Az leggyakrabban használt ilyen szelekciós operátor a **rátermettség-arányos szelekció** (*fitness proportionate selection*), amely szerint egy megoldás kiválasztásának a valószínűsége annál nagyobb, minél nagyobb a rátermettsége a populáció rátermettségi átlagához képest. Megemlíthetjük a **rang szelekciót** (*rank selection*), a **verseny szelekciót** (*tournament selection*) és az **elitlistát** használó szelekciót is. A fent leírt szelekciós operátorokat tetszőleges számszor alkalmazhatjuk, annak függvényében, hogy hány egyedre van szükségünk az új populációban. Ha a régi populáció összes tagját le akarjuk cserélni, akkor az adott szelekciós algoritmust annyiszor hajtjuk végre amennyi egyedet tartalmaz a populáció.

Mutáció

A mutáció unáris operátor, azaz egy operandust igényel, ami egy megoldás és ebből állít elő egy újabbat. A célja az egyedek, a populáció frissítése, változatosság bevitele a populációba. Ez biztosítja a lokális optimumba való beragadástól a védelmet. Ekkor véletlenszerűen kiválasztott pozíciókon levő allél értékeket változtatunk meg. Általában ezt úgy oldják meg, hogy minden pozíció egy rögzített valószínűséggel mutálódhat, és ezt úgy állítják be, hogy átlagosan egy pozíció változzon meg egy kódban.

Keresztezés, rekombináció

A rekombinációk (kereszteződés) több kiindulási megoldásból (szülőből) állítanak elő újabb megoldást, tehát egy bináris operátor. A célja az, hogy különböző egyedek tulajdonságait ötvözve újabb, esetleg rátermettebb egyedeket hozzunk létre. Az egyik ilyen rekombinációs operátor az **egypontos keresztezés** (*1-point crossover*). Véletlenszerűen választunk egy **kereszteződési pont**ot (*crossover point*) és a kapott két fél kódból új megoldást rakunk össze. Jól látszik, hogy a leszármazottak mind a két szülőtől örökölnek tulajdonságokat. A másik rekombinációs operátor az **egyenletes keresztezés** (*uniform crossover*). Itt a mutációkhoz hasonlóan, kiválasztott pozíciókon a kiindulási kódok betűket cserélnek. Egy pozíció kiválasztásának a valószínűsége rendszerint ½, ami jóval nagyobb mint a mutáció esetében.

A genetikus algoritmusok a sok jó tulajdonságok mellett rendelkeznek egy pár hátulütővel. Az egyik ilyen probléma az, hogy beragadhatnak a lokális optimumba. Más szavakkal megkapnak egy elfogadható megoldást, de nem a legjobbat. A probléma kiküszöbölésére számos megoldás van, az együtt-fejlődés (*co-evolution*) vagy koevolúció is erre hivatott. Ezen elv értelmében egyidejűleg több populációt is ki lehet fejleszteni párhuzamosan, azaz szálakon (*thread*) futó genetikus algoritmusok által. Egy többprocesszoros rendszeren, amely támogatja az SMP-t (*Simmetric Multiprocessing*) ez a feladat nem jelent plusz időt. A különböző populációk közötti adatcserét a kromoszómák vándorlásával oldjuk meg, biztosítva ily módon a fontos génanyagot.

A genetikus algoritmus struktúrája nagyon egyszerű. Egy vázlatos és általános formában megadott algoritmus:

```
Hozzuk létre a P0 kezdeti populációt
t := 0
while not Kilépés(Pt)
      Pt' := UjElemek(Pt)
      Pt+1 := UjPopuláció(Pt',Pt)
      t := t + 1
endwhile
```

3.2. Szimulált (le)hűtés

A szimulált hűtés a statisztikai mechanikából származtatott sztochasztikus számítási technika és terjedelmes optimalizálási feladatok globális minimális költségét hivatott megkeresni, vagy legalább is azt jól megközelíteni. S. Kirkpatrick és társai voltak az elsők akik alkalmazták ezt a statisztikai fizikából vett szimulációs technikát a kombinatorikus optimalizálási feladatokra. Ezt a módszert elsősorban a huzal vezetésre (*wire routing*) és a magas fokon integrált áramkörök (***V**ery **L**arge **S**cale **I**ntegration . VLSI*) tervezésére használták, amikor is a cél az elektronikus komponensek elhelyezésének az optimalizálása volt. A szimulált hűtés struktúrája:

```
t := 0
T kezdeti hőmérséklet inicializálása
Válassz véletlenszerűen egy vc stringet
Értékeld ki vc-t
repeat
      repeat
      Válassz ki egy új stringet vc környezetéből
                        megváltoztatva vc egyetlen bitjét
      if f(vc)<f(vn)
            then vc := vn
      else if random[0,1)<exp{(f(vn)-f(vc))/T}
            then vc := vn
      until (befejeződési feltétel)
```

```
        T := g(T,t)
        t := t + 1
    until (megállási feltétel)
```

A `(befejeződési feltétel)` megnézi, hogy beállt-e a „hőmérsékleti egyensúly”, azaz a kiválasztott új string valószínűségi eloszlása elérte-e a Boltzmann eloszlást. Általában ezt a ciklust egy előre meghatározott számszor végzik el. A T hőmérsékletet lépésekben csökkentik, amíg el nem ér egy alacsony értéket, amire a `(megállási feltétel)` figyelmeztet, vagyis a rendszer „megfagyott”. A végeredmény az algoritmus befejezésével a v_c változóban lesz.

A szimulált hűtés esetében a populáció egyelemű és a kereső operátor a mutáció. Az genetikus algoritmusnál használt `UjPopuláció(`P_t'`,`P_t`)` a következőképpen változik meg: ha az új megoldás rosszabb, mint a régi, akkor is elfogadjuk egy bizonyos valószínűséggel, amit a hőmérséklet paraméter ad meg. Ha a hőmérséklet nulla, csak akkor fogadjuk el, ha nem rosszabb mint a régi megoldás.

3.3. Tabukeresés

A tabukeresés egy lokális keresési algoritmus, iteratív javító algoritmus. Olyan probléma típusokra alkalmazható, ahol a megoldások egy gráf csomópontjain helyezkednek el. Egy probléma megengedett megoldásai (állapotai) közül keresi meg a legjobbat, vagy pedig csak megközelíti azt, és így a megoldás csak lokálisan lesz a legjobb.
A tabukeresés egy lehetséges algoritmusa:

```
s (kezdeti megengedett megoldás kiválasztása)
s* := s
k := 1
T := üres halmaz
while not (megállási feltétel) do
     s' := legjobb elem (s szomszédai – T) halmazból
     T frissítése s'-el
     s := s'
     if s' jobb mint s* then s* := s'
     k := k + 1
endwhile
```

Az s az aktuális megengedett megoldást, míg s^* a keresés során talált legjobb megoldást jelöli és az algoritmus leállása után ebben lesz a végső megoldás is. A T halmaz a tabuhalmaz, ami tartalmazza a legutóbb megvizsgált megoldásokat. Mivel nem lehet az összes utoljára megvizsgált műveletet eltárolni, így egy bizonyos idő után a legrégebbit kitörli. Ezért nevezik a tabuhalmazt még rövid távú memóriának (*recency based memory*) is.
A tabuhalmazt a gyakorlatban egy előre megadott rögzített hosszúságú FIFO adatszerkezettel oldják meg. A `(megállási feltétel)` a következő feltételek egyike lehet: k túllép egy bizonyos határt, vagy k túllép egy bizonyos határt s^* utolsó módosítása óta, vagy az (s szomszédai – T) halmaz üres.

A tabukeresésnél is egyelemű a populáció és a kereső operátor szintén a mutáció. Itt is az `UjPopuláció(`P_t'`,`P_t`)` függvény az eltérő. Az új populáció kiválasztásához megnézzük, hogy az új elem nem szerepel-e a tabulistában. Ha, igen akkor elvetjük, ellenben, ha nem rosszabb mint a régi akkor elfogadjuk. Az új megoldás a tabulistába kerül és a legrégebbi elem törlődik a listából.

3.4. A hegymászó algoritmus

Sok verziója létezik a hegymászó algoritmusnak. Nagyrészt abban különböznek, hogy hogyan választják ki az új elemet vagy stringet (mivel genetikus algoritmusokról van szó) a keresési térből annak érdekében, hogy ezt összehasonlítsa az aktuális legjobb értékkel.
Egy egyszerű iterált hegymászó algoritmus (legmeredekebb felszállás algoritmusa - *steepest ascent hillclimbing*):

```
t := 0
repeat
     lokális := FALSE
     Válassz véletlenszerűen egy vc stringet
     Értékeld ki vc-t
     repeat
     Válassz ki n darab új stringet vc környezetéből
                          megváltoztatva vc egyetlen bitjét
     Válaszd ki a vn stringet az új stringek halmazából,
                amelynek az f célfüggvény-értéke a legnagyobb
     if f(vc)<f(vn)
          then vc := vn
          else lokális := TRUE
     until lokális
     t := t + 1
until t = MAX
```

Érdekes megjegyezni, hogy a fenti algoritmus egyszeri iterációjának sikere a kezdeti stringtől függ. Ha ez a string .nagyon rossz., kicsi a függvény értéke, akkor egyetlen bit megváltoztatásával (egyből nulla és nullából egy) nem tudunk nagy javulást elérni, legfeljebb a lokális optimumot kapjuk meg.
A sztochasztikus hegymászó algoritmusnál a populáció szintén egyelemű és a keresési operátor itt is a mutáció. Az új megoldás felváltja a régit, ha ugyanolyan rátermett vagy rátermettebb. Észrevehető, hogy a hegymászó egy

nulla hőmérsékleten futó szimulált hűtés, vagy nulla hosszúságú tabulistás tabukeresés, sőt egyelemű populációt használó elitista genetikus algoritmusként is felfogható, annak ellenére, hogy ezek a módszerek egymástól független természeti folyamatok modelljei.

4. A program algoritmusa

Inicializálás

Beolvassuk a háromszög harmadik csúcsának koordinátáit.

Kiszámoljuk a háromszög AC és BC oldalának irányvektorait.

Kor_general eljárás

Max := Kor

Ismételjük

    T := 2

    Old := Kor

    Kor_modositas eljárás

    Ha Terület(Kor)<Terület(Old) akkor

        Kor := Old

    Egyébként

        Ha Exp(Terület(Kor)/Terület(Old)/T > random akkor elfogadjuk

    T-t csökkentjük;

Legjobb megoldás kirajzolása és értékek kiírása egy ellenőrző fájlba.

Kor_general eljárás:

Kör X koordinátája := random

Kör Y koordinátája := random

Addig generálunk, amíg a kapott pont a háromszög belsejébe esik.

Vesszük a három oldal távolságát a ponttól és ha már vannak meglévő köreink, akkor azoktól is, és ezen távolságok legkisebbike lesz az adott kör sugara.

Kor_modositas eljárás:

Kör X koordinátája := Kör X koordinátája + (random*2-1)/1000

Kör Y koordinátája := Kör Y koordinátája + (random*2-1)/1000

Addig generálunk, amíg a kapott pont a háromszög belsejébe esik.

Vesszük a három oldal távolságát a ponttól és ha már vannak meglévő köreink, akkor azoktól is, és ezen távolságok legkisebbike lesz az adott kör sugara.

5. A program működése

A szoftver- és hardver feltételei: A programot negyedik generációs programnyelven írtam (DELPHI 7), ezért a következők szükségesek a futtatásához:

- Windows 9x/ME/NT/2000/XP operációs rendszer
- Pentium processzor
- 10 MB HDD
- 64 MB RAM
- 800x600 képernyő-felbontás, high color / 16 bites

A program nyitóképernyőjén egy nyomógombot (Button) és két szövegbeviteli mezőt (Edit) láthatunk.

Meg kell határoznunk a háromszöget, amelybe a szoftver keresni fogja a legjobban „beleillő” köröket. Az általánosság megsértése nélkül a háromszög két csúcsát fixen tartjuk és a harmadik csúcsát változtathatjuk. Elsőként meg

kell adnunk a háromszögünk harmadik csúcsának két koordinátáját. Az csúcsok koordinátái rendre a következők:

```
A(0;0)
B(1;0)
C(edit1.text/400;edit2.text/400).
```

Mivel 0 és 1 közötti értékekkel számolunk, ezért a C csúcs egyes koordinátáit, a program megfelelő működése érdekében maximálisan 400-ra válasszuk.

Az algoritmus futtatása nevű gombbal indítjuk el a számolást.

Első lépésként kiszámoljuk az AC és CB egyenes irányvektoros egyenletét. Ezeket a későbbiekben arra használjuk, hogy a véletlenszerűen generált pontok a háromszög belsejébe essenek.

Inicializáljuk a változóinkat, majd generálunk három véletlen kört, az alábbi szempont alapján:

Véletlenszámokkal kapjuk az első kör középpontjának két koordinátáját. Ekkor meg kell néznünk mekkora lehet a kör sugara. Ebbe beleszól a három oldal távolsága. Ezek közül az értékek közül a legkisebbet fogjuk választani.

A második kör generálásánál hasonlóan járunk el mint az első kör meghatározásánál, de itt az újabb kör sugarának megadásánál figyelembe kell vennünk a középpont és az előző kör távolságát.

A harmadik kör generálása analóg módon történik az előzőhöz képest, figyelembe véve a második kör középponttól való távolságát is.

Ezekkel a lépésekkel megadtuk egy kiindulópontot, ahhoz hogy eljussunk a megoldásig. Az első jelenleg a legjobb megoldás. A továbbiakban fogjuk alkalmazni a szimulált lehűtés algoritmusát, hogy közelebb kerüljünk az optimális megoldáshoz.

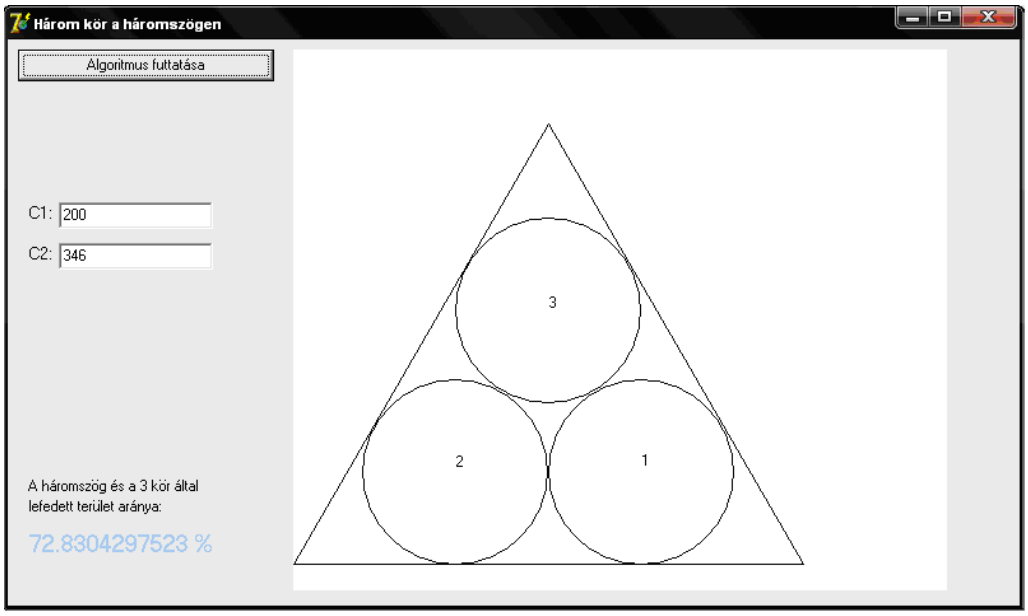
Úgynevezett „véletlen bolyongással” módosítjuk a körök középpontját és újra számoljuk a lehetséges sugarakat természetesen úgy, hogy a

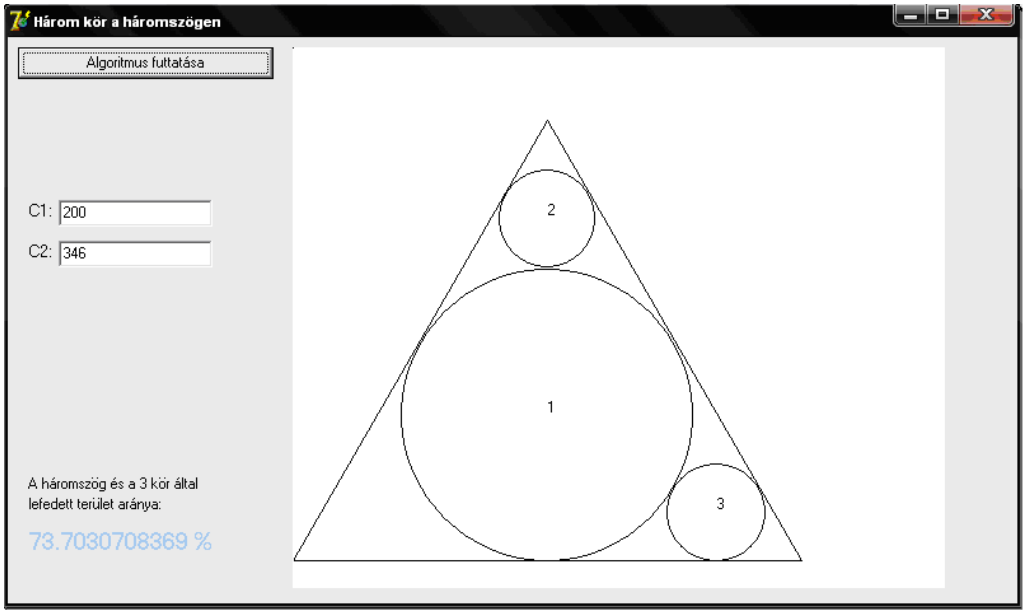
kiindulásnál megadott feltételek teljesüljenek. A köröknek a háromszögbe kell esniük és nem szabad fedniük egymást.

A szoftver egy *t* hőmérsékletnek megfelelően, mely 2-től indul módosítja először az első kör adatait, majd a rendre a többi kör középpontjait és sugarait. Minden egyes változtatás után mindig elfogadjuk a pozitív előrelépést, és bizonyos valószínűséggel fogadjuk el az esetleges „rosszabb" megoldásokat. Miután megtörtént mindhárom körön a változtatás, akkor csökkentjük a *t* értékét. A kapott megoldást összehasonlítjuk és ha nagyobb az összterülete a köröknek, mint a *final* tömb típusú változóban, akkor eltátoljuk és elkezdjük elölről az egészet. Inicializálunk, újra generálunk három kört és elvégezzük rajtuk a módosításokat, majd a kapott megoldást eltároljuk a *final* változóban, ha az jobb, mint az előző.

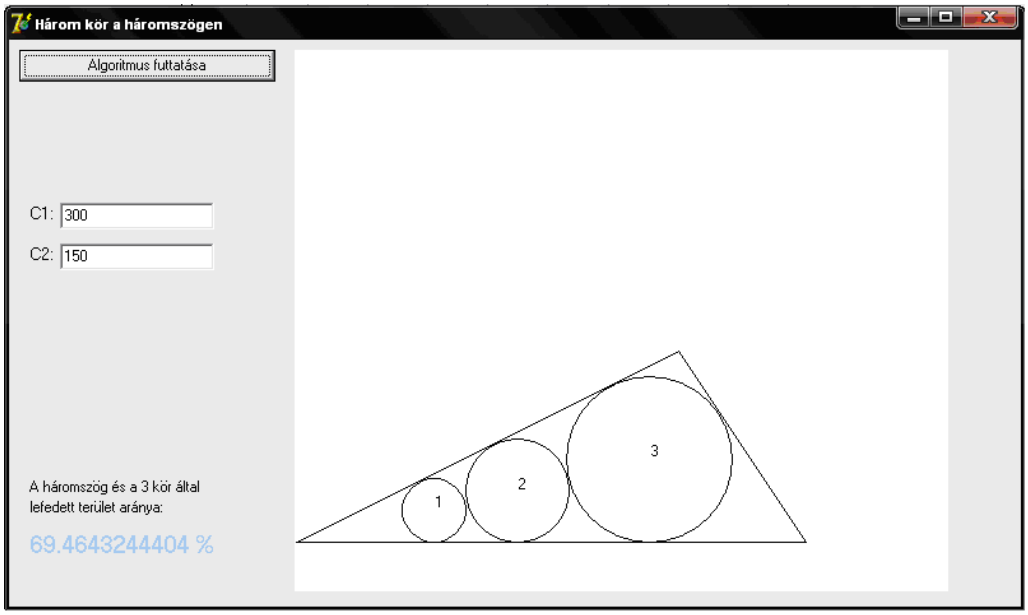
Nem maradt már más hátra csak a végső, legjobb megoldás kirajzolása és egy ellenőrző fájlba kiíratjuk a háromszög és körök adatait. Ezt részben a program megírása közben használtam ellenőrzésképpen.

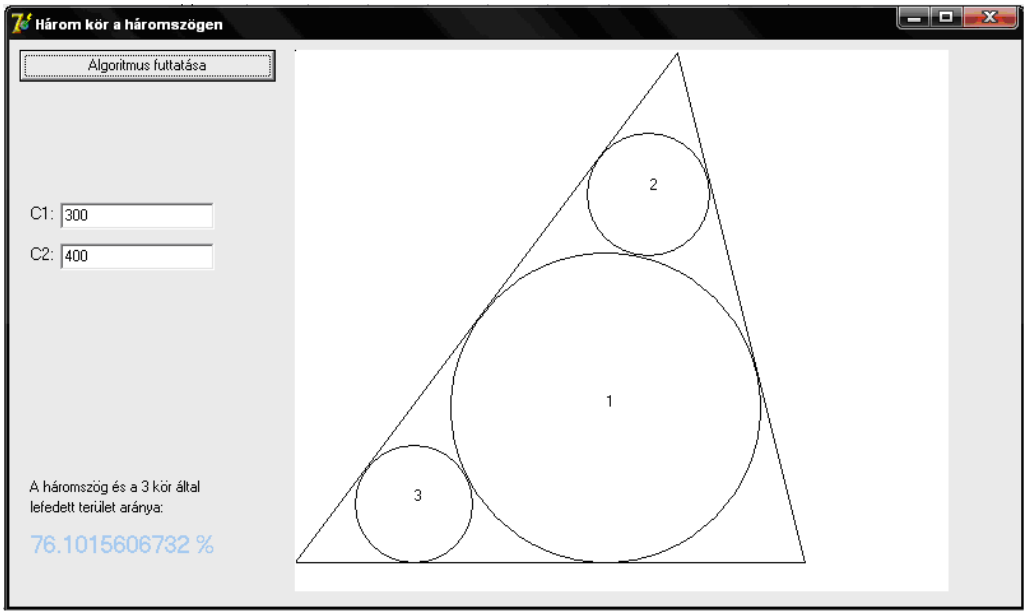
Példa egy szabályos háromszögben arra, hogy a Malfatti-körök nem a legnagyobb területet fedik le egy háromszögben:





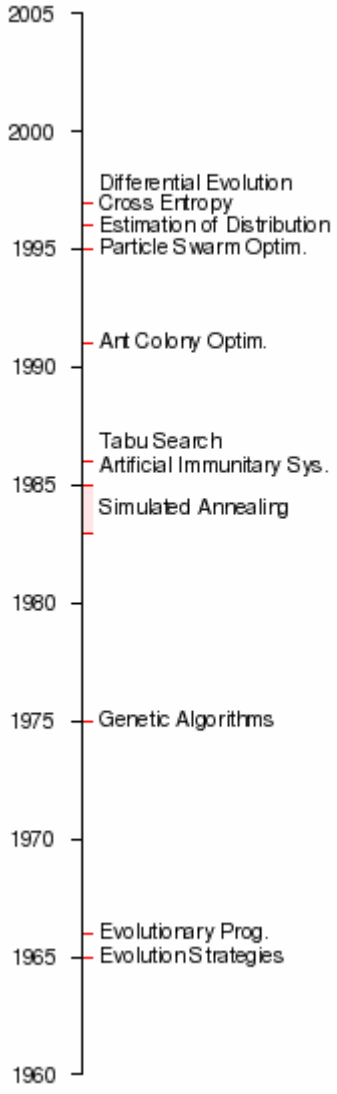
Futási képek a programból:





6. Összegzés

Napjainkban a nagy keresési térrel rendelkező problémák megoldása sok tárhelyet és időt igényel, mindazok mellett, hogy a számítástechnika és a számítástechnikai eszközök robbanásszerűen fejlődnek. Ezért van szükség fejlett kereső algoritmusokra. Ezek a „természetből vett ötletek” még gyerekcipőben járnak, még a kezdeti formulák szolgálnak alapul. A hagyományos keresési eljárásokhoz képest hatékony, terület-független keresést tudunk megvalósítani, így nagyon sokféle területen alkalmazható. Az általam írt programban a feladatra specializálódott szimulált lehűtés algoritmust használtam fel.



Lehetőség nyílt arra, hogy szemléletesen lássuk, annak ellenére, hogy a dolgozatban bemutatott módszerek egymástól független természeti folyamatok modelljei, valamilyen kapcsolat van közöttük. A közös a genetikus algoritmusok. Sok lehetőség rejlik még ezekben az algoritmusokban, felhasználási területük nincs még teljesen kiaknázva. A természetben lezajló folyamatok körülvesznek bennünket mindig, de a figyelem fókuszába 1975-től léptek be. A párhuzamos programozás területén is nagy jelentőségük lehet, mellyel még gyorsabbá és pontosabbá tehetők. Ezek a ránézésre igen egyszerű algoritmusok jól megadott paraméterek esetén könnyen, gyorsan, optimálishoz közeli megoldásokat adnak. Sok esetben nem célunk az optimális megoldás megtalálása, elég egy bizonyos határértéken belüli megoldás megtalálása, és ezen területek egyenlőre ezeknek az algoritmusoknak az erőssége.

A dolgozatban egy máig megoldatlan geometriai problémára – egy adott háromszögben mely az a három, egymás nem átfedő kör, mely maximális területet fed le a háromszögből – kerestem a legjobb megoldást. A szimulált lehűtés algoritmust használtam fel és egy általam fejlesztett program segítségével történt a keresés és a szemléltetés egyaránt.

A program adott esetekben cáfolja azt a hipotézist, mely szerint a Malfatti-körök fedik le a legnagyobb területet egy háromszögben. Speciális háromszögekben közel vannak a Malfatti-körök az optimálishoz, de találhatók nagyobb összterületű körök.

7. Irodalomjegyzék

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest: Algoritmusok, Budapest, Műszaki Könyvkiadó, 2003

Marco Cantù: Mastering Delphi 7, Sybex Inc., 2003

Lovász L., Gács P.: Algoritmusok, Budapest, Műszaki Könyvkiadó, 1978

Papadimitriou, C. H.: Számítási bonyolultság, Budapest, Novadata, 1999

Álmos Attila, Győri Sándor, Horváth Gábor, Várkonyiné Kóczi Annamária: Genetikus Algoritmusok, Budapest, Typotex, 2002

Globális optimalizálás:

http://en.wikipedia.org/wiki/Global_optimization

Szimulált (le)hűtés:

http://en.wikipedia.org/wiki/Simulated_annealing

Ezúton szeretnék köszönetet mondani témavezetőmnek, ***dr. Aszalós László*** *egyetemi adjunktusnak, a szakdolgozatom elkészítéséhez nyújtott hasznos tanácsaiért és készséges segítségnyújtásáért.*