WEB ALKALMAZÁS FEJLESZTÉSE JAVAFX KÖRNYEZETBEN OPTIMÁLÁSI PROBLÉMÁK MEGOLDÁSÁRA

DEVELOPING A WEB APPLICATION IN JAVAFX ENVIRONMENT TO SOLVE OPTIMIZATION PROBLEMS

*Marcsák Gábor Zoltán**

ABSTRACT

Despite the rapid progress of computer technology, there are still a lot of problems that cannot be solved only by raw computational power. The Travelling salesman problem is a bright example of the nonlinear programming, NP problem class. The utilization of informed search (so-called "heuristic") algorithms is a possible solution in these situations. On the one hand, compared to non-heuristic algorithms, heuristic strategies require much less computation in order to find potential solutions. In many cases, very difficult problems can be solved with them. On the other hand, they have a disadvantage. There is no guarantee that the optimal solution was found. My goal was to develop an advanced web application, which is a standalone collection of modern heuristic algorithms.

1. BEVEZETÉS

A számítógépek rohamos fejlődése ellenére még mindig sok olyan feladat ismert, mely nem megoldható pusztán a számítási teljesítményre alapozva. Az utazó ügynök problémája egy ékes példája az NP-nehéz bonyolultságú problémaosztálynak. Adott egy ügynök, illetve városok halmaza. Az ügynök körutat szeretne tenni oly módon, hogy minden várost pontosan egyszer érintve az útiköltség minimális legyen. A megoldás során a problémát az okozza, hogy a városok számának növekedésével a különböző lehetséges körutak száma faktoriálisan növekszik. Már viszonylag kevés, 120 darab város esetén is megközelítőleg $6{,}69*10^{198}$ lehetséges megoldásból kell kiválasztani az optimálist. Nyilvánvaló, hogy nagy számú város esetén az összes lehetséges permutáció vizsgálata a tudomány jelenlegi állása szerint belátható időn belül nem kivitelezhető.

A megoldást az informált kereső, úgynevezett heurisztikus algoritmusok jelentik. Előnyük, hogy a lineáris módszerekhez viszonyítva jóval kevesebb számítással képesek eljutni a megoldáshoz. Nagy bonyolultságú feladatoknál is sok esetben sikerrel alkalmazhatók. Legfontosabb hátrányuk azonban, hogy nem garantálható teljes bizonyossággal az optimális megoldás megtalálása [1]. A különböző heurisztikus algoritmusok a tudomány számos területén korábban megoldhatatlannak vélt problémák megoldásával bizonyították létjogosultságukat. A célom egy olyan korszerű web alkalmazás kifejlesztése volt, mely egyedülálló gyűjteménye a modern heurisztikus algoritmusoknak.

2. A JAVAFX KERETRENDSZER

A heurisztikus algoritmusok gyűjteményéhez mindenekelőtt egy keretrendszert kellett fejlesztenem, mely tárolja, rendszeri azokat. A program tervezésekor több fontos szempontot vettem figyelembe.

- Egy, az Interneten bárki számára könnyen és gyorsan hozzáférhető web alkalmazást szerettem volna létrehozni, amit nem kell külön telepíteni, böngészőből használható.
- A legtöbb heurisztikus algoritmus implementációja sokkal egyszerűbb modern, objektum orientált paradigmát követő programozási nyelvek segítségével. Annak ellenére, hogy a nem informált kereső eljárásokhoz viszonyítva kevesebb számítás árán jutnak eredményre, bonyolult problémák esetén fontos a választott technológia futási sebessége.
- Alapvető követelmény a könnyen kezelhető, informatív felhasználói felület. A beépített heurisztikus algoritmusok története, elméleti leírása és pszeudokódja egyaránt elérhető kell legyen. A működésük megértése azonban gyakorlati példákkal szemléltetve sokszor egyszerűbb, ehhez is ki kellett dolgoznom a megfelelő programrészeket.
- Fontos volt továbbá, hogy az algoritmusok gyűjteményét könnyen bővíteni lehessen. A keretrendszer ennek megfelelően egyfajta moduláris struktúrát követ. Az új tartalom ezáltal könnyen illeszthető a már meglévőhöz.

* *logisztikai mérnök MSc hallgató, Miskolci Egyetem*

A fent vázolt szempontok figyelembevételével a viszonylag újnak számító JavaFX technológiát választottam a megvalósításhoz. Egyik legfontosabb előnye a Java technológiából eredő platformfüggetlenség, mely lehetővé teszi használatát Windows, Mac OS X és Linux rendszereken. A közeljövőben várhatóan az Android és iOS mobil operációs rendszerek is támogatottá válnak, ezzel lehetővé téve a technológia használatát okostelefonokon és táblagépeken. A Java egy széles körben elterjedt programozási nyelv, mely megbízható, jól dokumentált és gazdag beépített függvénykészlettel rendelkezik. A JavaFX programok asztali alkalmazásként és web alkalmazásként egyaránt képesek működni, a két mód között pedig nincs szignifikáns sebességbeli különbség, mivel futtatásukról ugyanaz a Java Virtuális Gép (JVM) gondoskodik.

3. BEÉPÍTETT HEURISZTIKUS ALGORITMUSOK

Újabban előtérbe kerültek azok a heurisztikus algoritmusok, melyek működési elve valamilyen természeti jelenségen alapul, például Darwin evolúciós elméletén. A különböző élőlények adott helyzetben való viselkedésmintájából merítve jó eredményeket lehet elérni, ennek analógiájára dolgozták ki többek között a madár és halrajok mozgásán (Particle Swarm), baktérium (Bacterial Foraging) és hangyakolóniák (Ant Colony) táplálkozásán, vagy a szentjánosbogarak fényjelenségein (Firefly) alapuló módszereket. A cikk írásakor három természeti jelenségen alapuló evolúciós algoritmus érhető el az alkalmazásban, Java nyelven implementált forráskódjuk szintén megtekinthető, letölthető.

3.1. Particle Swarm algoritmus

A Particle Swarm (részecske-csapat) algoritmust 1995-ben dolgozták ki. Működését a madár és halrajok mozgása inspirálta. A nagyszámú egyed mozgásában egyfajta rendezettség figyelhető meg. Érzékelik egymás helyzetét, bizonyos mértékben pedig emlékeznek a korábbi pozíciókra. A keresés adott számú részecske létrehozásával kezdődik, amik véletlenszerű kiindulási pontokban helyezkednek el. A részecskék tisztában vannak aktuális és addigi legjobb pozíciójukkal, valamint a részecske csapat addigi legjobb pozíciójával. Hogy az adott pozíció mennyire jó az adott keresés szempontjából, mindig egy fitness függvénnyel határozható meg. Az iterációk során az alábbi képletek alapján számítható ki a részecskék új pozíciója (1), (2):

$$V_{id} = w \times V_{id} + c_1 \times rand() \times (p_{id} - x_{id}) + c_2 \times rand() \times (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + V_{id} \quad (2)$$

V_{id} részecske sebessége
w gyorsulás konstans
c_1 egyéni súlyozás konstans
c_2 szociális súlyozás konstans
p_{id} részecske eddigi legjobb pozíciója
p_{gd} csapat eddigi legjobb pozíciója
x_{id} részecske aktuális pozíciója

Az iterálás addig folytatódik, míg nem teljesül valamilyen leállási feltétel. Evolúciós algoritmusok esetében sokszor nem egyszerű egyértelműen jó leállási feltételt találni. A Particle Swarm módszernél a feltétel többek között lehet megszabott számú iteráció végrehajtása, vagy ha a csapat legjobb pozíciója bizonyos számú iteráció után sem változott meg [2].

3.2. Differential Evolution algoritmus

A Differenciális Evolúció az élőlények genetikus öröklődésén, mutációján alapul. A részecske-csapat módszerhez hasonlóan először adott számú (N) kiinduló egyedet hoz létre. Az iterációk során adott generáció ($_G$ index) minden tagjának egy új leszármazott egyedet ($_{G+1}$ index) hoz létre keresztezéssel az alábbi képlet alapján:

$$\underline{x}_{i,G+1} = \underline{x}_{r_1,G} + F(\underline{x}_{r_2,G} - \underline{x}_{r_3,G}) \quad (3)$$

$\underline{x}_r1, \underline{x}_r2, \underline{x}_r3$ véletlenszerűen kiválasztott egyedek
F mutációs konstans

Ezután összehasonlítja az eredeti és a leszármazott egyed rátermettségét az adott probléma szempontjából. A rátermettséget a fitness függvény alapján határozzuk meg. A kedvezőbb megoldást adó egyed lesz tagja a következő generációnak [3].

3.3. Bacterial Foraging algoritmus

A 2002-ben megjelent algoritmus viszonylag újnak számít a természeti jelenségeken alapuló metaheurisztikus stratégiák családjában. Az E. coli baktériumkolóniák táplálékkereső és reprodukciós viselkedésmintáin alapul a működése. Az Escherichia Coli az ostoros baktériumok családjába tartozik. Ostorszerű végtagjai, az úgynevezett flagellumok segítségével képes az önálló úszásra. Ez a mozgás a kemotaxis, mely során a fő cél a tápanyagban gazdag helyek elérése.

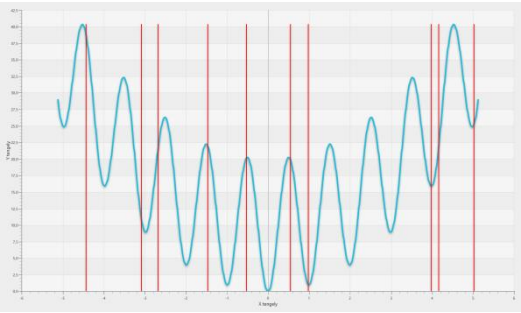
A Bacterial Foraging lényegében ezt a kemotaxist másolja a keresés során. A kolóniához tartozó baktérium egyedek a Particle Swarm algoritmushoz hasonlóan kommunikálnak egymással, ismerik a kolónia által megtalált legjobb pozíciót. Szintén evolúciós algoritmusról van szó, az egyedek meghatározott élettartammal rendelkeznek. Minél jobb pozícióban vannak az adott probléma szempontjából, annál kevésbé csökken az élettartamuk. Minden iteráció során az

algoritmus rendezi az egyedeket élettartam szempontjából, majd a halmazt középen ketté bontva az idősebb baktériumok helyére átmásolja a fiatalabb példányokat, ezzel szimulálva az osztódást. A három ismertetett heurisztikus algoritmusból véleményem szerint a Bacterial Foraging a legbonyolultabb. A módszer részletes matematikai leírása megtalálható a [4], [5] forrásművekben.

4. FÜGGVÉNYEK GLOBÁLIS SZÉLSŐÉRTÉKÉNEK KERESÉSE

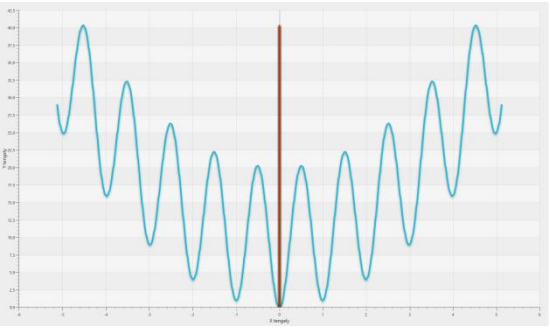
A heurisztikus algoritmusok működésének gyakorlati szemléltetésére egy és kétváltozós függvényeken végezhető globális minimum és maximum keresést építettem a programba. A felhasználó választhat az öt beépített tesztfüggvény (De Jong's, Rosenbrock's valley, Rastrigin's, Schwefel's, Easom's) közül, vagy sajátot is definiálhat [6].

Egyváltozós függvény esetén a keresés egy grafikonon követhető nyomon. Az 1. ábrán Particle Swarm algoritmussal, 10 darab részecske segítségével végeztem minimum keresést a Rastrigin's tesztfüggvény egyváltozós változatán. A függvény abszolút minimuma az x=0 pontnál van. Az algoritmus első lépése a részecskék létrehozása véletlenszerű pozíciókban.



1. ábra Particle Swarm keresés indítása

Leállási feltételként 200 darab iteráció végrehajtását határoztam meg. Az eredmény a 2. ábrán látható.



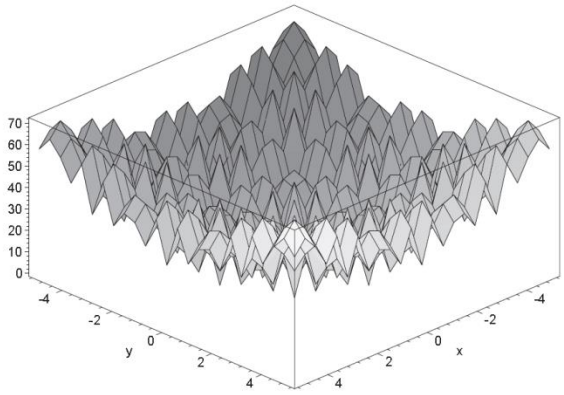
2. ábra Particle Swarm keresés vége

Az algoritmusok működési paramétereit a felhasználó minden részletre kiterjedően állíthatja a megfelelő vezérlők segítségével.

Kétváltozós függvények esetében bonyolultabb a keresés szemléltetése, mivel az ábrázoláshoz 3 dimenziós felület leképezése, vagy speciális grafikon szükséges. A program továbbfejlesztésének egyik fő iránya ezen funkció megvalósítása lesz.

5. AZ ALGORITMUSOK TESZTELÉSE

Az algoritmusok teszteléséhez a Rastrigin's tesztfüggvény kétváltozós változatát választottam, mely a 3. ábrán látható:



3. ábra A kétváltozós Rastrigin's függvény [6]

A tesztfüggvénynek az x=0.0, y=0.0 helyen van globális minimuma, melynek értéke 0.0. A teszt célja a következő kérdések megválaszolása volt:

- Milyen hatékonysággal találják meg az algoritmusok az optimális megoldást?
- Ha mindhárom algoritmus esetén azonos számú populációt adok meg, valamint leállási feltételként is ugyanazon iterációs korlátot kapják, akkor megállapítható, hogy melyik algoritmus működik a leggyorsabban. A leggyorsabb működés természetesen nem feltétlenül jelent egyúttal leghatékonyabbat is.
- Fejlettebb leállási feltétel definiálásával milyen mértékben növelhető a hatékonyság?

	Idő [ms]	*x*	*y*	Fitness
PSO	26	$-7{,}92*10^{-10}$	$2{,}27*10^{-11}$	$1{,}40*10^{-9}$
PSO*	18	$3{,}82*10^{-10}$	$-6{,}94*10^{-10}$	$8{,}25*10^{-9}$
DE	30	$5{,}17*10^{-6}$	$-1{,}25*10^{-7}$	$4{,}43*10^{-6}$
BFO	43	$3{,}41*10^{-6}$	$2{,}18*10^{-5}$	$9{,}63*10^{-5}$

1. táblázat Teszteredmények

A tesztelés során minden esetben 100-as nagyságú populációkkal dolgoztam. Leállási feltételként egy kivételtől (PSO*) eltekintve a 400 iterációs határt szabtam meg. Minden keresést háromszor ismételtem meg, majd a feljegyzett eredmények átlagát tüntettem fel a táblázatban.

Az algoritmusok hatékonyságának elemzéséhez először meg kell határozni, hogy milyen pontosságra van szükségünk. Ha négy tizedesjegy pontosság megfelelő, akkor megállapítható, hogy mindegyik algoritmus megtalálta az optimális megoldást. Ellenben ha például nyolc tizedesjegy pontosságú keresésre van szükség, akkor a Differenciális Evolúció és a Bacterial Foraging esetében kiválóan megfigyelhető a heurisztikus algoritmusok gyenge pontja. Nagyon közel vannak az optimális megoldáshoz, de nem azt találták meg. Vélhetően nagyobb populáció vagy iterációszám javítana a pontosságon.

A leggyorsabb működést a Particle Swarm produkálta, tehát iterációnként a PSO-nak van a legkisebb számításigénye a vizsgált algoritmusok közül. Érdekes módon egyben a leghatékonyabb algoritmus is a részecske-csapat volt az adott tesztkörülmények között.

A táblázatban szereplő PSO* esetében leállási kritériumként iterációs határ helyett a feltétel az volt, hogy valamennyi részecske legjobb pozíciója egyezzen meg a globális legjobb pozícióval. Ez átlagosan 307 iteráció után következett be. A táblázatból leolvasható, hogy a 400 iterációs határhoz képest ez 23,25%-al kevesebb számítást jelent, ami pontosan megmutatkozik a futási időkben is.

A heurisztikus algoritmusok tesztelése egy igen nehéz és összetett folyamat, mivel a sok paraméter és nagyszámú változó gyakorlatilag végtelen variációs lehetőséget jelent.

6. ÖSSZEFOGLALÁS

Cikkemben a heurisztikus optimáló algoritmusok működését, és a használatukban rejlő sokszínű lehetőségeket vizsgáltam. Véleményem szerint a tudomány és technológia fejlődésével számos új területen merül majd fel az igény a nagy bonyolultságú, hagyományos eszközökkel megoldhatatlannak tűnő számítási feladatok elvégzésére. Ez az a problémakör, ahol a metaheurisztikus optimáló algoritmusok a leginkább erősek.

Célom, hogy tovább bővítsem a web alkalmazásba épített algoritmusok számát, valamint olyan új gyakorlati példákkal szemléltethessem működésüket, mint az utazó ügynök probléma megoldása, különböző szerkezetoptimalizálási, logisztikai, operációkutatási feladatok. A web alkalmazás a http://users.iit.uni-miskolc.hu/~marcsak/ címen érhető el.

7. KÖSZÖNETNYILVÁNÍTÁS

A bemutatott kutató munka a TÁMOP-4.2.1.B-10/2/KONV-2010-0001 jelű projekt részeként az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg"

8. IRODALOM

[1] JÁRMAI K., IVÁNYI M.: Gazdaságos fémszerkezetek analízise és tervezése, Műegyetemi kiadó, Budapest, 2001. ISBN 963 420 674 3

[2] HU X., EBERHART R.: Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization. In Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002), volume 5. Orlando, USA, IIIS, July 2002.

[3] STORN R.: On the Usage of Differential Evolution for Function Optimization, 1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS 1996), Berkeley, pp. 519–523. IEEE, New York, NY, USA.

[4] DAS S., BISWAS A., DASGUPTA S., ABRAHAM A.: Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications http://tutorial.softcomputing.net/bfoa-chapter.pdf

[5] LIU Y., PASSINO K. M.: Biomimicry of Social Foraging Bacteria for Distributed Optimization: Models, Principles, and Emergent Behaviors JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 115, No. 3, pp. 603–628, December 2002

[6] MOLGA M., SMUTNICKI C.: Test functions for optimization needs, 2005, http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf