A MISKOLCI EGYETEM HABILITÁCIÓS FÜZETEI
INFORMATIKAI TUDOMÁNYOK TUDOMÁNYÁGI HABILITÁCIÓS BIZOTTSÁG

ALKALMAZOTT INFORMATIKAI KUTATÁSOK: MODELL OPTIMALIZÁLÁS ÉS MESTERSÉGES INTELLIGENCIA ALKALMAZÁSOK

Tudományos munkásság áttekintő összefoglalása

Készítette:

DR. NEHÉZ KÁROLY RÓBERT
egyetemi docens
aki a

Informatikai Tudományok Tudományágban
„dr. habil.” cím elnyerésére pályázik

Miskolc
2024

TARTALOMJEGYZÉK

1 ELŐSZÓ

1997-ben Gépészmérnöki oklevelet informatika szakirányon szereztem, majd doktoranduszként az ME Szerszámgépek tanszékre kerültem. A 1997-2000 között végzett doktori tanulmányaimat követően a „Marás számítógépes szimulációja és optimalizálási kérdései” című PhD értekezésemet 2003-ban védtem meg summa cum laude minősítéssel. A fokozatszerzés után folytattam a tudományos tevékenységet az alkalmazott informatika szerteágazó területein.

Az elmúlt 20 évben informatikai algoritmusok fejlesztésével foglalkoztam főként műszaki, termelésinformatika területeken. A tézisfüzet öt tématerület rövid összefoglalója, melyek kiindulópontjait a tanszék és az egyetem számos sikeresen megoldott korábbi K+F munkái adták. Mindig fontosnak tartottam, hogy az elvégzett K+F munkák eredményeit, tudományos igényességgel publikáljuk, vagy akár szabadalmaztassunk egy jól működő gyakorlati módszert.

Az évek során elvégzett tudományos munkám áttekintő összefoglalásaként az alábbi öt tématerületet ismertetem:

1.) Készletgazdálkodási modellek: ebben a részben a készletgazdálkodási modellekkel, különösen a klasszikus *újságárus* (newsvendor) modellel és annak speciális kiterjesztéseivel foglalkozom. A modellek célja: bizonytalan rendelés-előrejelzésekre alapozva, hogyan lehet optimális készletezési és gyártási politikát meghatározni. A fejezet tartalmaz egy újszerű analitikai megközelítést, a *büntető költségek* elemzését, diszkrét hagyományos modellek bemutatását, számítógépes szimulációkat a politikák ellenőrzésére, és összefoglalom a fajlagos költségmodelleket és a termékkifutás modellezésének kérdését is.

2.) Modern feladat kiértékelők: ez a terület a MOOC-okat (Massive Open Online Courses) és az online programozás oktatását és feladatkiértékelési mutatja be. A projekt keretében testre szabtam egy saját MOOC rendszert, amit a mai napig használunk az oktatásban. A száraz tananyagok mellett rugalmas feladat kiértékelő modult fejlesztettem, aminek kiértékelési módszere nemcsak teljes programok, hanem részfeladatok, függvények finomabb kiértékelésére, feladatgenerálásra is alkalmas. A fejezet részletezi a publikált eredményeimet a hallgatók által írt programok automatikus kiértékelési módszereivel kapcsolatban.

3.) Egyes rajzi elemek automatikus felismerése: ebben a részben az optikai karakterfelismerés (OCR) és a kapcsolódó módszerek áttekintése található, valamint bemutatok egy saját fejlesztésű speciális eljárást a hagyományos OCR-ek által nehezen használható esetekre. A kifejlesztett módszert benyújtott magyar szabadalom védi. Az összetett algoritmus egy sikeres többfelhasználós interaktív alkalmazásban lett megvalósítva, ami a modern autóipari termékfejlesztésben használt géprajzok egyes rajzi elemeinek utólagos kinyerésének és adatfeldolgozásának megkönnyítésére szolgál.

4.) Mesterséges intelligencia módszerek a szoftverek minőségbiztosításában: ez a fejezet adatbányászati módszereket mutat be a szoftverhibák előrejelzéséhez. A szoftverhibák (közismert nevén bug-ok) és a "code smell"-ek jelenlétének előrejelzése a folyamatosan fejlesztett programkódokban, nagy jelentőséggel bír a minőségbiztosítási folyamatokban. A problémát először klasszikus módon, majd később modern neurális hálókkal is megvizsgáltam. A fejezetben tárgyalom a modellalkotás és kiértékelés folyamatát, valamint a kapcsolódó publikációkat és irodalmat is bemutatom.

5.) Hibrid evolúciós algoritmusok a termelésütemezési feladatokban: a termelésütemezési feladatok optimalizációja az ME Alkalmazott Informatikai Tanszékének évtizedes múlttal rendelkező kutatási területe. A fejezetben a klasszikus flowshop ütemezési feladat optimális átfutási idejének meghatározását, a bakteriális memetikus - (DBMEA), a Monte-Carlo fakeresés -, és a saját fejlesztésű DBMEA algoritmus keresztezéséből létrehozott hibrid módszer segítségével mutatom be. Az új megközelítéssel a nemzetközi benchmark teszteken néhány esetben hatékonyabb ütemezési sorrendet lehet elérni.

2 Készletgazdálkodási modellek

2.1 Bevezetés

Ebben a fejezetben a 2003 és 2008 közötti időszak legfontosabb kutatási területét foglalom össze. Manapság a fejlett országok gazdaságpolitikájának fontos célja a fenntartható fejlődés fenntartása. Az árutermelés, mint a gazdaság egyik fontos területe, a fejlett országokban a magas GDP és a jólét alapjává vált a tömeggyártás sikereinek következtében. A piacon a tömegtermékek iránti kereslet magas, új igények jelennek meg, az életciklusok egyre rövidebbek, a divatos formák és speciális csomagolások iránti kereslet nőtt. A vállalatok egyre inkább komponensekből szerelik össze termékeiket, és a beszállítók pedig fontos szerepet játszanak ebben a folyamatban. A globális üzleti környezetben a hatékony és sikeres vállalatok számára elengedhetetlen a beszállítói láncokkal való szoros együttműködés. Az informatikai fejlődés hozzájárul az együttműködés szorosabbá tételéhez. A korábbi egyszerű „vásárló – eladó" viszony egyre inkább kooperatív, együttműködő kapcsolatokká válik. A gyorsuló világban, informatikai rendszerek alkalmazása nélkül a vállalatok közötti valós idejű együttműködés szinte lehetetlen.

A tömeggyártásban, a beszállítói láncok összetettek és hosszabb folyamatokból állnak, amelyekben a késések és más váratlan, sztochasztikus jelenségek komoly kockázatot jelentenek. Ezért a vállalatoknak szükségük van hatékony és pontos döntéstámogató megoldásokra. Az informatikai fejlődés lehetővé tette a nagy, integrált vállalatirányítási rendszerek alkalmazását, amelyek hatékonyan támogatják a beszállítói láncok tervezését és irányítását. A kutatás célja a beszállító és végtermék gyártó kapcsolatának vizsgálata a kollaboratív beszállítói készletezési politika kontextusában. A beszállítói és végtermék gyártói kapcsolatok összetettek és változók, ezért széles körű elemzés és modellezés szükséges [1]. A piaci ingadozások és bizonytalanságok miatt a készletgazdálkodás és a pontos szállítás kritikus szerepet játszanak. Célunk volt a hatékony beszállítói készletezési politikák kidolgozása a végtermék gyártói igények kielégítése mellett.

Beszállító és végtermék gyártó közötti kapcsolat összetett és a készletgazdálkodás egyik fontos elemét képezi ennek a viszonynak. Gyakorlatban a végtermék gyártó szükséglet előrejelzései és a konkrét lehívások természetszerűleg különböznek, ami bizonytalanságot okoz. Az optimális beszállítói készletezési politika kialakítása a közös együttműködési viszonyok figyelembevételével történik. A kutatás kiemelte a készletek időbeli menedzselését, valamint a végtermék gyártó és beszállító kötelezettségeit. Feltételeztük, hogy az alapanyagok és gyártókapacitások rendelkezésre állnak. A kutatás alapvetően azt vizsgálta, hogyan lehet hatékonyan irányítani a beszállítói készleteket a végtermék gyártói igények kiszolgálása és a vállalati célok megvalósítása érdekében. A beszállítói kapcsolat összetett, és az optimális készletgazdálkodási politika kidolgozása fontos szerepet játszik a hatékony együttműködésben [2].

A hatékony készletgazdálkodási modellek kidolgozása során számos fontos lépést érdemes figyelembe venni. Elsőként elengedhetetlenül fontos megismerkedni az anyag beáramlásának és kiáramlásának folyamataival, valamint a készletek mozgására jellemző sajátosságokkal. Ez lehetővé teszi a későbbi modellezés során a valóságnak megfelelőbb képet. A modell kidolgozásának további lépéseiben kiemelt figyelmet kell fordítani a vizsgálatba bevonható költségelemek tisztázására. Ennek során mélyebb betekintést nyerhetünk a készletgazdálkodással összefüggő költségek összefüggéseibe és jelentőségébe.

Ezután elérkezik a célkitűzések meghatározásának ideje. Fontos rögzíteni az elvárt eredményeket vagy célokat, amelyeket a kialakítandó készletgazdálkodási modell segítségével kívánunk elérni. Ez a lépés irányt szab a további modellezési folyamatnak. A modellezés során matematikai összefüggéseket kell felállítani, ami összekapcsolja a folyamatok változóit, a korlátozó feltételeket és a célfüggvényeket. Ezek az összefüggések szolgáltatják a modell alapját, és segítenek az optimalizálási folyamatban. A készletgazdálkodási politika meghatározása döntési változók segítségével történik. Ezen döntési változók értékeit olyan eljárással kell kiszámítani, amely a kitűzött célokat és költségstruktúrát összességében figyelembe véve hatékony megoldást nyújthat. [5]

A probléma hatékony megoldása érdekében megfelelő megoldó algoritmust kell választani, amely képes a matematikai modell alapján a szükséges számítások hatékonyan elvégezéséhez. Az algoritmus kiválasztása kulcsfontosságú a hatékony és pontos eredmények elérésében és informatikai szempontból is ez a pont jelenti a legnagyobb kihívást. Továbbá az információs folyamat támogatására szolgáló szoftver is kritikus szerepet játszik. Ez teszi lehetővé az adatok kezelését, a számítások végrehajtását és az eredmények vizualizációját. Ennek segítségével könnyebben monitorozható és optimalizálható a készletgazdálkodási folyamat. Végül, de nem utolsósorban, az elkészült modellt integrálni kell a teljes vállalatirányítási rendszerbe. Ennek révén a készletgazdálkodási döntések és eredmények szervesen beépülnek a vállalat szélesebb körű működésébe és stratégiájába. Összességében elmondható, hogy a készletgazdálkodási modellek kialakítása matematikai, üzleti, informatikai és szoftverfejlesztési módszereket egyaránt igényel.

2.2 A klasszikus újságárus (newsvendor) modell és kiterjesztése

A beszállítói láncok tudományos irodalmában korán kiemelt figyelmet kap az ún. klasszikus újságárus modell [3][4], amely a sztochasztikus készletgazdálkodás elméletének fontos alapját képezi. Ebben a modellben egy újságárus viselkedését vizsgáljuk, aki különféle napilapokat rendel a vásárlóknak, előre meghatározott gyakorisággal. A beszerzési döntés során azt kell eldöntenie, hogy egy-egy napon mennyi újságot rendel, hogy a várható nyeresége maximalizálódjon, ugyanakkor a vásárlók elégedettsége is a lehető legmagasabb szinten legyen, azaz ne fogyjon el túl hamar a napi készlete. Az optimális rendelési mennyiség meghatározása kihívást jelent egy napilap árusnak, hiszen túlzott mennyiség beszerzése esetén az eladatlan újságok veszteséget okoznak, míg alacsony beszerzési darabszám esetén a potenciális vásárlók

üres kézzel távoznak. A klasszikus újságárus modell több periódusra való első alkalmazása Herbert Scarf nevéhez fűződik, aki a klasszikus modell egy periódusra vonatkozó költségfüggvényét alkalmazta [3]. Koncepciója szerint a megoldást valamilyen dinamikus programozási módszer segítségével kell keresni. Az informatikai hardver eszközök rohamos fejlődése napjainkban már lehetővé teszik az egyre bonyolultabb megközelítések és heurisztikák alkalmazását is.

2.2.1 Analitikus megközelítés

A sikeres beszállítói készletezési modellek kialakításakor kritikus és optimális raktárkészletek meghatározása mellett az optimális raktározási politika döntései is hangsúlyosak. Ezek döntések sorozatként valósulnak meg és befolyásolják, hogy milyen mértékű készleteket kell tartani a tervezett kiszolgálási szint biztosítása érdekében vagy akár a hiánymentes szolgáltatáshoz. Emellett meghatározzák, hogy mikor és milyen mennyiségű terméket kell gyártani a készletek feltöltésére. Ezen döntések meghozatala minden beszállító cég számára elengedhetetlen. Ameddig a többlettermelés esetén számolni kell a készletek finanszírozási és tárolási többletköltségeivel, addig az alul teljesítés miatti büntetések kifizetéséből is többlet költségek merülhetnek fel. A büntetési költségek modellje különösen komplex, hiszen ezek a költségek akár a megrendelő bizalmának teljes elvesztéséig, a beszállítói szerződés felbontásáig is terjedhet. A készletezési modellek céljai változatosak lehetnek, és ezek segítségével mind a beszállító, mind a vevő érdekei érvényesülhetnek, vagy akár közös érdekeket is kifejezhetnek (együttes célfüggvények). Bizonyos esetekben előfordulhat, hogy szigorúan tilos a hiány, például a VMI (Vendor Managed Inventory) és rövid ciklusú JIT (Just-In-Time) beszállítói rendszereknél. A kialakított modell általában engedi a kis kockázatú hiányt, de a büntetési költségek beállításával, hangolásával szabályozható, hogy milyen gyakran fordulhatnak elő hiányok.

2.2.2 A büntető költség

A büntető költségek kifejezést a készletezési politika szempontjából a szakirodalom három különböző megközelítéssel használja:

- Az első értelmezés alapján a beszállító büntető költséget a ki nem elégített rendelésekre (lehívásokra) értelmezi, ami a végtermék-gyártónál jelentkező üzletveszteség részleges vagy teljes átvállalását jelenti. Ezen megközelítés a *hideg* vásárló-eladó viszonyban releváns.
- A második megközelítés szerint a beszállítói hiány minden esetben növeli a teljes termelési lánc költségeit, még akkor is, ha nincs üzleti veszteség. Ebben az esetben a végtermék gyártónak többlet belső tevékenységeket, késztermék készleteket, időpontátütemezéseket stb. kell bevezetnie a beszállítói hiány hatásainak kiegyenlítése érdekében. Ebben az esetben már egy *melegebb* (kooperatív) kapcsolatot feltételez a felek között.

- A harmadik megközelítés szerint nemcsak a belső rendelések teljesítésének elmulasztása jelent üzleti veszteséget, hanem a túlzottan nagy készletek is, a teljes termelési láncban, mivel a plusz készlet már nem értékesíthető. Ezt a veszteséget a feleknek közösen kell viselniük, ez a veszteség adódhat az előre jelző rendszer pontatlanságából. Ebben az esetben a felek között szoros üzleti, termelési és logisztikai kapcsolatot feltételez, ami hosszú távú érdekközösséget teremt, egyfajta *virtuális vállalatszerű* együttműködést.

A kialakított modellben a második típusú együttműködést feltételeztük. Az igényeket, amelyek a lehívási események alkalmával jelennek meg, időfüggőnek és valószínűségi változónak tekintjük. Ezen valószínűségi változók az előrejelzések alapján kiszámítható várható értékkel, szórással, valamint eloszlás függvényeik is ismertek (egyszerű esetben minden időpontban egyenletes eloszlást feltételezünk). Az igények előre meghatározottak és ismertek, és fix periodicitással érkeznek a beszállítóhoz. Itt rendelési periódus alatt a két lehívás között eltelt időt értjük, amely mindig a rendelkezésre álló legfinomabb előrejelzésre vonatkozik. A fix periódus mérete általában egy naptári hétnek felel meg a gyakorlatban. A gyártási ciklus alatt azokat a periódusokat értjük, amelyek két készletfeltöltés közötti időt reprezentálják. Fontos megjegyezni, hogy a bemutatott modellek periodikus jellegükből adódóan mindig az előrejelzések finomságának megfelelően adnak optimális megoldást. Emellett a vizsgált modellek négy előfeltétele a következő: 1.) A rendelések utánpótlása (gyártás és szállítás) azonnal történik. 2.) A beszállítóhoz érkező igény minden időperiódus végén valósul meg. 3.) Az átállás költségei minden beszerzési ciklus elején jelentkeznek. A legszélsőségesebb eset akkor áll fenn, ha egy ciklus egyetlen periódusból áll. Ekkor minden egymást követő periódusban jelentkezik ez a költség.

Több termék egyidejű gyártása/beszerzése esetén az átállások egymástól függetlenek és nem oszthatók meg a termékek között.

2.3 A modell bemutatása

Kiindulásként tekintsük a klasszikus újságárus modell fix költség taggal kiegészített változatát [4]. Az egy periódus lefedésére alkalmas modell költség alapú célfüggvénye a következő:

$$K(q) = c_f + c_v(q - I) + pE[D - q]^+ + hE[q - D]^+. \qquad (1)$$

ahol $[q - D]^+ = max(q - D, 0)$, $[D - q]^+ = max(D - q, 0)$. A függvényben c_f kifejezi, hogy minden sorozat gyártásának indítása valamilyen fix költséggel jár. Egy darab termék előállítási ára c_v. A modell döntési változója q, amely a raktáron lévő termékek darabszámát jelenti. I jelenti azt a kezdeti készletszintet, amivel a beszállító a periódus elején rendelkezik. A kumulált tárolási költségek paramétere h. A beszállítóhoz érkező igényeket D, tetszőleges eloszlású valószínűségi változó reprezentálja, valamint a ki nem elégített rendelés büntetését minden

termék után p jelenti. A (1) költségfüggvény alapján a megoldás egy szélsőérték számítási feladatként adódik, ahol cél a költség függvény minimumának megtalálása:

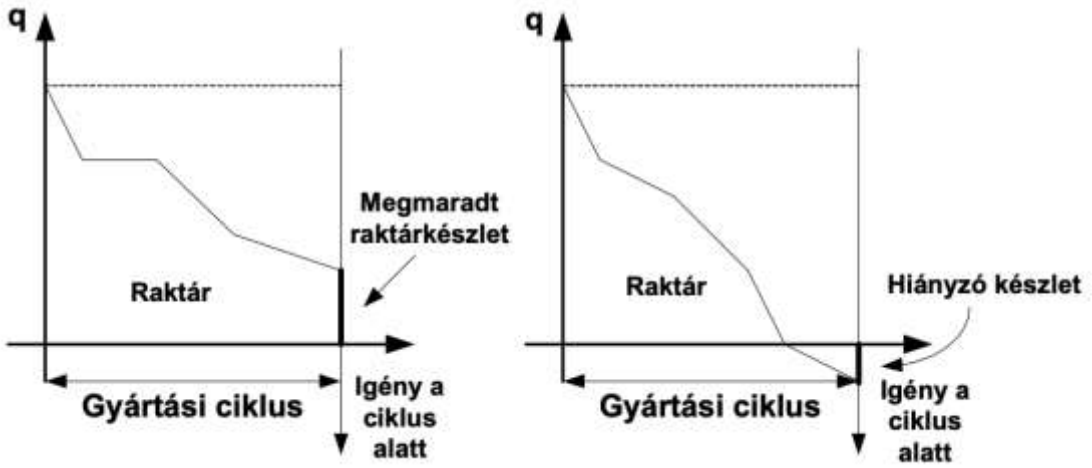
$$\frac{dK(q)}{dq} = \frac{d}{dq}\{c_f + c_v(q - I) + pE[D - q]^+ + hE[q - D]^+\} = 0. \quad (2)$$

A (2) alapján a q^* optimális értékre egy indirekt összefüggést kapunk, hosszú távon az optimális mennyiség az alábbiak alapján számítható:

$$F(q^*) = \frac{p - c_v}{p + h}, \quad (3)$$

ahol F az igény eloszlásfüggvénye.

A modellben használt célfüggvény négy fontos tagból áll. Az első tagja a már bemutatott gyártásindítási, fix költség, ami a gyakorlatban egy magas érték az egyes darabköltségekhez viszonyítva. A változó költséget a $c_v(q - I)$ összefüggés szimbolizálja, ahol a feltételezzük, hogy I darab termék már rendelkezésre áll, ezért a gyártási mennyiség döntési változó $m = q - I$ darab termék legyártását fogja elrendelni. A költségfüggvény harmadik tagja a büntető költség matematikai kifejezése, amely a kielégítetlen igényből származó költséget szimbolizálja. A költségfüggvényben szereplő $max(x, 0)$ függvény akkor lesz nullától eltérő, ha az igény nagyobb, mint a raktáron lévő korábban gyártott mennyiség. Lehetnek természetesen olyan esetek is, amikor a hiány nem megengedett. Ezt úgy lehet figyelembe venni és behangolni, hogy a modell p paramétere ilyenkor egy magasabb értéket kap. Ha az igény több, mint a már legyártott késztermék mennyisége, akkor természetesen nincs többlet raktározási költség, mivel a raktár kiürül a rendelés teljesítése után.



1. ábra Az igény és raktározási mennyiség kapcsolata

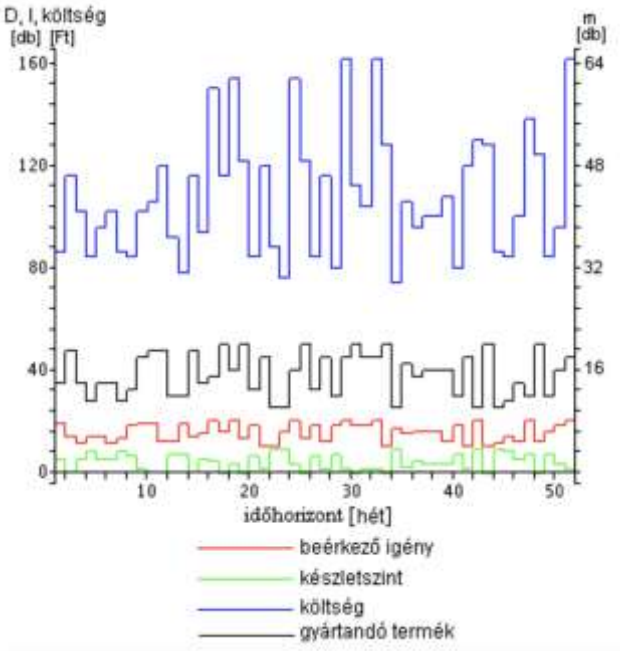
A raktározási költségtényező h értéke két további tényezőből tevődik össze: $h = H + h_t$, ahol h_t a termék tényleges tárolásának költségét jelenti. Ez önmagában azonban nem elegendő, hiszen a raktáron maradt termék értéke magában foglalja a beszállítónak korábban a termékbe fektetett pénzét is. Ennek megfelelően szükséges a termékenként értelmezett H forgótőke lekötési költség értelmezése és bevezetése is. A modellben a következő két ellenérdek ütközik: az egyik a termékek raktáron való tárolása, amely a gyártási mennyiség növelésével érhető el. Természetesen ez többlet-tárolási költséggel, forgótőke lekötéssel jár. A másik a hiány kockázatának viselése, amely azt jelenti, hogy kevesebb terméket raktározunk. Fontos észrevenni azt, hogy egy termék hiányának költsége nem egyenlő a termék árával, hanem attól magasabb és akár még a vásárló elvesztésével is járhat.

2.4 Szimulációk a politika ellenőrzésére

A továbbiakban a megoldott készletezési feladat ellenőrzését egy 52 hetes szimuláció segítségével végezzük el. A modellben a beszállító és vevő között kollaboratív kapcsolatot tételeztünk fel. Ez azt jelenti, hogy hiány keletkezése esetén a következő gyártási ciklusban a beszállító köteles az előző ciklus hiányát pótolni. Ennek szankcióit és a kockázat megosztását a felek szerződése a p értékében fejezi ki. A szimuláció elvégzéséhez MAPLE matematikai programcsomagot alkalmaztunk. Az illusztratív példa jellegű szimuláció alapadatai a következők:

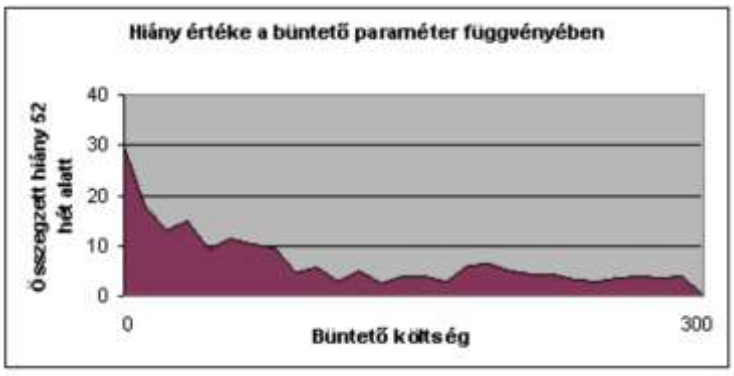
A termékekre való igény egyenletes eloszlást követ 10 db/hét és 20 db/hét intervallumban. A büntető költség minden hiányzó elem esetén $p = 50$ egység és a változó költség $c_v = 10$ egység. A kumulált raktározási költség termékenként $h = 5$ egység/periódus. A gyártási költség fix része

$c_f = 30$ egység/sorozat. A raktár kezdetben üres. A szimuláció eredményét a következő ábra szemlélteti.



2. ábra A beszállítói probléma szimulációs eredménye

Hiány és ezzel büntető költség keletkezik akkor, ha a beszállító raktárszintje negatív, ami egyben a ki nem elégített rendelés mennyiségét is jelenti. Ez akkor fordul elő, ha a beérkező igény nagyobb, mint a raktári mennyiség. Fontos megjegyezni, hogy nem történik gyártás akkor, amikor a készlet éppen az optimális és a kritikus raktárszint közé esik. Természetesen ekkor megnő a hiány bekövetkezésének valószínűsége. Valós feladatokban a termékek egy részére hiány nem, vagy csak minimális mértékben engedhető meg. Ez a feltétel a *p* büntető paraméter értékének magasra állításával építhető be a modellbe. A következő ábra mutatja, hogyan csökken a büntetés fizetésének gyakorisága nullára, miközben a büntető költség növekszik.



3. ábra A büntetőköltség növelésének hatása a valós hiányra

2.5 A feladat kiterjesztése több periódusra

Terjesszük ki a modellt két periódusra: az első periódus költségei már ismertek. Induljunk ki abból, hogy a második periódus az első folytatásaként értelmezhető és a felmerülő

költségtípusok azonosak. A két periódus együttes költségeinek meghatározására kiegészített költségfüggvény a következőként írható fel:

$$K_{12}(q_1, q_2) = c_f + c_v[(q_1 + q_2) - I_1] + hE[(q_1 + q_2) - D_1]^+ + hE[(q_1 + q_2) - D_1 - D_2]^+ + pE[D_1 - (q_1 + q_2)]^+ + pE[D_2 + [D_1 - (q_1 + q_2)\,]^-]^+. \quad (4)$$

Az első tag a gyártás un. *setup* költsége (ami egy fix költség), többperiódusos gyártásnál csak az első periódus előtt történik gyártás, és csak egy *setup* költséggel számolunk. Két periódus együttes gyártása esetén a második periódus mennyisége az elsővel együtt kerül legyártásra, így a készletezési mennyiségek összeadhatók $(q_1 + q_2)$. Ezt jelzi a második tag, ahol I_1 az első periódus kezdeti raktárkészletét jelenti. A harmadik tag az első periódus tárolási költséget reprezentálja, amely az első periódus igénye és a raktáron lévő mennyiségek különbségeként jelentkezik. Ha az első periódusról megmaradt mennyiség nagyobb, mint a második periódusban beérkező igény, akkor raktározási költség jelenhet meg a második periódusban is. A költség létezésének további feltétele az, ha az első hétről megmaradt mennyiség nagyobb, mint a második periódusbeli igény. Ezt a negyedik tag fejezi ki. Az ötödik tagban az első periódus ki nem elégített igényeinek költsége jelenik meg. Ha az első periódus igénye nagyobb a raktáron lévő mennyiségnél, akkor büntető költség keletkezik. Hiány azonban nem csak az első periódusban keletkezhet, hanem a másodikban is. Ennek költségét az utolsó, bonyolultabb tag fejezi ki. Az egyenletet megoldva az alábbi összefüggést kapjuk:

$$F_{12}(q_{12}^*) = \frac{p - c_v - hF_1(q_{12}^*)}{h + p}. \quad (5)$$

Az összefüggésben q_{12} mennyiség kifejezi, hogy mennyi késztermékneknek kell a raktáron lennie a vevői igény megjelenésekor két periódust együttesen vizsgálva. A megoldásban F_{12} jelenti az igények összegének együttes eloszlásfüggvényét, F_1 pedig az első periódusban beérkező igény eloszlásfüggvénye. Több periódus esetén most terjedelmi okokból nem részletezem az alapképletet, csak a megoldást, a levezetés megtalálható a [P3.7]-ben:

$$F_{123\ldots n}(q_{123\ldots n}^*) = \frac{p - c_v - (n-2)h - hF_{123\ldots n-1}(q_{123\ldots n}^*)}{p + h}. \quad (6)$$

A (6)-ban a számláló negatív értéket vesz fel abban az esetben, amikor a termékek periódusonként keletkezett tárolási költsége meghalad egy küszöbértéket, ekkor nincs

optimális megoldás, mivel a beszállító számára ekkor a „nem gyártás” költsége kisebb, mint a gyártásé.

2.6 Fajlagos költségmodell

A gyakorlati alkalmazások miatt vezessük be a fajlagos költség fogalmát. Jelölje $\widehat{K_i} = \frac{K_i}{q_i}$ az adott időhorizontra vonatkozó termékegységre eső beszállítói költséget, ahol K_i jelenti a korábban definiált i darab együtt gyártott periódus termelési költségét, q_i^* az i darab együtt gyártott periódus optimális készletezési mennyisége és $i = 1, \ldots, n$. Feltételezzük, hogy a fajlagos költség értéke a különböző darabszámú összevontan gyártott periódusok esetén különböző lesz. Költség alapú politika esetén a cél ezen értékekből a minimális megtalálása. Az a minimális fajlagos költséggel rendelkező K_{min}, amely kielégíti az alábbi összefüggést, az egyszeri gyártásindítással kielégíthető periódusok optimális darabszámát egyértelműen meghatározza a hozzá tartozó i változó értékével [P3.7]:

$$\widehat{K_{min}} = \min\{\widehat{K_1}, \widehat{K_2}, \widehat{K_3}, \ldots, \widehat{K_n}\}. \tag{7}$$

Példaként tekintsük egy rendelési periódusnak egy hetet. Az előrejelzésünk alapján a termékekre való igény periódusonként normális eloszlást követ 15 középértékkel és $\sigma = 3$ szórással. Legyen a büntető költség darabonként $p = 40$ egység. A kumulált raktározási költség $h = 2$ egység / periódus. A gyártási költség fix része $c_f = 120$ egység/sorozat. A változó költség értéke pedig $c_v = 5$ egység. A periódus elején tételezzük fel, hogy a raktárunk üres. A számítások során $n = 1, \ldots, 7$ db hét együttes gyártásának fajlagos költségeit és készletezett optimális q mennyiségeket a következő táblázatban foglaljuk össze.

	1.hét	2.hét	3.hét	4.hét	5.hét	6.hét	7.hét
Optimális mennyiség	16.9135	33.2634	**49.1625**	64.6729	79.8314	94.6601	109.1720
Fajlagos költség	12.6830	9.9330	**9.7844**	10.2182	10.8671	11.6133	12.4070

1. táblázat: Fajlagos költség minimuma, több hét együttes gyártása esetén

A táblázat oszlopai az együtt gyártott hetek számát jelentik. Az első sor jelzi az optimális mennyiséget az együtt gyártott hetek esetén, az utolsó sor pedig a fajlagos költséget szimbolizálja. Jól látható, hogy a fajlagos költség a harmadik oszlop esetében a legkisebb. Ez azt jelenti, hogy a definiált paraméterek függvényében *három hét együttes gyártása* esetén lesz a beszállító készletezési költsége minimális.

2.7 Termékkifutás modellezése

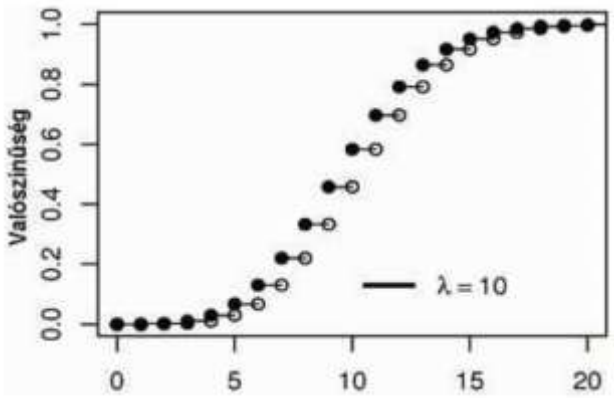
A piacon végzett gyakorlati vizsgálatok megerősítik, hogy a termékek életciklusában a kereslet ingadozásai mellett határozott *rövidülés* észlelhető. Gyakori eset, hogy olyan speciális termékek esetében, amelyek a hirtelen változó igényekhez igazodnak (például szezonális, karácsonyi csomagolóanyagok), a kereslet egy idő után teljesen megszűnik, és eladhatatlan készletek keletkeznek. Az ily módon keletkezett felesleges készletek nem csupán a komponensgyártó vállalat számára növelik a költségeket, hanem komoly veszteségeket okozhatnak a teljes ellátási láncban is. Napjainkban szinte minden kereskedő és beszállító vállalat szembesül ezzel a problémával, és általában tapasztalati alapú vezetői döntésekkel igyekszik megelőzni a károkat. A váratlan keresletcsökkenés vagy akár teljes megszűnés számos esetben nehezen előre jelezhető. Más esetekben azonban vannak előjelek. Egy ilyen összetett probléma kezeléséhez a készletgazdálkodás területén nélkülözhetetlen az emberi intelligencia szerepe. A döntéshozatal támogatására a készletgazdálkodási modellt olyan elemekkel kell kiegészíteni, amelyek az emberi intuíciót erősítik meg, ezzel segítve a probléma megoldását. Ezáltal a beszállító képes lehet különböző megoldási alternatívákat, készletkezelési stratégiákat és verziókat felmérni, az előrejelzések alapján értékelni a gazdaságosságot. A termékkifutás modellezésével tehát hatékonyabban lehet megválasztani a beszállítói készletkezelési politikát.

A készletgazdálkodás tudományos irodalmában eddig még csekély számú publikáció foglalkozik ezen problémával, és az eddig megjelent megoldási javaslatok túlnyomórészt a logisztikus eloszlás alkalmazására épülnek. [7] [8]

A Poisson eloszlás diszkrét, azaz a valószínűségi változó a természetes egész számok halmazából vehet fel értékeket. A Poisson eloszlás klasszikus formulája az alábbi:

$$P(\zeta = n) = \frac{\lambda^n}{n!} e^{-\lambda}, \quad n = 0,1,2, \dots, n \in N. \tag{8}$$

A következő ábrán a Poisson eloszlás diszkrét eloszlásfüggvénye látható, λ = 10 paraméter esetén.



4. ábra A Poisson eloszlás diszkrét eloszlásfüggvénye λ = 10 esetén

A raktáron marad termékekre új büntető tagot vezetünk be a *dR(i, λ)* (Poisson eloszlásfüggvény, ahol *i* a periódusokat jelöli) és a megmaradt termékmennyiség szorzattal. A költségfüggvényben eddig is szerelő raktározási költség tagokat pedig $1-dR(i, \lambda)$ összefüggéssel szorozzuk. A kifutás után büntető költség már nem jelentkezhet, ezért a függvényben megjelenő büntető tagokat szintén $1-dR(i, \lambda)$ összefüggéssel szorozzuk. A módszert a két periódusos gyártáson keresztül mutatom be. Bár a két periódus esetén nem nagyon lehet termékkifutásról beszélni, a példa célja a megértés egyszerűsítése. A két periódusos modell költségfüggvénye a bevezetett újabb tagokkal a következőképpen írható fel:

$$K_{12}(q_{12}) = c_f + c_v[q_{12} - I] + h(1-a)E[(q_{12}) - D_1]^+ + h(1-b)E[q_{12} - D_{12}]^+ + (1-a)\,pE[D_1 - q_{12}]^+ + (1-b)pE[D_2 + [D_1 - q_{12}\,]^-]^+ + aE[q_{12} - D_1]^+ + bE[q_{12} - D_{12}]^+, \quad (9)$$

ahol $a = dR(1, \lambda)$, $b=dR(2, \lambda)$.

A költségminimalizálás ezzel acélfüggvénnyel is elvégezhető, a levezetést és a végeredményt terjedelmi korlátok miatt itt nem mutatom be, megtalálható itt [P3.7]. A módszer helyességét szimulációkkal igazoltuk.

2.8 Összefoglalás

A rugalmas tömeggyártás kooperatív beszállítási feladatainak támogatására olyan új készletezési modellt fejlesztettünk és vizsgálunk, ami a klasszikus újságárus modell kiterjesztésének tekinthető. Az eredeti modellt több periódusra általánosítottam, így jobban illeszkedik a modern beszállítói láncok problémáinak kezeléséhez. Újszerűsége az volt, hogy a megoldást hagyományos analitikus úton kerestük és nem alkalmaztunk heurisztikus, genetikus, játékelméleti vagy egyéb soft-computing megoldásokat. A disszertációmban korábban megalkotott modell – marás szimuláció és optimalizáció – elsőre teljesen eltérő probléma, viszont már ott is felmerült az analitikus modellre való visszatérés, a diszkrét és soft-computing eljárások helyett, mivel abban hittem, hogy a gyorsabb számítógépek alkalmazása nemcsak a diszkrét modelleknek kedvez, hanem új lehetőséget nyithat komolyabb matematikai levezetések „végig számolására” is.

Az eredményeket mintapéldákon keresztül is publikáltuk és összehasonlítottuk a fent említett hagyományos módszerekkel. Az analitikus megközelítések általános előnye, hogy nagyon gyorsan pontos eredményt kapunk. Viszont sokszor a célfüggvény valós feladatok esetén nagyon bonyolult, mert sok paramétert kell kezelni, nincs lehetőség az egyszerűsítésre sem. A levezetésekhez természetesen alkalmaztunk szimbolikus matematikai szoftvert is (Maple V), amivel kezelhetőbbé vált a levezetések kidolgozása, de a probléma összetettsége miatt komoly kitartást igényelt a végeredményhez való jutás folyamata. A kutatás során a hagyományos eljárásokat is implementáltuk, hogy legyen összehasonlítási alapunk a végeredmény

ellenőrzéséhez. Mérésekkel azt a hipotézist is igazoltuk, hogy a 30-40 periódusos, több száz termékes feladatok optimális készletszintjeinek kiszámítási ideje a töredékére csökken a bemutatott eljárás alkalmazásával, hiszen az eredményt néhány számítási lépés adja.

2.9 A témához kapcsolódó publikációk

[P2.1] P. Mileff and K. Nehéz, "An Extended Newsvendor Model for Solving Capacity Constraint Problems in a Multi-item Multi-period Environment," *PRODUCTION SYSTEMS AND INFORMATION ENGINEERING*, vol. 5, pp. 95–108, 2009.

[P2.2] P. Mileff and K. Nehéz, "Modelling and Solving Inventory Control Problems in Customized Mass Production," in *Proceedings of the Manufacturing 2006 Conference*, 2007, pp. 141–149.

[P2.3] P. Mileff and K. Nehéz, "Az igény szerinti tömeggyártás készletgazdálkodási problémáinak megoldása módosított újságárus modell segítségével," in *Doktoranduszok Fóruma 2006*, 2007, pp. 135–142.

[P2.4] K. Nehéz and P. Mileff, "Solving Capacity Constraint Problems in a Multi-Item, Multi-Period Newsvendor Model," in *microCAD 2007, M szekció*, 2007, pp. 169–176.

[P2.5] K. Nehéz and P. Mileff, "Evaluating The Proper Service Level in a Cooperative Supply Chain Environment," in *MIM'07. IFAC workshop on manufacturing modelling, management and control*, 2007, pp. 123–126.

[P2.6] P. Mileff, K. Nehéz, and T. Tóth, "A New Inventory Control Method for Supply Chain Management," in *Machine Design and Production. Proceedings of the 12th International Conference on Machine Design and Production UMTIK*, 2006, pp. 393–408.

[P2.7] P. Mileff and K. Nehéz, "An Extended Newsvendor Model for Customized Mass Production," *ADVANCED MODELING AND OPTIMIZATION*, vol. 8, no. 2, pp. 169–186, 2006.

[P2.8] P. Mileff and K. Nehéz, "Applying analitical methods in Inventory Control Problems," in *microCAD 2006*, 2006, pp. 217–222.

[P2.9] P. Mileff and K. Nehéz, "Applying game theory in Inventory Control Problems," in *microCAD 2006*, 2006, pp. 223–229.

[P2.10] P. Mileff and K. Nehéz, "A new heuristic method for inventory control of customized mass production," in *Proceedings of the 8th International Conference on The Modern Information Technology in the Innovation Processes of the Industrial Enterprises*, 2006, pp. 353–358.

[P2.11] P. Mileff and K. Nehéz, “Beszállítói láncok elemzése analitikus, játékelméleti és korlátozás programozás módszerével,” in *XI. FMTÜ Nemzetközi Tudományos Konferencia*, 2006, pp. 267–270.

[P2.12] P. Mileff and K. Nehéz, “Módosított újságárus probléma alkalmazása az igény szerinti tömeggyártásban,” *GÉP*, vol. 57, no. 10, pp. 35–43, 2006.

[P2.13] K. Nehéz, P. Mileff, F. Erdélyi, and T. Tóth, *Hatékony elvek és módszerek alkalmazása a beszállítói készletgazdálkodási politikában*. Miskolc: Miskolci Egyetem Alkalmazott Informatika Tanszék, 2006.

[P2.14] P. Mileff and K. Nehéz, “Collaborative Inventory Control Policies in Supply Chains,” *PRODUCTION SYSTEMS AND INFORMATION ENGINEERING*, vol. 3, pp. 71–83, 2005.

2.10 Felhasznált szakirodalom

[1] S. COHEN and J. ROUSSEL. Strategic Supply Chain Management: The Five Disciplines for Top Performance, McGraw-Hill Companies, 2005

[2] P. H. ZIPKIN. Foundations of Inventory Management, McGraw-Hill, 2000.

[3] H. SCARF, A Survey of Analytic Techniques in Inventory Theory, In Scarf et al, editors, Multistage Inventory Models and Techniques, 1963.

[4] G. HADLEY and T. M. WHITIN, Analysis of Inventory Systems, Prentice Hall, Inc., Englewood Chiffs, New Jersey, 1963.

[5] H. AYHAN, J. DAI, R. D. FOLEY, and J. WU, Newsvendor Notes, ISyE 3232 Stochastic Manufacturing & Service Systems, pages 67-78, 2004.

[6] T. VOLLMANN, L. B. WILLIAM, C. D. WHYBRAK, and F. R. JACOBS. Manufacturing Planning and Control for supply chain management, Fifth Edition. McGraw-Hill, 2005.

[7] P. EGRI and J. VÁNCZA. Incentives for Cooperative Planning in Focal Supply Networks, Proc. of the 6th International Workshop on Emergent Synthesis, pages 1724, 2006.

[8] E. W. WEISSTEIN. Logistic Distribution, In: MathWorld - A Wolfram Web Resource. http://mathworld.wolfram.com/LogisticDistribution.html., 2006.

3 Modern feladat kiértékelők

3.1 Bevezetés

Az oktatás egy új online lehetősége a MOOCs (Massive Online Courses) informatikai rendszer megközelítés, amely írott anyagokkal, video leckékkel és tesztekkel segíti a tanulási folyamatot. Későbbiekben, a járvány miatti veszélyhelyzet és lezárások szinte rákényszerítették a felsőoktatást is az online lehetőségek széleskörű használatára. (természetesen ezt még a kutatás kezdetekor nem tudhattuk előre)

A MOOC kurzusok/tanfolyamok olyan modern online oktatási megoldások, amelyek a hagyományos egyetemi kurzusok anyagait nagy számú résztvevőnek teszik elérhetővé a világhálón keresztül. Egy jellemző MOOC tanfolyam videó előadásokból, gazdag olvasóanyagból és a hallgatók számára könnyen hozzáférhető tudásfelmérő tesztekből állnak, emellett interaktív fórumokat biztosítanak a tanuló közösség számára, így támogatva a tudásmegosztást, a csoportmunkát és a kommunikációt a tanulók, a tanárok és a tanítást támogató személyek számára, ezzel is fejlesztve a nem kognitív funkciókat [1, 2, 3]. A programozás tanulása azonban jóval többet jelent, mint oktató videók nézegetése, tájékoztató anyagok olvasása vagy hagyományos tesztek kitöltése. A programozás/program írás speciális tanulási folyamata algoritmikus gondolkodást, probléma megoldási képességeket követel meg, ráadásul hosszú távon lezajló folyamat [4]. Különbséget kell tenni a programozási ismeretek (például azt, hogy képes valaki megállapítani, hogyan működik egy „do while” ciklus) és a programozási stratégiák (képes valaki „do while” ciklust használni egy adott probléma megoldásánál) között. A diákok számára komoly nehézségeket okoz a programozási utasítások (feltételek, ciklusok, függvények stb.) kombinált használata.

A programozás tanulása ezért azt kívánja a hallgatóktól, hogy gyakorlati és problémamegoldó készségeket szerezzenek. Ennek megfelelően az online környezetben elérhető programozási tananyagoknak olyan kódolási feladatokat kell tartalmazniuk, amelyek lehetővé teszik az aktív gyakorlást, valamint az elkészült programok/megoldások kiértékelését és a gyors visszajelzést a megoldás minőségéről [5].

Ahhoz, hogy egy a fentieknek eleget tevő tananyagot, illetve kurzust készítsünk, nélkülözhetetlen egy megbízható tartalomkezelő rendszer (Learning Management System, LMS). Ilyen rendszerek közé tartoznak például a Moodle, a Listmos, a Canvas, a Talent Learning Management System és az Open edX, amelyet a MeMOOC kurzusokhoz használnak. Ezek a LMS-ek lehetővé teszik a kurzus anyagainak és tevékenységeinek hatékony kezelését, a tanulók számára könnyű hozzáférhetőséget biztosítanak a tartalmakhoz, valamint lehetőséget adnak az értékelésre és visszajelzésre. Az LMS rendszerek segítségével a kurzusok szervezése, adminisztrációja és nyomon követése egyszerűsödik, ami a tanulók és oktatók számára is kényelmesebb és hatékonyabb tanulási élményt eredményez.

3.2 Programok automatikus kiértékelése

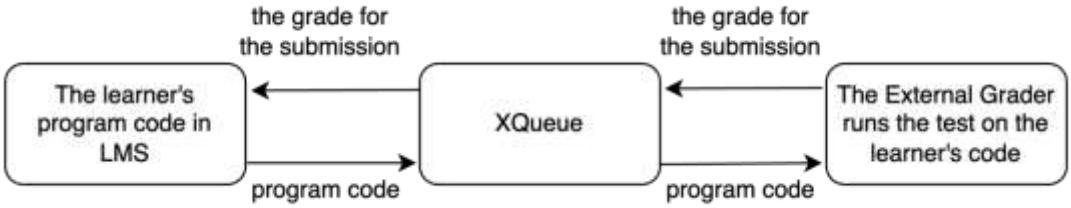
A MeMOOC projektben a kiértékelő rendszer az edX egy külső alrendszere. Amikor egy programozási kurzushoz kiértékelőt tervezünk, figyelembe kell vennünk az alábbi szempontokat [P3.3]:

- A kiértékelőnek le kell futtatnia a kódot. Többnyire nem elegendő, ha a kapott kódot, mint szöveget összehasonlítja az elvárt kódszöveggel.
- A kiértékelőnek a diákok kódját egymástól elkülönítetten kell futtatnia.
- A kártékony vagy hibásan megírt kód nem befolyásolhatja a MOOC rendszer működését. Például, ha *Alice* végtelen ciklust tartalmazó kódot tölt fel, a MOOC rendszernek továbbra is működőképesnek kell maradnia. Ugyanígy, ha Bob kódja a merevlemezt próbálná törölni, vagy le akar foglalni mondjuk 10Tb memóriát, akkor ez nem lehet engedélyezett egy MOOC rendszerben.
- A hatékonyság kulcsfontosságú tényező. A rendszernek képesnek kell lennie a kérések sorba állítására és a lehető leggyorsabb válaszadásra, hiszen több ezer felhasználó küldheti be a kódokat egyidejűleg.
- A kiértékelőnek támogatnia kell a virtuális végrehajtó környezetet. Ha a MOOC rendszert egy UNIX alapú rendszeren futtatjuk, de Windows programozást oktatunk, akkor a külső kiértékelőnek egy virtuális környezetben kell működnie.
- A biztonság kiemelkedő fontosságú. Mivel a diákok krediteket kapnak a diplomához vezető úton, nem szabad megengedni, hogy megváltoztathassák a beküldött kódjukra kapott értékelést.
- A kiértékelőtől kapott válasznak egyértelműen jelzést kell adnia arról, hogy a kód elfogadható-e. Ha hiba van, támogatnia kell a kód javítását.
- A kiértékelőnek mindig biztosítania kell a névtelenséget, nem szabad tudnia, hogy az értékelt kód *Alice* vagy *Bob* kódja.

3.3 Program kiértékelés megvalósítása a MeMOOC rendszerben

A kifejlesztett kiértékelő rendszer az edX-hez kapcsolt külső modul, amely szolgáltatásként fut, elvileg az edX platformtól függetlenül is használható. Az edX egy XQueue elnevezésű interfészen keresztül kommunikál a kiértékelővel. Elküldi a tanuló megoldását a kiértékelőnek, majd aszinkron módon vár a kiértékelő eredményére és visszajuttatja azt az LMS-be. A beküldések egy üzenetsorban tárolódnak mindaddig, amíg a kiértékelő el nem kéri azt feldolgozás céljából. A külső kiértékelő rendszeresen lekérdezi az XQueue-t. Amikor a kiértékelés megtörtént, a választ visszaküldi az XQueue számára egy RESTful interfészen keresztül. Az XQueue ezután válaszol az edX LMS-nek. Így működik a kiértékelő rendszer,

amely tehát aszinkron módon veszi ki (pull) az üzeneteket a sorból és teszi vissza a válaszokat (push) (5. ábra).

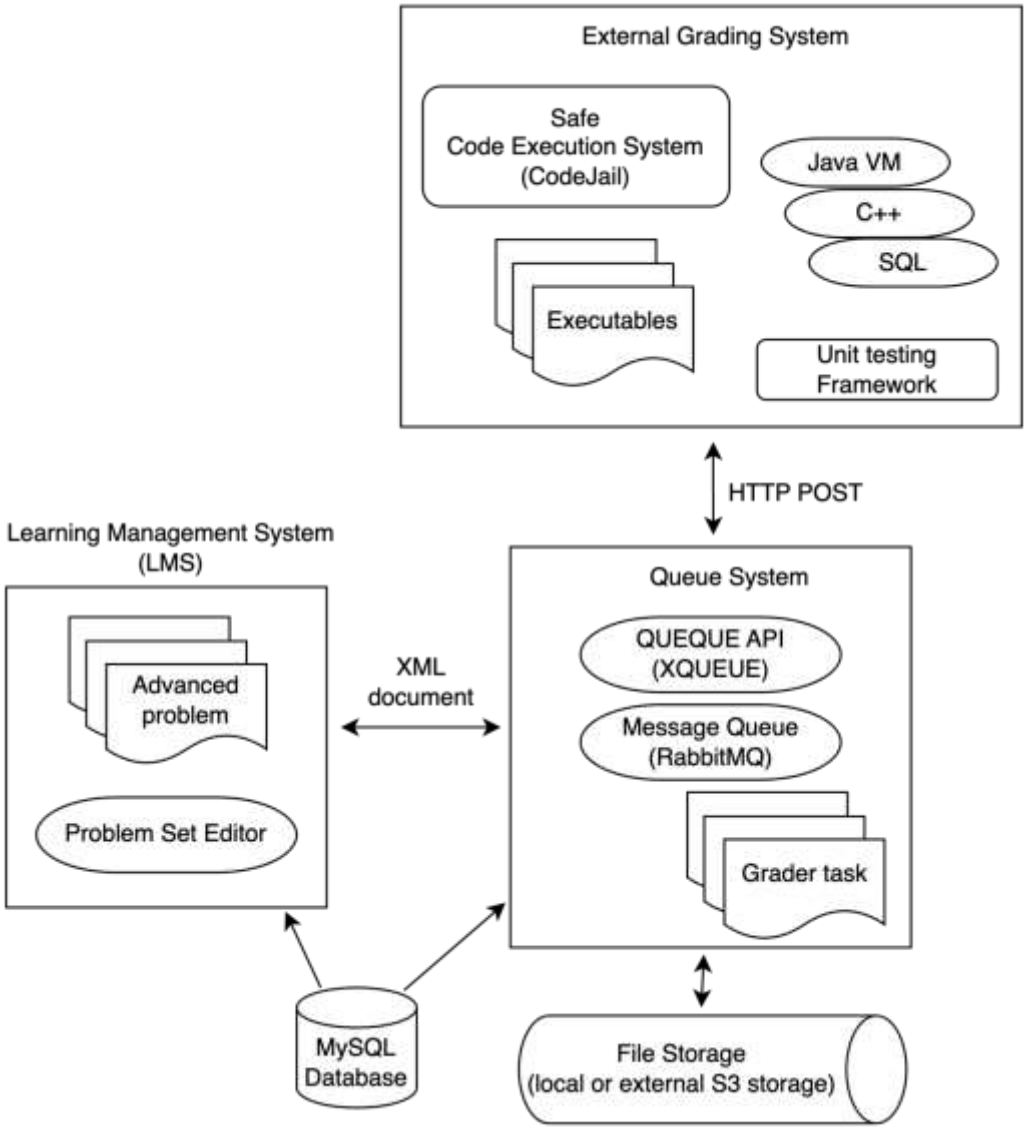


5. ábra a külső kiértékelő elvi működése

A feltöltött kódok nem egy mappaszerkezetben találhatók, mint egy egyszerű, közvetlen megoldás esetében, hanem adatbázisban. Az edX alapértelmezésben támogatja nagy fájlok feltöltését egy Amazon S3 bucket-ben és a rá mutató URL-jét tárolja ezeknek a lokális adatbázisban. Több ezer beküldött fájl esetén ez a megközelítés nagyon hatékony. A hátránya az, hogy az Amazon S3 kereskedelmi szolgáltatás, ami növelheti a fenntartási költségeket, és külső függést okoz a szolgáltatásban. Ezért a MeMOOC-ban egy lokális MySQL adatbázist használtunk a beküldött fájlok tárolására. A rendszerünk tartalmaz egy második üzenetsort is (RabbitMQ), ami a következőképpen egészíti ki a működést [P3.3], [P3.4], [P3.5]:

(1) Az XQueue sor értesítést kap az új feladatbeküldésről. (2) A RabbitMQ kiolvassa az adatot az adatbázisból, majd (3) serializálja azt. Ezután (4) a MeMOOC rendszer küld egy HTTP üzenetet a külső kiértékelőnek és (5) várakozik a válaszra. Eközben a küldő kiértékelő (6) inicializálja a virtuális környezetet, (7) lefordítja a beküldött programkódokat és (8) futtatja a megfelelő egységteszteket, ami többféle aspektusból ellenőrzi a beküldött kódot. A beküldött program kimenete egy fájlban lesz tárolva, melynek neve véletlenszerűen generált karakterekből áll.

A következő, 6. ábra mutatja a javasolt rendszer elvi működését.



6. ábra: A beküldött program kiértékelésének menete „External Grader” segítségével

A megvalósított kiértékelő rendszer nemcsak teljes, önállóan is működőképes programokat, hanem metódusokat, struktúrákat, sőt c++ nyelv esetében osztályok deklarációit is tudja tesztelni. Ehhez Java nyelv esetén a JUnit assertEquals metódusát használhatjuk, a teszt eredményét JSON *string*-ben adja vissza, amely tartalmazza, hogy a beküldött kód helyes volt-e, a pontot és egy tetszőleges üzenetet. C++ és nem teljes kódok esetén, a tesztet és a kiértékelendő kódot akár össze is fűzhetjük és indítás után a teszt kimenete lesz az eredmény.

A rendszerben a programozási feladatokat egy belső XML-ben lehet megadni. Ráadásul ebben a fájlban lehetőség van olyan belső Python kódok írására is, ami lehetővé teszi dinamikusan változó feladatok írását.

A következő, 7. ábrán látható egy feladat olyan formában, ahogyan a tanuló látja, alatta pedig az XML változat a Python kódokkal:

7. ábra: programozási mintafeladat a MeMOOC rendszerben

A fentieknek megfelelően a MeMOOC rendszer képes:

- ugyanazt a kódolási feladatot adni minden felhasználónak,
- ugyanazt a kódolási feladatot adni véletlenszerűen generált értékelési paraméterekkel,
- véletlenszerűen adni egy kódolási feladatot.

A MeMOOC rendszerben alkalmazott, új, saját fejlesztésű kiértékelő program a kiértékelő rendszerek mind a négy aspektusa szerint képes volt a beküldött kódok ellenőrzésére, ami nemzetközi viszonylatban is ritkaságnak számít.

A *szintaktika aspektus* azt jelenti, hogy a diák kódját le kell futtatni, illetve átadni egy értelmezőnek (interpreter), ennek megfelelően a kódnak, a használt nyelv szintaktikai szabályait kell követnie. A modern szoftverfejlesztési gyakorlat, széles körben elfogadott és nyelvenként eltérő komoly kód formázási és több szintű elnevezési megkötéseket ír elő, ezeket a szabályokat összefoglalóan *kódolási konvencióknak* nevezzük. A hallgatókat már az adott nyelv elsajátításának korai fázisában szoktatni kell ezen szigorú szabályok betartására. A kódolási konvenció annak ellenőrzésére szolgál, hogy a hallgatók betartották-e az általunk előírt kódolási útmutatást. A MeMOOC projekt esetében ennek egy haladó módszerét használtuk a kódolási stílus ellenőrzésére. A *Checkstyle* egy nagyon jól konfigurálható kódstílus- elemző eszköz, amely segíti a programozókat, hogy olyan Java kódot írnak, amely egy előre definiált (Sun vagy Google) standard kódoláshoz kapcsolódik. Modulok olyan halmazát határozza meg, amelyek képesek megvizsgálni a gyakori stílushibákat, például

- osztályok, metódusok és attribútumok helytelen név konvencióit;

- kötelező fejlécek jelenlétét;
- nem megfelelő importálást;
- hatókör-módosítókat;
- utasításblokkokat;
- hosszú sorhosszakat.

A következő, 8. ábrán látható kód a helytelen kapcsos zárójel használatot mutatja be. A Google Java Style Guide (2016) definiálja a megfelelő zárójel használatot, a kód megsért ezt. A CheckStyle- t futtatva az észreveszi a hibát és jelzi azt.

```
public class Person
 {  // Violates 4.1.2 point of [25]: No line break before the opening brace.
   private Date birthDay;
   public Date getBirthDay()
   { // the same formatting problem
     return birthDay;
   }
   public void setBirthDay(Date birthDay) {
     this.birthDay = birthDay;
   }
 }
```

```
Starting audit...
[WARN] Person.java:2:1: '{' at column 1 should be on the previous line. [LeftCurly]
[WARN] Person.java:5:3: '{' at column 3 should be on the previous line. [LeftCurly]
Audit done.Audit done.
```

8. ábra: helytelen kódolási konvenció használata

A *strukturális aspektust* akkor használjuk, amikor a tanulók már képesek komplex adatokat, struktúrákat létrehozni, a kiértékelőnek pedig ezeket kell tudnia ellenőrizni. Például a feladat szerint a diáknak egy *Person* osztályt kell deklarálnia, amelyben az egyik mező neve szöveg típusú, neve *name*, valamint implementálnia kell egy *getName* metódust, amely visszaadja a személy nevét. Az osztálynak van még egy integer típusú *age* nevű mezője is. A kiértékelőnek ellenőriznie kell az alábbiakat [P3.1]:

- a *Person* osztály deklarálva lett-e,
- deklarálva lett-e az *age* mező,
- létezik-e *getName* metódus.

Az átalunk készített kiértékelő a JUnit-ot használja az ellenőrzéskor. Megpróbálja betölteni a *Person* osztályt dinamikusan. Ha az nem létezik, akkor egy a *ClassNotFoundException*-t dob, amely el lesz küldve a beküldő számára. Majd ellenőrzésre került a getName metódus is. Ha az nem létezik, akkor egy *assertion failure* keletkezik, azaz a unit teszt sikertelen. Ha a metódusnak

paramétereket kell átadni, akkor a *person.getMethod* hívásakor azokat is át kell adni. A következő, 9. ábra mutatja azt is, hogyan ellenőrizzük az *age* mezőt. A kód megkapja a Person osztály mezőit, a kapcsolódó unit teszt hibát ad, ha az *age* nem található.

Az *működési logika* aspektusa esetén a rendszerünk a standard JUnit teszteket használta a program a működési logika ellenőrzésére.

```
@Test
 public void testRetirementAge() {
  Person p = new Person();
  p.setAge(60);
  assertEquals(p.getAgesUntilRetirement(), 5);

  p.setAge(70);
  assertEquals(p.getAgesUntilRetirement(), -1);
 }
```

9. ábra: példa unit tesztre

Ha a feladat azt például azt kéri, hogy hozzunk létre egy olyan függvényt, ami meghatározza, hogy egy személynek hány évet kell még dolgoznia nyugdíjba vonulásáig (pl. 65 éves koráig). A fenti logikát megvalósító programkód tesztelésére, az 5. ábrán látható unit tesztet lehet futtatni (szándékosan egyszerűsítve a kiírást egyetlen számra). A MeMOOC projektben került sor olyan programokat kiértékelő rendszer létrehozására, amely nemcsak a feltöltött programokat tudta tesztelni, hanem képes volt a kiértékelő rendszerek mind a négy aspektusa szerint ellenőrizni a kódot.

Habár a parancssori interfész (CLI) alkalmazások többnyire elegendőek a programozási készségek fejlesztésére, a grafikus felhasználói interfészek (GUI) az alkalmazásokban növelhetik a diákok motivációját és a felhasználói élményt [6]. Az általunk kifejlesztett rendszerben a GUI-t tartalmazó Java programok értékelésére a *Mockito* framework-öt használtunk. Tegyük fel, hogy egy programnak három *checkbox* elemet kell a felülethez hozzáadnia, valamint implementálnia három eseménykezelőt. A *Mockito* rendszerben a tesztet úgy kell implementálni, hogy megvizsgáljuk a három *checkbox* objektum példányosítva van-e, és az *addItemListener* metódus meg lett-e hívva háromszor. A külső GUI létrehozásához *Mockito* és *Powermock-module-* JUnit installálására volt szükség a szerver oldalon. A tesztelő programban a következő módon kell ellenőrizni az osztály létezését: az értékelő programunk megpróbál hivatkozni egy objektumra, amelyet a diák programjában létre kellett hozni (*checkbox*), és ha az nem létezik egy *try-catch* blokkban, az értékelő elkapja a hibát. Az elkapott kivétel alapján a külső értékelő üzenetet küld a diáknak arról, hogy mi hiányzik a programjából.

3.4 Összefoglalás

Ebben a fejezetben a MeMOOC projekt keretében kifejlesztett online programozási kurzusok és az ezekhez kapcsolódó automatikus kiértékelő rendszer működését mutattam be. A

kutatás célja, hogy hatékony és megbízható oktatási környezetet biztosítson a programozás tanulásához, valamint segítse a diákokat fejlesztéseikben és készségeik elsajátításában.

Az oktatási környezet bemutatása során részleteztem a MOOC-okat, melyek az online oktatás egy új formáját képviselik. A MOOC-ok a hagyományos kurzusanyagokat elektronikus formában kínálják a világhálón keresztül, lehetővé téve a tömeges létszámú hallgatóknak az elérhetőséget. Ezek a kurzusok videó előadásokkal, gazdag olvasóanyaggal és tesztekkel segítik a tanulási folyamatot, valamint interaktív fórumokat biztosítanak a hallgatók közötti kommunikációhoz és együttműködéshez. Azonban a programozás tanulásához további elemekre van szükség, mint az algoritmikus gondolkodás és problémamegoldó készségek, amelyek hosszú távú fejlődést igényelnek. Ezért fontos, hogy az online kurzusok olyan gyakorlati kódolási feladatokat tartalmazzanak, amelyek lehetővé teszik az aktív gyakorlást és a gyors visszajelzést.

A program kiértékelő rendszernek számos szempontot kell figyelembe vennie, például a kód futtatását, a diákok kódjainak elkülönítését, a hatékonyságot és a virtuális környezet támogatását. Emellett a biztonság kulcsfontosságú szerepet játszik, hiszen nem engedhető meg, hogy a diákok kódjai biztonsági kockázatot jelentsenek a rendszer működésére. A MeMOOC rendszer a beküldött kódokat egy lokális adatbázisban tárolja, és a kiértékelést aszinkron módon végzi, hogy hatékonyan kezelje a nagy mennyiségű beküldött kódot. A kiértékelő rendszer különböző aspektusokat vizsgál meg, mint például a szintaktika, a struktúra, a működési logika és a grafikus felhasználói interfész. A MeMOOC rendszer saját fejlesztésű kiértékelő programot hozott létre, amely képes a négy aspektus alapján ellenőrizni a beküldött programokat. Ennek eredményeként a diákok részletes visszajelzést kapnak a kódjuk helyességéről és a fejlődési pontokról.

Összességében a MeMOOC projekt által kidolgozott online programozási kurzusok és az automatikus kiértékelő rendszer hatékonyan támogatják a számítógép programozás tanulását, segítve a diákokat a készségeik fejlesztésében és a gyakorlásban. A kutatás eredményei elősegítik az online oktatás fejlődését és hatékonyságát a programozás területén.

3.5 A témához kapcsolódó publikációk

[P3.1] Szabó Martin, Nehéz Károly: MOOCs rendszerek alkalmazása a programozás oktatásában In: Bíró-Károly, Ágoston; Sebestyén-Pál, György; Szabó, Lóránd (szerk.) ENELKO 2018 XIX. Nemzetközi Energetika-Elektrotechnika Konferencia SzámOkt 2018 XXVIII. Nemzetközi Számítástechnika és Oktatás Konferencia, Kolozsvár, Románia: Erdélyi Magyar Műszaki Tudományos Társaság (EMT) (2018) pp. 297-300.

[P3.2] Szabó Martin, Nehéz Károly: Grading Java Code Submissions In MeMOOC, In: Kékesi, Tamás (szerk.) Multiscience XXXII. MicroCAD International Multidisciplinary Scientific Conference, Miskolc-Egyetemváros, Magyarország: Miskolci Egyetem (ME) (2018) Paper: C2_4, 5 p.

[P3.3] Király Sándor, Nehéz Károly, Hornyák Olivér: Some aspects of grading Java code submissions in MOOCs, RESEARCH IN LEARNING TECHNOLOGY 25 Paper: 1945, 16 p. (2017)

[P3.4] Dr. Kusper Gábor, Percze Gábor, Dr. Kovács László, Dr. Nehéz Károly, Dr. Radványi Tibor, Dr. Király Sándor, A MeMOOC online informatikai egyetem koncepciója In: NIIFI,. (szerk.) Networkshop 2016, Budapest, Magyarország: Nemzeti Információs Infrastruktúra Fejlesztési Intézet (NIIF Intézet) (2016) p. CD

[P3.5] Kusper Gábor, Havasi Gábor, Király Sándor, Kocsis-Baán Mária, Nehéz Károly, Hornyák Olivér, Mileff Péter, Introducing MeMOOC and Recent Results in e-Learning at University of Miskolc, In: Darco, Jansen; Lizzie, Konings (szerk.) MOOCs in Europe: Overview of papers representing a collective European response on MOOCs as presented during the HOME conference in Rome November 2015 : Papers 'WOW! Europe embraces MOOCS', Heerlen, Hollandia: European Association of Distance Teaching Universities (EADTU) (2016) 221 p. pp. 75-78. , 4 p.

3.6 Felhasznált szakirodalom

[1] Arefin, A. S. (2015): Pedagogy of Computer Programming: An Interactive and Collaborative Learning Approach. Macquarie University Postgraduate Certificate of Higher Education EDCN 871 Final Project

[2] McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002): The effects of pair-programming on performance in an introductory programming course. SIGCSE '02 Proceedings of the 33rd SIGCSE technical symposium on Computer science education: Pages 38-42.

[3] Teague, D., & Roe, P. (2008): Collaborative learning: towards a solution for novice programmers. Paper presented at the Proceedings of the tenth conference on Australasian computing education- Volume 78.

[4] Muratet, M., Torguet, P., Viallet, F. & Jessel, J.P. (2010): Experimental feedback on Prog&Play: a serious game for programming practice. In: L. Kjelldahl and G. Baronosk (Eds.), EUROGRAPHICS 1–8.

[5] Vihavainen, A., Luukkainen, M. & Kurhila, J. (2012): Multi-faceted support for MOOC in programming. Proceedings of the 13th Annual Conference on Information Technology Education, ACM, New York, pp. 171–176.

[6] Staubitz, T., Pfeiffer, T., Renz, J., Willems, C., & Meinel, C. (2015). Collaborative learning. In ICERI2015 8th annual International Conference of Education, Research and Innovation, Seville, Spain (pp. 18-20).

4 Egyes rajzi elemek automatikus felismerése

4.1 Bevezetés

2017 és 2021 közötti időszakban a GINOP-2.3.4.-15-2016-00004 azonosítójú - *Korszerű anyagok és intelligens technológiák FIEK létrehozása a Miskolci Egyetemen* címmel - nagyprojektben volt konzorciumvezető. A FIEK (felsőoktatási és ipari együttműködési központ) projektben az informatika részprojekt vezetésével bíztak meg, amiben a projekt teljes időtartama alatt és a fenntartási időszakban is részt vettem. A konzorcium tagjai több informatikai feladatot is megfogalmaztak, aminek egyik elemét sikerült olyan szinten megvalósítani, hogy szabadalmat nyújtottunk be az eljárás jogvédelme érdekében 2019-ben.

4.2 Optikai karakterfelismerés

A műszaki innovációs és dokumentációs folyamatok végrehajtásának első állomása a műszaki tervek és/vagy felmérési dokumentációk készítése. A műszaki termékek gyors és sokféle megvalósítása manapság, különösen szükségessé teszi a digitális módszerrel elkészített és számítógépes rendszerekben tárolt rajzok automatikus rajzolvasását. A tervlapokra a tervezők a tervezési szakágaknak megfelelően nézeti, metszeti vagy axonometrikus ábrázolásokat készítenek a megtervezendő objektumról vagy berendezésről. A tervezők a rajzi ábrázolásra méretvonalakat, méreteket, betű vagy szám karaktereket és sok esetben rajzi alakzatokat helyeznek el. Az alakzatok, lehetnek zászlók négyszögek, ellipszisek és más nyitott vagy zárt vonal együttessel határolt tetszőleges tervlap részletek. Ezek az alakzatok általában fontos rajzi elemeket hordoznak a rajzi alakzatokra ráírva vagy beleírva. A gépészetben gyakran találkozunk ilyen rajzi alakzatokban megadott helyzet és alaktűrésekkel. Az alakzatba beírt vagy beszerkesztett információhoz a rajzok felhasználói csak vizuálisan a teljes rajz részletes áttanulmányozásával tudnak hozzájutni, majd onnan kijegyzetelve eltárolni azokat.

Az optikai karakterfelismerés egy elektronikus eljárás a kézzel írt, vagy nyomtatott dokumentumot ábrázoló képeken szereplő szöveg gépi, szerkeszthető szöveggé való átalakítására. Széles körben használt nyomtatott dokumentumok például számlák, receptek, levelek, információinak számítógépbe való automatizált bevitelére alkalmasak. Az optikai karakterfelismerés előnye, hogy az eljárás elvégzése után az ilyen módon kinyert információ digitálisan szerkeszthető, tartalma kereshető, egyszerű a tárolása, használható további gépi folyamatok bemeneteként például gépi fordítás, adatbányászat, de a szöveg hanggá alakítása is lehetséges.

Az eddig használt optikai karakterfelismerő módszerek olyan számítógépes eljárások, amelyek ismerik a felhasználók által használt legtöbb betűtípust és karakterkészletet. Sajnos gyakran előfordul, hogy a felismerés bemenetét képező dokumentumon egy olyan kevésbé ismert rajzi elem, például szokatlan betűtípus, és/vagy sajátos szimbólum jelenik meg, amelyre kizárólag

speciális szakterületeken van igény. Ilyen rajzi elemek, a géprajzokon található átmérő vagy párhuzamosság szimbólumok, többsoros tűréselemek, vagy matematikai egyenletek. Ezeket az eddig ismert eljárások nem tudják a digitális rajzokról kinyerni és számszaki értéküket megjeleníteni. matematikai a géprajzokon használnak fel, amelyeket az adatkinyerő módszerek nem tudnak értelmezni. Erre az esetre ismert megoldás egy betanításra alkalmas optikai karakterfelismerő eljárás használata. Így az ismeretlen szimbólumok, és betűtípusok manuális, vagy automatizált úton megtaníthatók a karakterfelismerő eljárásnak. Azonban ez a folyamat körülményes, hosszú időbe telik, és sokszor a végeredmény sem kielégítő. A technika mai állása szerint különböző optikai írás felismerő rendszer típusokat különböztetünk meg. Az optikai karakterfelismerő eljárás karakterenként ismeri fel az írott szöveget. Az optikai szó felismerő eljárás során szavanként történik az írott szöveg felismerése. A szavak határát a szóközök, illetve egyéb elválasztó jelek jelölik. Intelligens karakterfelismerő eljárások az optikai karakterfelismerő rendszerhez hasonlóan karakterenként ismeri fel az írott szöveget. Ennek az eljárásnak a feltalálói bevezették a gépi tanulás módszerét. Az intelligens szófelismerő eljárás szintén szavanként ismeri fel az írott szöveget, azonban a feltalálók ez esetben is kiegészítették a módszert a gépi tanulás lehetőségével. Ez az eljárás akkor is képes felismer egy szót, ha a határai nincsenek egyértelműen jelölve. Néhány ismert eljárás ma már matematikai formulákat is képes felismerni.

4.3 Az irodalomban ismert módszerek áttekintése

Az US0337443 számú szabadalom [1] mobiltelefonokon is használható, helyhez nem kötött optikai karakterfelismerő eljárást mutat be elsősorban számlakezelési célzattal. Ezen eljárás szerint a felismerendő karaktereket általában vásárlási számla tartalmazza, amit a mobiltelefon digitális képalkotó rendszere digitalizál és küld el egy központi szerverre. A központi szerverben történik meg a számlakép felismerése összehasonlító klisék alkalmazásával. A kiolvasott, és a beküldőnek is visszaküldött adatokat a szerver tovább feldolgozza, majd csoportosított bolti egyenleget készít. Az eljárás rajzi jelek kiolvasására nem alkalmas, különlegesen azért nem, mert az eljárás az analizálható felületen zárt vagy nyitott alakzatban szereplő karaktereket kiolvasni nem képes és így az eljárás segítségével nem lehetséges alakzatszelektálás sem.

Az US 8370117 számú szabadalom [2] digitális rajzokon található rajzi jelek felismerésére és egy előre megadott szabály szerinti vizsgálatára alkalmas eljárás. Az eljárás szerint minden méret és a hozzá tartozó tűrésmező beolvasásra kerül, ezzel automatikus tűrésmező szélesség ellenőrzés történik meg. A szabadalom szerint a rajzon egy előre meghatározott rajzi alakzatban található, előre meghatározott rendszer szerinti rajzi tartalom, például tűrésmező és a hozzá tartozó méret kiolvasására képes az eljárás. A rajzi jelek tekintetében az összes rajzon található azonos rajzi elemet és a hozzá tartozó méretet kigyűjti az eljárás és meg is jeleníti a felhasználó számára. Az eljárás különlegessége az, hogy a rajzi elemek tartalmát nemcsak kiolvassa a számítógépes eljárás, hanem értelmezi is azokat és a vonatkozó szabályoknak és méretezési előírásoknak megfelelően döntéseket is hoz arról, hogy az adott rajzjelben található és kiolvasott

és számmá konvertált karakter, például tűréskarakter megfelel-e a gyárthatóság feltételeinek. Amennyiben gyárthatósági nehézség várható az adatfelismerés és számítás eredményeként, akkor azt a felhasználóval az eljáráshoz kapcsolódó információs egységen keresztül közli az eljárás. A vizsgálati eredményeket vizuális alakzatba konvertálja az eljárás és az eredményeket hozzáfűzi a rajzi megjelenítéshez. Az eljárás azonban csak a rajz alapéleivel vagy párhuzamos vagy merőleges helyzetű és az előre meghatározott általában téglalap formátumban megadott rajzi jelek vagy karakterek felismerésére és adatainak kiolvasására és feldolgozására képes. A szabadalom tetszőleges irányban álló, tetszőleges alakú rajzi alakzatok felismerésére és kigyűjtésére nem képes, így azok teljes tartalmának kiolvasására sem képes. Amennyiben az eljárás nem talál az értékeléshez szükséges kritériumokat a keresés és kiértékelés leáll.

A JP2001318952 szabadalom [3] optikai karakterfelismerő rendszerrel ellátott számítógépen futtatható eljárást ismertet. Ez az eljárás minden szöveges karaktert felismer és külön egységen keresztül közli a felhasználóval. Az eljárást digitálisan feldolgozható rajzok szövegmezőinek kiolvasására hozták létre. Az eljárás csak merőleges vonalakkal határolt és keretbe beírt szám vagy betűkaraktereket ismer fel, függetlenül azok méretétől vagy alakjuktól. Az eljárásnak van központi adatfeldolgozó processzora, adatbeviteli egysége, adattárolója és megjelenítő egysége. A központi processzorhoz ellenőrző egység kapcsolódik. A szabadalom szerinti megoldás nem képes tetszőleges helyzetű rajzi elemekből történő optikai karakterfelismerésre és megjelenítésre. Az eljárás minden szöveges vagy számmal ellátott karaktert felismer, ami merőlegesen határolt keretben van, de nem képes a rajzi elemek szelekciójára és azok teljes tartalmának teljes kiolvasására.

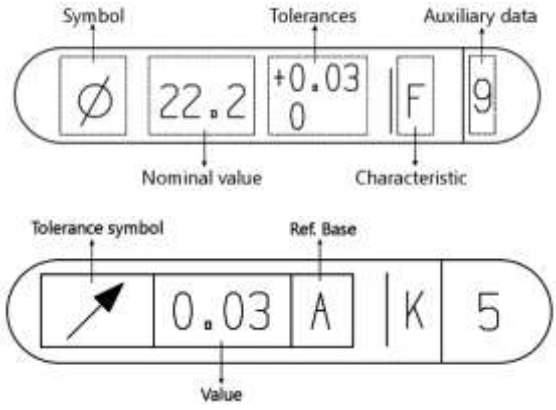
Az US 0372161 számú szabadalom [4] gépjármű rendszámok felismerésére és az ebből nyerhető adatok közlésére szolgáló eljárást mutat be. Az eljárás képi adatbevitel segítségével egy számítógép központi processzorának alkalmazásával végzi el a rendszámtáblákon felismerhető karakterek felismerését. A karakterek kiolvasásához olyan megfelelő nagyítást használ és a kiolvasandó karaktereket egy tűrésezett keretbe illeszti, amelyben ezt követően optikai karakterfelismerést végez. Az eljárás hátránya az, hogy rajzi felületeken nem alkalmazható, mivel minden rajzi keretben levő karaktereket felismer, szimbólumok azonosítására viszont nem alkalmas. Minden egyes ki nem olvasható karaktert hibaüzenettel visszaküld a felhasználónak korrekcióra.

A kezdeti karakterfelismerő eljárásokat az egyes karakterek képeivel egyenként kellett betanítani. A jelenlegi rendszerek már képesek magas fokú pontossággal felismerni a legtöbb ismert betűtípust, és számos különböző digitális képfájl formátumot támogatnak. Néhány eljárás képes az eredeti bemenettel közel megegyező formázott kimenet létrehozására, beleértve a képeket, táblázatokat, és egyéb nem-szöveges komponenseket. A hagyományos szófelismerő eljárások jellemzője az, hogy a szöveg azonos orientációban, általában sorfolytonosan van rendezve. A jelenleg is használt optikai karakterfelismerő alkalmazások szinte kivétel nélkül ugyanazt az eljárást követik a szimbólumok felismeréséhez. Az ismert optikai karakterfelismerő eljárások általános eljárás elemei a digitalizálás, elő feldolgozás, szegmentálás, karakter

kinyerés, karakter megjelenítés. Az ismert eljárások ugyan rendelkeznek betanító eljárással, de annak használata nem ad teljes adatkinyerést különlegesen azokban az esetekben, amikor a szimbólumok és a karakterek nyitott vagy zárt alakzatban, eltérő méretekkel és orientációban szerepelnek a dokumentumokon. Ezen különleges esetekben minden eddig ismert eljárás hibát vét és a rendelkezésre álló információt elhagyja, mivel nem tudja értelmezni azokat. Az eddigiekben használt karakterfelismerő eljárásokkal az információvesztés lehetséges.

4.4 Az alkalmazott saját fejlesztésű eljárás

Az előzőekben feltárt és ismert eljárások hibáinak kiküszöbölésével a találmány célja olyan eljárás létrehozása volt, amely alkalmas kétdimenziós képi formában megjeleníthető rajzokon vagy térképeken, vagy kétdimenziós képet hordozó digitális adathordozón található szelektált rajzi alakzatok információtartalmának teljes kinyerésére megjelenítésére és azok adattárolására. Az eljárás a szelektíven válogatott és tárolt adatokat szövegesen vagy akár hanganyag formában is közli a felhasználóval, és így teljes információkinyerést hajt végre, még akkor is, hogyha az információt hordozó alakzat zárt vagy nyitott és tetszőleges szöghelyzetű tetszőleges rajzi elem, tartalma pedig tetszőleges, de két dimenzióban ábrázolható jelölés.

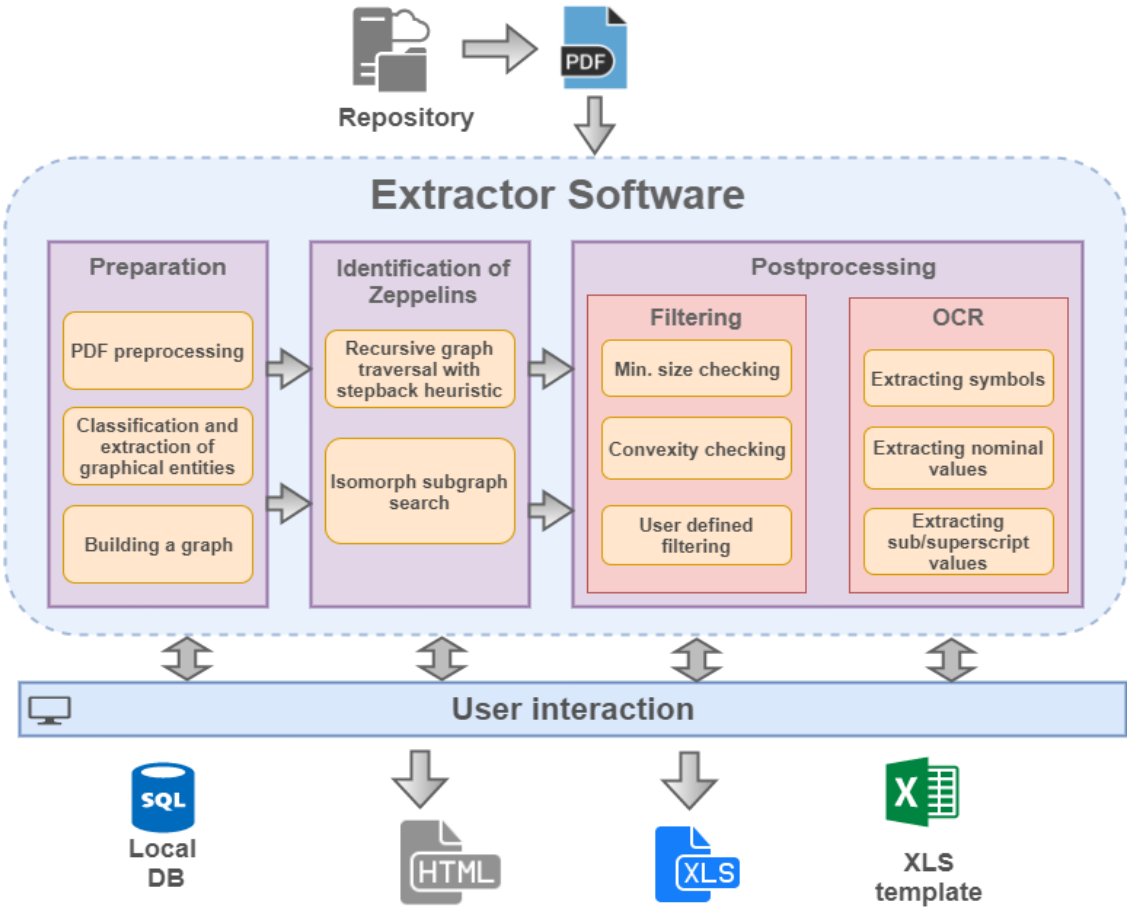


10. ábra A Zeppelin-ek általános felépítése

A jelen eljárásunk kiindulópontja egy elektronikusan vagy tetszőleges adathordozón előálló műszaki, vagy más kétdimenziós rajz. Ez a rajz lehet szabadkézi vagy kinyomtatott műszaki rajz utólagosan digitalizált dokumentum vagy számítógép segítette tervezői rendszerekből kiemelt elektronikus dokumentum. A felismerési folyamat előtt alakzatfelismerés, majd alakzat szelekció történik. Kiválasztásra kerül a sokféle speciális alakzatú jelölés közül, az amelyik megjelenik a feldolgozandó dokumentumban vagy dokumentumokban. Az eljárás következő lépéseként a dokumentum alakzatait, vektorizált formára átalakítjuk. A vektorizált adatformátum egyeneseket és görbéket tartalmaz, amelyekből a következő lépésben virtuálisan matematikai gráfot építünk. Az átalakítás során figyelembe vesszük, hogy a vektor elemek végpontjai a digitalizálás/exportálás miatt pontatlanul illeszkedhetnek és az illesztési hibákat automatikusan kijavítjuk. A bementi dokumentum vektorgrafikus alakzatait, a módszer matematikai gráf alakba transzformája. Az átalakítás során figyelembe veszi, hogy a vektor

alakzatok végpontjai pontatlanul illeszkedhetnek. Az elő feldolgozás célja a lehető legkevesebb csúcsponttal rendelkező, összefüggő gráf kialakítása. A következő lépésben a géprajzi szabványokban szereplő befoglaló jelöléseket részgráf kereső algoritmussal azonosítja majd a megtalált részgráfok közül automatikus vagy kézi szűrési eljárással szelektáljuk ki azokat, amelyekből adatkinyerést végzünk. Az eljárás a dokumentum tetszőleges részében, tetszőleges orientációjú és méretű, de azonos alakú alakzat megjelöléseket is kezel. Ezt követően a szelektált alakzatból karakterkinyeréssel rajzi elemeket nyerünk ki.

A karakter kinyerés során új alakzat felismerő algoritmust használunk, amely tetszőleges méretű, elrendezésű és orientációjú, vektorizált formában eltárolt alakzatokat ismer fel. A felismerendő alakzatokat alakzatkezeléssel a már ismert alakzatokkal történő összevetés útján azonosítjuk. Az eljárás része az alakzattár elkészítésének és automatikus bővítésének folyamata. Az eljárás végül az alakzat tartalmát felismeri és a folyamat által kinyert információt feldolgozza, rendszerezi, és adatbázisba menti. Ennek során új optikai karakterfelismerő eljárás létrehozásával, olyan eljárási részfolyamatokból összekapcsolt olyan új eljárást hoztunk létre, amely elindulása után megtörténik a dokumentum bevitele, majd a dokumentumban az eljárás eredményeképpen automatikusan kijelölésre kerül az összes olvasandó alakzat. Ezt követően az alakzatokból karakterkinyerést végzünk, majd megjelenítjük és szükség szerint továbbítjuk a rajzi alakzatok tartalmát és megállítjuk az eljárást. Az eljárás működéséhez szükség van tudásbázis bővítés végrehajtására, ami lehetővé teszi az alakzatok információ tartalmának teljes kinyerését és megjelenítését.



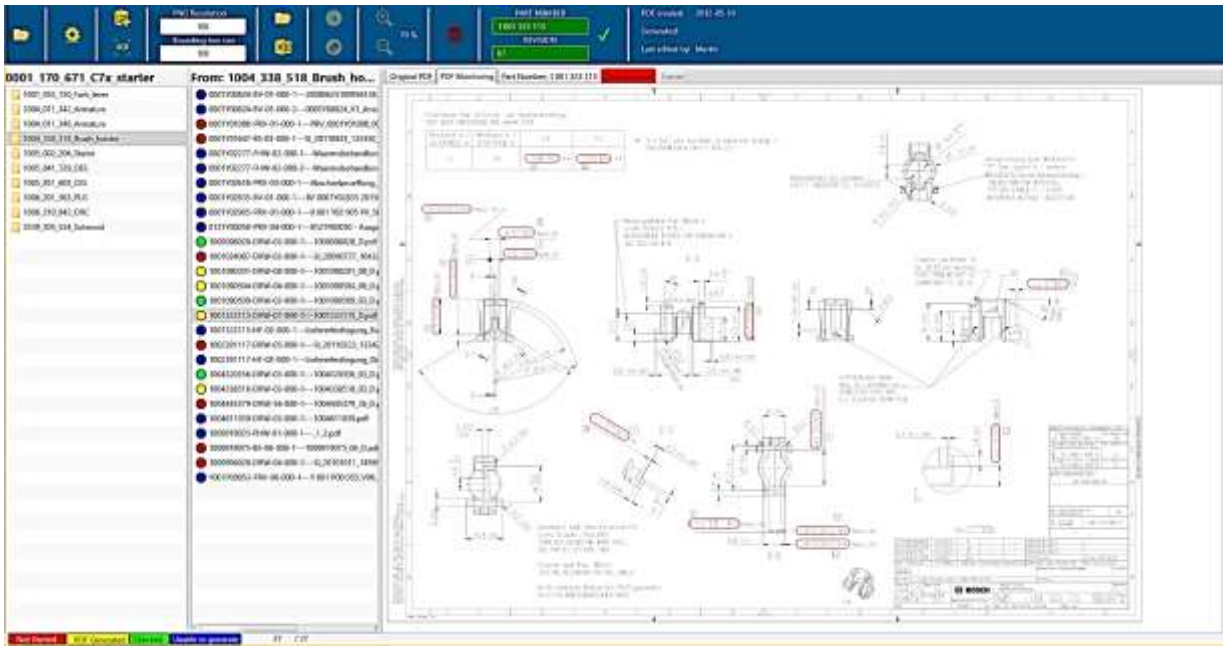
11. ábra Az alkalmazás architektúrája – az alakzatkinyerés elvi lépései

Az 11. ábra a kinyerés folyamatát mutatja be a FIEK keretén belül implementált szoftverben. Ebben a megvalósításban a Siemens TeamCenter fájltárolóból dolgozunk, itt tárolják a komplex termékek alkatrészeinek műhelyrajzait PDF formátumban. Az előkészítési folyamat előtt letöltjük a műhelyrajzokat és adatbázisban tároljuk. A PDF dokumentumokat előfeldolgozzuk: ezek tartalmazhatnak szakaszokat, görbéket, font-okat. A szakaszok végpontjai sokszor keresztezik egymást, nem illeszkednek. A görbék és szakaszok sokaságából gráfot alakítunk ki, figyelembe véve a metszéseket és az illeszkedéseket. Először az 10. ábra szerinti alakzatok körvonalát szeretnénk kinyerni. Ezek a speciális jelölések bármilyen szögben elforgatva is megtalálhatóak a rajzokon. Egy úgynevezett rekurzív visszalépéses gráfkereső algoritmust használunk a Zeppelinek körvonalának megkereséséhez.

A rajzokon az elemek száma több millió is lehet: termékváltozatonként a 40-50 PDF kinyerési idejét sikerült percekre csökkenteni, így több napos fárasztó munkától megkímélve a mérnököket. A zeppelinek körvonalának megtalálása után nem egyszerű feladat az 10. ábra szerinti alakzatok adatainak kinyerése sem. Ezzel a problémával a postprocessing fázisban birkóztunk meg. A feliratok lehetnek *true type* fontok vagy *block* fontok, az előbbi esetén OCR API-t használtunk, az utóbbi esetén saját automata betanító algoritmust.

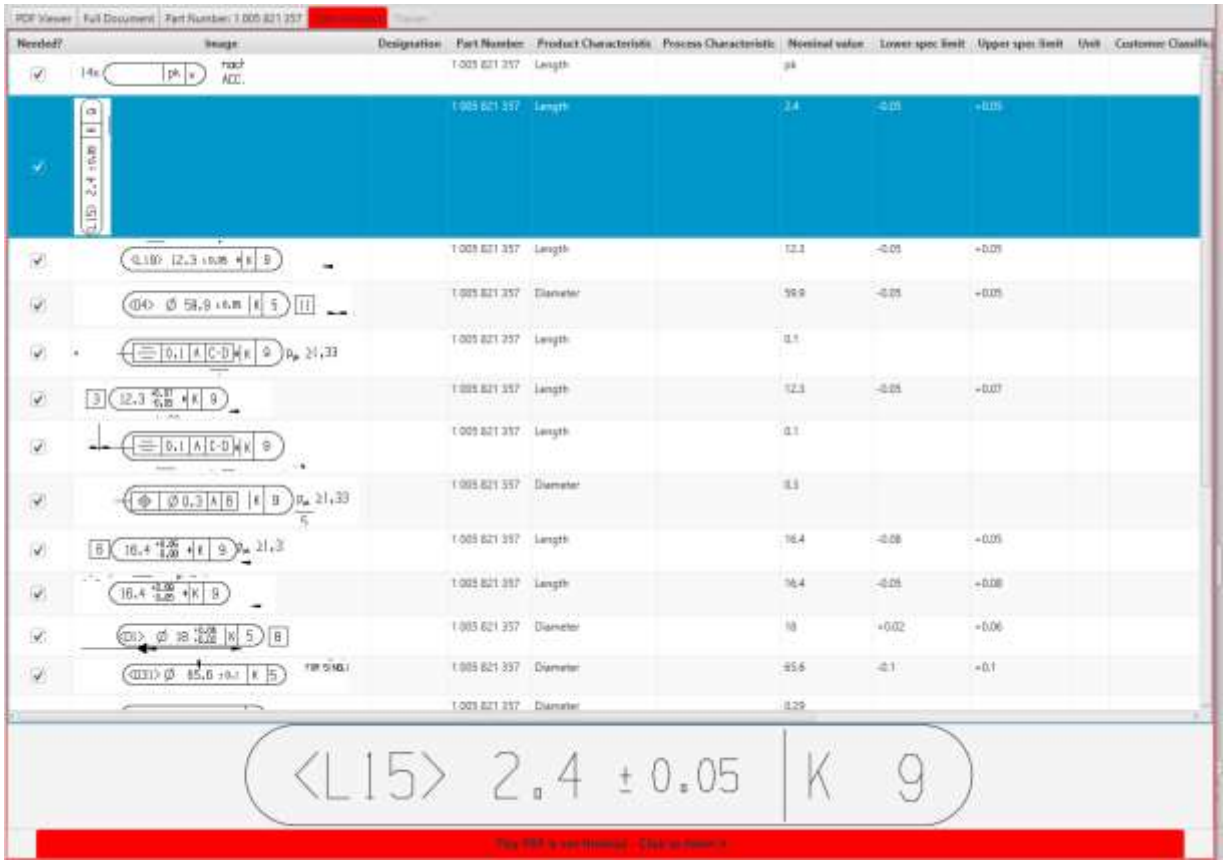
A méréseink alapján a Zeppelinek felismerési aránya 98% körüli, a feliratok 80%-os pontossággal felismerhetőek. A feliratok pontosságát kézi betanítással tovább lehet növelni. 11. ábra Az alkalmazás architektúrája – az alakzatkinyerés elvi lépései

4.5 Többfelhasználós interaktív alkalmazás



12. ábra A rajzfeldolgozó alkalmazás interaktív felülete

A 12. ábra a feldolgozó alkalmazás főképernyőjét mutatja, a bal oldali listában az indítómotor fő komponensei, jobbra az egyes műhelyrajzok belső kódolása látható. Színkódokkal jelöltük az egyes dokumentumok feldolgozottsági szintjét. Külön kiemelendő, hogy az alkalmazott informatikai háttérrendszer támogatja, hogy egy időben többen is párhuzamosan dolgozhatnak, módosításaik nem írják felül egymás munkáját.



13. ábra A szimbólumkinyerés eredménye

A fenti ábra az adatkinyerés képernyőjét mutatja, ahol soronként vizsgálhatóak és fontosság szerint rendezhetőek a sorok. A kinyer adatok kézzel módosíthatóak, a hibás számjegyek, szimbólumok külön betaníthatók.

4.6 Összefoglalás

A fejezetben bemutatott találmány tárgya *eljárás szelektált rajzi alakzatok teljes tartalmának kinyerésére és megjelenítésére*, amely elsősorban a műszaki rajzokon fellelhető rajzi alakzatokat szelektálja, majd azok teljes tartalmát felismeri és automatikusan olyan adatbázisba rendezi, amely közvetlenül a felhasználó személyek számára a rajz olvasása nélkül értelmezhető és/vagy amelyet számítógépes adatfeldolgozó szoftverek segítségével további műveletek elvégzésére lehet felhasználni. Az eljárás alkalmas elektronikus dokumentumokban szereplő tetszőleges helyzetű, géprajzi szabványokban vagy szabványon kívüli, de állandó formájú kétdimenziós rögzített rajzi alakzat és/vagy alakzatba foglalt rajzi elemek automatikus és teljes felismerésére, kinyerésére és megjelenítésére. A rajzi elemek lehetnek szimbólumok, karakterek, ismertetőjelek, kiegészítő jelzetek vagy ezektől eltérő, de információ tartalommal bíró jelölések. Az eljárás magában foglalja a jelölések felismerésének, az adatok kinyerésének és megjelenítésének folyamatát is. Ezt az eljárást elsősorban azok az ipari résztvevők használhatják a termelésük hatékonyságának emelésére, akik a tervezés és/vagy a gyártás során nagymennyiségű rajzi dokumentációt kezelnek, de felhasználhatják a térképészetben dolgozó

szakemberek is egyedi szimbólumok felismerésére és szükség esetén azok tartalmának automatikus kinyerésére.

4.7 A témához kapcsolódó publikációk

[P4.1] M. Szabó, K. Nehéz, P. Mileff, and O. Hornyák, "Eljárás szelektált rajzi alakzatok teljes tartalmának kinyerésére és megjelenítésére," Benyújtás száma: P1800403, Magyar szabadalom, 2018.

[P4.2] M. Szabó and K. Nehéz, "Optikai karakterfelismerő technikák és alkalmazási lehetőségek áttekintése," in *ENELKO 2018 XIX. Nemzetközi Energetika-Elektrotechnika Konferencia SzámOkt 2018 XXVIII. Nemzetközi Számítástechnika és Oktatás Konferencia*, 2018, pp. 301–305.

4.8 Felhasznált irodalom

[1] Inventors: Kolton, Achiav; Bentov, Albert (Holon, IL), LOCATION BASED OPTICAL CHARACTER RECOGNITION (OCR), United States Patent Application 20170337443, 2017

[2] Inventors: Dewey, McKinley, Sims, Jr., RoyalOak; Baljit, Bains, Canton, SYSTEMS, METHODS, AND TOOLS FOR PROOFING A COMPUTER-AIDED DESIGN OBJECT, United States Patent Application US 8,370,117 B2, 2013

[3] Inventors: Mitsuhisa Moriyama, 光久, 森山, METHOD AND DEVICE FOR EXTRACTING CHARACTER INFORMATION FROM CAD PICTURE, Application JP2000176095A events, 2000

[4] Inventors: Ricardo Andre S. G. de Almeida, Antonio R. Pinto, Romeu Rodrigues Figueira, INTELLIGENT AUTOMATIC LICENSE PLATE RECOGNITIONFOR ELECTRONIC TOLLING ENVIRONMENT, United States Patent Application US20170372161, 2017

[5] Kono Hideki, SYMBOL RECOGNIZING SYSTEM, Nippon Steen Coorporation, JP3240303(B2) , Japán szabadalom, 1994

[6] Suzuki Kaoru, et. all. DRAWING INPUT DEVICE, Tosgibe Corporation, JP19900140041, Japán szabadalom, 1992

5 Mesterséges intelligencia módszerek a szoftverek minőségbiztosításában

5.1 Bevezetés

Az egyik friss kutatási téma visszavezet a szoftvertechnológia világába. 2019 végén megcéloztam a szűkebb szakterületem egy új és érdekes témájának vizsgálatát, modern mesterséges intelligencia módszerek felhasználásával. Ez a témakör, a *repository mining* (kódtárak adatbányászata) területéhez kapcsolódik leginkább, de később a fókuszt áthelyeztem a szoftverhibák (bug-ok) keletkezésének okainak tanulmányozására és előrejelzésére.

A kódtárak (repo-k) használata ma már elengedhetetlen a modern szoftverfejlesztési munka napi gyakorlatában. A fejlesztők minden forráskód módosítást ilyen rendszereken keresztül teszik elérhetővé a kollegáiknak, ezért ezek a tárolók fontos történeti információt tárolnak egy szoftver fejlesztése során bekövetkező változásokról és tartalmazzák a fejlesztési lépések részletes lenyomatait, a forráskód lépésről-lépésre történő változásain keresztül. [T5.6] [T5.7] Ma már minden sikeres szoftvercég akár saját vagy szolgáltatásként igénybe vett kódtárat használ a napi munkája támogatására. Minden módosítás időbélyeget kap, ami utólag nem változtatható meg. A modern kódtárakhoz hibajelentések és feladatdokumentációk is tartoznak, sok esetben visszakövethetők, hogy milyen módosítások okoztak *töréseket* a tesztekben, mely módosítások befolyásolták negatívan a szoftver minőséget. Bár kódtárak korábban is léteztek, de ezek tudományos vizsgálata 2016 körül kezdett kialakulni. A ma legelterjedtebb kódtár a *git* kifejlesztése Linus Torwald nevéhez fűződik, aki a Linux kernel atyjaként ismert, viszont utólag is teljesen világos, hogy a 25 évvel ezelőtt szokatlan *decentralizált* fejlesztési modellt az általa kifejlesztett *git* kódtár nélkül lehetetlen lett volna világsikerre vinni. Az utóbbi években világossá vált, hogy a kódtárakban rengeteg rejtett információ lehet, amit érdemes lenne elemezni és a levont következtetések visszavezetni a szoftverfejlesztési folyamatba. Az új megközelítés szerint érdemes a kódtárak változásait mesterséges intelligencia módszerekkel folyamatosan nyomon követni és megtanítani, hogy az egyes változtatások, milyen valószínűséggel okoznak hibákat? Ennek a kérdésnek a megválaszolására vállalkoztam ebben a munkában. [T5.11]

5.2 A szoftverhibák előrejelzése

A szoftverhibák előrejelzése (SDP – software defect prediction) manapság az egyik népszerű újabb kutatási terület a szoftvertechnológiában, és az alkalmazásfejlesztés, kód karbantartás során is létfontosságú tevékenység [1]. Az SDP célja a szoftverminőség javítása és a fejlesztési költségek csökkentése azáltal, hogy a hibákat, a fejlesztés korai szakaszában azonosítják és kijavítják [2]. A szoftverhibák olyan hibák – közismert nevén *bug*-ok - a szoftverkódban, amelyek váratlan vagy kiszámíthatatlan viselkedést okozhatnak a szoftverben. Ezek a hibák szoftverösszeomlásokhoz, biztonsági sebezhetőségekhez, adatvesztéshez és más negatív következményekhez vezethetnek [3]. A hibák korai azonosítása és kijavítása a fejlesztési folyamatban időt és pénzt takarít meg a költséges utólagos tesztelések elkerülésével és

természetesen a szoftverhibák kockázatának csökkentésével. A hibajelentések adatbázisba szervezése (bug report database) alapvető szoftverfejlesztési dokumentációs módszer, amely a szoftverhibákat írja le és rendszerezi [4]. A hibákat alapvetően két osztályba sorolják: az belülről jövő un. intrinsics hibák olyan hibák, amelyeket a forráskód egy vagy több specifikus változása okozott, és a kívűlről jövő u.n. extrinsic) hibák olyan hibák, amelyeket az külső környezeti, konfigurációs, függőségi stb. változások okoztak, de nem rögzítették őket a verziókezelő rendszerben. Az SDP megvalósítása során több technika is alkalmazásra kerülhet, ideértve a statisztikai modelleket, gépi tanulási algoritmusokat és adatbányászati technikákat. Ezek a technikák a szoftverhibák történeti adatait használják, mint például a hibajelentéseket és a kódbeli változtatásokat, a jövőbeli hibák valószínűségének előrejelzéséhez [5]. Az kiinduló adat típusától és az előrejelzés kontextusától függően az SDP különböző típusokba sorolható, amelyek a következők:

- A projekten belüli hibajóslás (WPDP - within-project defect prediction) megközelítés magában foglalja a történeti adatok használatát egyetlen projektben belüli hibák előrejelzéséhez. A WPDP megközelítés a projektből származó adatokat használja a modell létrehozásához, például a forráskód metrikákat, a hibajelentéseket és a kód review-kat.
- A projektek közötti hibajóslás (CPDP-S - Cross-project defect prediction for a similar dataset) megközelítés hasonló adathalmazokhoz: ez a megközelítés egy új projektben létrejöhető hibák előrejelzéséhez, korábbi hasonló projektek történeti adatait alkalmazza. A CPDP megközelítés az egy vagy több hasonló korábbi projekt adatait használja az előrejelző modell kialakításához, majd ezt alkalmazza az új projektre. Az új projekt kezdetekor nem rendelkezünk még elegendő adattal, később átléphetünk a korábbi WPDP módszerre.
- A projektek közötti hibajóslás (CPDP-H - Cross-project defect prediction for heterogeneus datasets) megközelítés heterogén adathalmazhoz: ez a megközelítés az új projektben fellépő hibák előrejelzéséhez használja számos olyan projekt történeti adatait, amelyek fejlesztési környezetükben vagy jellemzőikben különböznek. A CPDP megközelítés sokféle, heterogén projekt adatait használja a modell betanításához, majd alkalmazza azt az új projekt hibáinak előrejelzésére.

Mindegyik SDP megközelítésnek vannak előnyei és természetes korlátai. A WPDP általában pontosabb, mivel az előrejelzés projekt specifikus kontextuson alapul, de jelentős mennyiségű történeti adatot igényel ugyanabból a projektből, így ezt csak régóta fejlesztett szoftverprojekteken lehet alkalmazni. A történeti adatokból a tanuló minták elkészítése is komoly kihívás lehet, főleg egy rosszul menedzselt szoftver esetén. A CPDP-S hasonló adathalmazai akkor lehetnek hasznosak, ha nincs elegendő adat a WPDP-hez. Ugyanakkor feltételezi, hogy az új projekt hasonló fejlesztési környezettel rendelkezik, hasonló technológiai megoldásokat alkalmaz. A heterogén adathalmazhoz tartozó CPDP-H kihívást jelenthet, mivel a betanításra használt projektek fejlesztési környezete és az új projekt jelentősen különbözhet.

Azonban hasznos lehet, ha nincs elegendő adat a WPDP-hez vagy a hasonló adathalmazhoz tartozó CPDP-hez [6] [7].

5.3 *Code Smell*-ek és jelentőségük

A *Code Smell*-ek olyan minták és jellegzetességek a forráskódban, amelyek nem feltétlenül (még!) hibákat, de potenciális problémákat vagy tervezési hiányosságokat jeleznek. Ezek a *"szagok"* arra utalnak, hogy a kód lehetne egyszerűbben, olvashatóbban és karbantarthatóbban is megvalósítva. A *Code Smell*-ek segítenek azonosítani olyan helyzeteket, ahol a tervezés során figyelmetlenség vagy hanyagság eredményezheti a kód nehezen érthetővé vagy bővíthetővé válását. A *Code Smell*-ek különböző formákban és kategóriákban jelentkezhetnek. Például az ismétlődő kód, a hosszú metódusok, és függvény paraméterek, a nehezen érthető változónevek mind olyan finom jelzések lehetnek, hogy a forráskód olvashatósága és karbantarthatósága fokozatosan romlik. Az ilyen problémák figyelmen kívül hagyása az alkalmazás egyre nehezebben karbantarthatóvá válik, amely végül megnöveli és bizonytalanná teszi a fejlesztési időt és költségeket.

5.3.1 Programhibák felismerése és kezelése

Mint láthattuk, a *Code Smell*-ek felismerése és folyamatos csökkentése kulcsfontosságú a kód minőségének fenntartásában. A jó tervezési gyakorlatok és a *tiszta kód* elveinek követése segíthet elkerülni ezeket a problémákat. A tervezési hibákat és problémás mintákat korai szakaszban kell azonosítani és kijavítani. A *refaktorálás*, vagyis a meglévő kód folyamatos átszervezése és tisztítása segít a *Code Smell*-ek eltávolításában.

Egy súlyos programhiba, futás közben programbezárást, lefagyást okoz, de vannak olyanok is amik hibás adatot hoznak létre. Az ilyen helytelen működést szakszerűen megírt tesztekkel ki lehet mutatni. Első példaként bemutatok egy olyan érdekes mintát, ami nem okoz fagyást, de a működése helytelen.

```
1 #include <iostream>
2 #include <cstring>
3
4 int main() {
5     const char* str = "Hello, world!";
6
7     char* mutableStr = const_cast<char*>(str);
8     mutableStr[0] = 'x';
9
10    std::cout << "Modified string: " << str << std::endl;
11
12    return 0;  // nem változik meg a 5. sorban definiált szöveg!
13 }
```

14. ábra Példa konstans adatok hibás kezelésére

Megjegyzés: A gyakorlatban ilyen közvetlen módon nem fogjuk ezt a típusú hibát megtalálni egyetlen szoftverben sem, hanem a konstans pointer az egyik osztály *getter* tagfüggvényének a visszatérési értéke lesz, amit már másvalaki elmentett egy összetett struktúrába, mondjuk egy *std::map*-be.

Ebben a példában az *str* változó egy konstans karakter mutató, azonban a 7. sorban mégis módosíthatóvá tesszük a *const_cast* alkalmazásával (azaz felülírjuk az eredeti viselkedést), majd megváltoztatjuk az első karaktert a "x" betűre. Bár ez az operáció látszólag sikeresen működik, de lefuttatva az első betű mégsem változik meg. [8]

A második példa - avatatlan szemnek - nehezen érthető, hogy miért fagy le. Az ehhez hasonló problémákat a statikus kódelemző szoftverek sem fedezik fel!

```
1 #include <iostream>
2 #include <vector>
3
4 class DataProcessor {
5 public:
6     DataProcessor() {
7         data = new int[10];
8         for (int i = 0; i < 10; ++i) {
9             data[i] = i;
10        }
11    }
12
13    ~DataProcessor() {
14        delete[] data;
15    }
16
17    int* getData() {
18        return data;
19    }
20
21 private:
22    int* data;
23 };
24
25 int main() {
26    DataProcessor processor;
27    int* ptr = processor.getData();
28
29    std::vector<DataProcessor> processorVector;
30    processorVector.push_back(processor);
31
32    std::cout << ptr[5] << std::endl; // kiírja az 5.öt!
33
34    return 0; // ez után lefagy! ??
35 }
```

15. ábra Erőforrások hibás kezelésére fagyást okoz

Ebben a példában a sérülékenység abból ered, hogy a *DataProcessor* objektumot egy vektorba helyezzük (a 30. sorban) és a vektor a másoló konstruktorát használja az objektum másolásához. A *DataProcessor* osztály azonban dinamikusan lefoglalt memóriát tartalmaz, amelyet a *destruktor* felszabadít, és a másoló konstruktor által létrehozott másolat azonban megpróbálja ugyanazt a dinamikusan lefoglalt területet felszabadítani, ami nem definiált viselkedést okoz (azaz nagy valószínűséggel lefagyást eredményez). A bemutatott sérülékenység a "*rule of three*"

(vagy "*rule of five*" C++11 után) elvét sérti, amely szerint, ha az osztály tartalmaz dinamikusan lefoglalt erőforrásokat, akkor saját másoló *konstruktort, értékadó operátort,* valamint *destruktort* is kötelező definiálni. [9]

Egy klasszikus *code smell* nem feltétlenül okoz ilyen drasztikus hatást, hanem a hatása hosszú távon növeli a fejlesztési időt, egyre lassabbá és keservesebbé válik a fejlesztők munkája, a termék élettartamát lerövidíti és megnöveli a költségeket is.

5.3.2 A code smell-ek típusai

Az irodalom négy fő típust sorol fel a *God class, Data class, Feature envy* és a *Long method* elnevezéssel.

- A „God class” egy olyan osztályt jelent, mely túlzottan komplex és a függvényeit szinte megkötés nélkül bárhonnan meg lehet hívni. Egy ilyen osztályban gyakran több funkció található, mint amennyi optimális lenne, ami megnehezíti a kód olvashatóságát és karbantarthatóságát. Egy nagyobb szoftver termékek szinte természetszerűleg tartalmazhatnak „God class”-t. Ennek az az oka, hogy a fejlesztők az évek során nem általánosítanak, hanem visszanyúlnak jellemzően a főablakot vezérlő osztályhoz és folyamatosan bővítik a funkcióit. A szoftver karbantarthatóságát jelentősen megnehezíti egy „God class” jelenléte a kódbázsban.
- A "Data class" azokra az osztályokra utal, melyek lényegében csak adatmezőket és azokhoz tartozó getter/setter metódusokat tartalmaznak. Ezek az osztályok általában nem valósítanak meg semmilyen valódi funkcionalitást, és csupán adattárolásra szolgálnak. Az ilyen osztályok könnyen kialakulhatnak, amikor a tervezés során a strukturális döntések nem megfelelően vannak meghozva. Az ilyen osztályok előfordulása csökkenthető a megfelelő absztrakció alkalmazásával.
- A "Feature envy" azt a helyzetet jelenti, amikor egy osztály túlzottan sokat "irigyel" egy másik osztálytól, azaz szorosan összekapcsolódik vele, és túlzottan hozzáférek az annak belső adataihoz és metódusaihoz. Ez a jelenség gyakran a szoftverkomponensek közötti helytelen interfész tervezésére és az információ rejtés hiányára vezethető vissza.
- Az "Long method" kifejezés hosszú metódusokra utal, melyek túl sok műveletet hajtanak végre egyetlen függvényhívás keretében. Ez megnehezítheti a kód olvashatóságát, karbantarthatóságát és újrafelhasználhatóságát is. A hosszú metódusok szétbontása kisebb, jól definiált részfunkciókká elősegíti a kód strukturáltabbá tételét és a fejlesztési folyamat hatékonyságának növelését. Ebbe a csoportba a nagyméretű osztályok is beletartoznak, amelyek nem ritkán több ezer sorosak.

5.3.3 Szoftver metrikák

A metrikák is fontos szerepet játszanak egy *bug* előrejelző modell létrehozásában. A szoftver metrikák nélkülözhetetlen eszközök a szoftverminőség mérésében és javításában [10]. A szoftver metrikákat fel lehet használni a szoftvertervezés szerkezeti tulajdonságaira vonatkozó

információk gyűjtéséhez, amelyeket további statisztikai elemzéseknek lehet alávetni, értelmezni és összekapcsolni a minőségével.

A szoftvermetrikák kvantitatív adatokat szolgáltatnak, amelyek elemzése során azonosíthatók a potenciális problémás területek. A kódbázis különböző aspektusainak mérésével, mint például a komplexitás (ciklomatikus vagy kognitív), a méret vagy a kódolási szabványok betartása [T5.1]. A szoftver metrikák segítenek azonosítani a hibákat vagy a kódproblémákat jelző mintákat és indikátorokat. A történeti adatok elemzése és a metrikák ismert problémákhoz történő korrelálása révén a fejlesztők felfedezhetik a visszatérő mintákat vagy a metrikák kombinációit, amelyek potenciális problémákat jeleznek. Ez lehetővé teszi számukra, hogy proaktívan kezeljék ezeket a területeket a hibák megelőzése vagy a kódminőség javítása érdekében. Emellett a szoftver metrikák támogatják a hibamegelőzésre és a kódminőség javítására irányuló döntéshozatalt. A metrikák felhasználásával a fejlesztők döntéseket hozhatnak a kód refaktorálásával, az architekturális változtatásokkal vagy a potenciális hibás területek hatékony kezeléséhez szükséges erőforrások elosztásával kapcsolatban [T5.1].

A szoftver metrikákat statikus kódmértékeknek és folyamatmetrikáknak lehet osztályozni. A statikus kódmértékek közvetlenül kinyerhetők a forráskódból, mint a kódsorok száma (LOC) és a ciklomatikus komplexitás (CCN). Az objektumorientált metrikák a statikus kódmértékek egy alkategóriáját alkotják, mint a leszármazási fa mélysége (DIT), az objektumok közötti kapcsolat (CBO), a gyermekek száma (NOC) és az osztály válaszai (RFC) [11]. Az objektumorientált metrikákat gyakran használják a forráskód tesztelhetőségének, karbantarthatóságának vagy újrafelhasználhatóságának értékelésére [11].

A statikus kódelemző szoftverek a kód futtatása nélkül végeznek szemantikai elemzést a forráskódon. Ezt a műveletet a ma ismert szinte valamennyi általánosan használt programozási nyelven.

5.4 Mintaadatok a modellépítéshez

Saját mintaadat készítésére nem vállalkoztunk a kutatás során, hanem felkutattuk az irodalomban bemutatott és széles körben használt nyilvánosan elérhető adatokat. A saját adathalmazok előállításának kérdése további érdekes kutatási feladat, amit jelenleg már elkezdtem vizsgálni.

5.4.1 Nyilvános adathalmazok

A szoftverhiba-előrejelzési kísérletek elvégzéséhez három különböző nyilvános *dataset*-et használtunk. Az első csoportot a NASA adathalmazaiból, négyet választottunk ki. Ezeket az adathalmazokat a NASA korábbi valós szoftverprojektekből gyűjtötte össze. (5.1. táblázat)

Projekt név	modulok száma	hiba %	Nyelv	Leírás
JM1	10885	19%	C	Valós idejű előrejelző földi rendszer: Szimulációkat használ előrejelzések készítéséhez.

PC1	1107	6.8%	C	Föld körüli pályán keringő műholdak repülési szoftvere.
KC1	2107	15.4%	C++	Tároláskezelés a földi adatok fogadásához és feldolgozásához.
KC2	523	20%	C++	Szoftver a tudományos adatok feldolgozásához.

5.1 táblázat: NASA adatbázis

A második csoportot egy nyilvános, egységesített hibaadatkészletből nyertük, a korábbi szerzők öt nyilvános adatkészletet vettek figyelembe, és letöltötték a megfelelő forráskódot az adatkészletekben szereplő minden egyes rendszerhez, és forráskódelemzést végeztek a forráskód-metrikák meghatározásához. Egységes hibaadathalmazt állítottak elő osztály- és fájlszinten, amely alkalmas új hibajelző modellek építésére. Továbbá összehasonlították a különböző hibaadathalmazok metrikadefinícióit és értékeit. Az 5.2. táblázat a nyilvános, egységesített hibaadatkészletre vonatkozó információit mutatja be.

Dataset	**Software**	**Kódsorok száma**
PROMISE	Ant, Camel, Ckjm, Forrest, Ivy, JEdit, Log4J, Lucene, PBeans, Poi, Synapse, Velocity, Xalan, Xerces	2,805,253
Eclipse Bug Dataset	Eclipse	3,087,826
Bug Prediction Dataset	Eclipse JDT Core, Eclipse PDE UI, Equinox Framework, Lucene, Mylyn	1,171,220
Bug catchers Bug Dataset	Apache Commons, ArgoUML, Eclipse JDT Core	1,833,876
GitHub Bug Dataset	Android Universal Image Loader, Antlr 4, Broadleaf Commerce, Ceylon IDE Eclipse Plugin, Elasticsearch, Hazelcast, JUnit, MapDB, mcMMO, MCT, Neo4J, Netty, OrientDB, Oryx, Titan	1,707,446

5.2 táblázat: Nyilvános egységesített adathalmazok

A harmadik csoport a PROMISE adattár adatállományaiból lettek felhasználva. A PROMISE-adatkészletből hat nyílt forráskódú Java-projektet választottunk ki. Valamennyi projekt forráskódja és a hozzájuk tartozó PROMISE-adatok nyilvánosak. Ezek a projektek olyan alkalmazásokat fednek le, mint az XML-elemzők, szöveges keresőmotor-könyvtárak és adattovábbítási adapterek, és ezek a projektek hagyományos statikus metrikákkal rendelkeznek minden egyes Java-fájlhoz. (5.3 táblázat)

Projekt név	**verzió**	**példányok száma**	**hiba %**
ant	1.7	745	22.28%
camel	1.6	965	19.48%
ivy	2.0	352	11.36%

jedit	4.3	492	2.23%
log4j	1.2	205	92.19%
xerces	1.4	588	74.31%

5.3 *táblázat:* PROMISE dataset

Az értékelési eredmények általánosságának biztosítása érdekében a kísérleti adathalmazok különböző méretű és hibaarányú projektekből állnak (a hat projektben a maximális példányszám 965, a minimális példányszám pedig 205). Ezenkívül a minimális hibaarány 2,23%, a maximális hibaarány pedig 92,19%). Az 5.3. táblázat a kiválasztott projektek alapvető adatait mutatja be, beleértve a projekt nevét, a projektverziót, a példányok számát és a hibaarányt vagy a hibás példányok százalékos arányát.

5.4.2 Adathalmazok kiegyensúlyozása

Az adathalmazok kiegyensúlyozása fontos feladat az adatelemzésben. A *bugok* természetéből adódóan a kódtárak egyes bejegyzései kis valószínűséggel tartalmaznak hibás kódot, a *commitok* döntő része helyes módsításokat tartalmaz. Ha egy adatbányászati módszert ilyen adatokkal tanítunk, akkor a modellek nem tudják a hibákat megfelelően általánosítani a kevés adat miatt és az előrejelzés pontossága nagyban lecsökken. A kutatás időszakában számos módszert kerestünk és sikerrel alkalmaztuk a hibás esetek számának mesterséges növelésére.

Az adathalmaz eloszlását különböző adatmintavételi módszerek alkalmazásával módosítottuk, mint például a Near Miss, a Random oversampling, a SMOTE, a SMOTE-Tomek Links és a Tomek Links módszerek. [T5.2] [T5.3] [T5.4] *Terjedelmi korlátok miatt itt ezt nem részletezem*.

5.5 Modellalkotás és kiértékelés

A kutatás kezdeti fázisában hagyományos adatbányászati módszereket teszteltünk, majd később áttértünk a ma divatos *deep learning* technikákra is.

Az képfelismerésnél széleskörben és sikeresen alkalmazott konvolúciós neurális hálót CNN, a szöveggenerálás közkedvelt LSTM (long-short term memory) modelljét, majd ennek kétirányú változatát Bi-LSTM és a továbbfejlesztésének tekinthető 2014-ben publikált GRU (Gated recurrent unit) háló alapján építettünk modelleket. A következő táblázat mutatja az egyes modellek fontosabb paramétereit.

Paraméterek	**Modellek**			
	Bi-LSTM	**LSTM**	**CNN**	**GRU**
Cell type (Bidirectional)	LSTM (64, 32), return_sequences =True	LSTM (64, 32), return sequences=True	-	-
Layers. GRU	-	-	-	100
Activation function	ReLU + sigmoid	ReLU + sigmoid	ReLU + Sigmoid	Tanh + Sigmoid

Dropouts	0.2	0.2	0.2	0.2
Dense	64, 1	64, 1	10, 1	1
Optimizer	Adam	Adam	Adam	Adam
Learning Rate	0.01	0.01	0.01	0.01
Loos Function	Mean squared error (MSE)	Mean squared error (MSE)	Mean squared error (MSE)	Mean squared error (MSE)
Batch Size	64	64	25	64
Epochs	100	100	100	100
Validation Split	0.1	0.1	0.1	0.1
Verbose	1	1	-	1

5.4 *táblázat:* A négy modell paraméterei

A modellek tesztelése során sikerült több esetben jobb előrejelzési pontosságot elérni, mint az irodalomban talált korábbi módszerek. Sikerült a tesztekkel is bizonyítani, hogy az adatkiegyenlítés jelentősen javítja a modellek teljesítményét. A következő táblázatban látható, hogy az LSTM és GRU modellek az eredeti és a kiegyenlített adatokkal valóban jobban teljesítenek.

Eredeti adathalmaz								
Modell	**Teljesítmény mérőszámok**							
	Accuracy	**Precision**	**Recall**	**F-measure**	**MCC**	**AUC**	**AUCPR**	**MSE**
LSTM	0.78	0.62	0.18	0.28	0.24	0.75	0.49	0.152
GRU	0.78	0.61	0.22	0.33	0.27	0.75	0.49	0.152
Balanced Dataset								
Modell	**Teljesítmény mérőszámok**							
	Accuracy	**Precision**	**Recall**	**F-measure**	**MCC**	**AUC**	**AUCPR**	**MSE**
LSTM	0.88	0.94	0.81	0.87	0.76	0.93	0.95	0.090
GRU	0.88	0.94	0.81	0.87	0.76	0.93	0.95	0.093

5.5 *táblázat:* Kiegyenlítettlen és kiegyenlített adatokon való futtatások mérőszámai

5.6 Összefoglalás

Ebben a fejezetben visszatértünk a szoftvertechnológia területéhez, ahol 2019 vége óta egy új kutatási területre is összpontosítottam, amely a modern mesterséges intelligencia módszereinek alkalmazását is magában foglalta. A témát általánosságban angolul „repository mining”-nak, vagy magyarul a „kódtárak adatbányászat”-nak hívjuk. A kutatás később ennek egy részterületére, mégpedig a szoftverhibák keletkezésének okaira irányult. A kódtárak elemzése 2016 körül vált tudományos kutatás tárgyává, és a jól ismert Git kódtár fejlesztése Linus Torwaldhoz köthető. A szoftverekben mindenki által ismert fogalom - a bug – mellett az újabban

code smell néven ismert finomabb, de a későbbi továbbfejlesztésben nagy valószínűségben bug-ot okozó, vagy a karbantartást megnehezítő problémák felismerével is foglalkoztam.

A fejezetben továbbá részletezem a szoftverhibák előrejelzésének (SDP) fontosságát, amely a szoftverminőség javítását és a fejlesztési költségek csökkentését célozza meg, különféle technikák, mint a statisztikai modellek, gépi tanulási algoritmusok, és adatbányászati módszerek alkalmazásának kombinálásával. Megkülönböztetem a szoftverprojekten belüli és a projektek közötti hibajóslást, valamint a *code smell*-ek jelentőségét tárgyaltam, amelyek tervezési hiányosságokat jeleznek a forráskódban. A fejezet végén, különböző deep learning technikák, mint a konvolúciós neurális hálózatok (CNN), az LSTM, a Bi-LSTM, és a GRU hálózatok alkalmazásával készült modellek eredményeit ismertetem, amelyek általában jobb előrejelzési pontosságot értek el az adatkiegyenlítés használatával. A kutatás jelenleg a transformer hálózatok alkalmazásával foglalkozik, a nyelvi modellek kedvelt hálózattípusa remélhetőleg jól teljesít a szoftverminőség területén is.

5.7 Témához kapcsolódó publikációk

[T5.1] N.A.A.Khleel and K.Nehéz, "Improving the Accuracy of Recurrent Neural Networks Models in Predicting Software Bug Based on Undersampling Methods", Indonesian Journal of Electrical Engineeringand Computer Science. Vol.32, No.1, pp. 478-493,2023. DOI: http://doi.org/10.11591/ijeecs.v32.i1.pp 478-493.

[T5.2] N. A. A. Khleel and K. Nehéz, “A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method,” JOURNAL OF INTELLIGENT INFORMATION SYSTEMS, 2023.

[T5.3] A. A. K. Nasraldeen and N. Károly, “Data balancing methods in ML-based software bug prediction,” in Doktoranduszok Fóruma 2022, 2023, pp. 59–67.

[T5.4] N. A. A. Khleel and K. Nehéz, “Deep convolutional neural network model for bad code smells detection based on oversampling method,” INDONESIAN JOURNAL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, vol. 26, no. 3, pp. 1725–1735, 2022.

[T5.5] A. A. K. Nasraldeen and N. Károly, “A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory,” PRODUCTION SYSTEMS AND INFORMATION ENGINEERING, vol. 10, no. 3, pp. 1–15, 2022.

[T5.6] A. A. K. Nasraldeen and N. Károly, “Overview of modern software bug prediction approaches,” in Doktoranduszok fóruma 2021 - Gépészmérnöki és Informatikai Kar szekciókiadványa, 2022, pp. 55–61.

[T5.7] N. A. A. Khleel and K. Nehéz, “Comprehensive Study on Machine Learning Techniques for Software Bug Prediction,” INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, vol. 12, no. 8, pp. 726–735, 2021.

[T5.8] N. A. A. Khleel and K. Nehéz, “Merging problems in modern version control systems,” MULTIDISZCIPLINÁRIS TUDOMÁNYOK: A MISKOLCI EGYETEM KÖZLEMÉNYE, vol. 10, no. 3, pp. 365–376, 2020.

[T5.9] N. A. A. Khleel and K. Nehéz, “Comparison of version control system tools,” MULTIDISZCIPLINÁRIS TUDOMÁNYOK: A MISKOLCI EGYETEM KÖZLEMÉNYE, vol. 10, no. 3, pp. 61–69, 2020.

[T5.10] A. A. K. Nasraldeen and N. Károly, “Mining Software Repository: An Overview,” in Doktoranduszok Fóruma, 2020, pp. 108–114.

[T5.11] K. Nehéz and N. A. A. Khleel, “Tools, processes and factors influencing of code review,” MULTIDISZCIPLINÁRIS TUDOMÁNYOK: A MISKOLCI EGYETEM KÖZLEMÉNYE, vol. 10, no. 3, pp. 277–284, 2020.

5.8 Felhasznált irodalom

[1] Omri S, & Sinz C, (2020) Deep learning for software defect prediction: A survey. In Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops (pp. 209–214). New York, NY, USA: Association for Computing Machinery.

[2] Ferenc, R., Bán, D., Grósz, T., et al. (2020). Deep learning in static, metric-based bug prediction. Array, 6, 100021.

[3] Tong, H., Liu, B., & Wang, S. (2018). Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. Information and Software Technology, 96, 94–111.

[4] Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2020). BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. Expert Systems with Applications, 144, 113085.

[5] Kalaivani, N., & Beena, R. (2018). Overview of software defect prediction using machine learning algorithms. International Journal of Pure and Applied Mathematics, 118(20), 3863–3873.

[6] Li, Z., Jing, X. Y., & Zhu, X. (2018). Progress on approaches to software defect prediction. IET Software, 12(3), 161–175.

[7] Omri S, & Sinz C, (2020) Deep learning for software defect prediction: A survey. In Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops (pp. 209–214). New York, NY, USA: Association for Computing Machinery.

[8] SonarLint: A cast shall not remove any const or volatile qualification from the type of a pointer or reference, https://rules.sonarsource.com/cpp/RSPEC-859/, 2023

[9] C++ reference: the rule of three/five/zero, https://en.cppreference.com/w/cpp/language/rule_of_three, 2023

[10] D. K. Kim, “Finding bad code smells with neural network models,” International Journal of Electrical and Computer Engineering, vol. 7, no. 6, pp. 3613–3621, Dec. 2017, doi: 10.11591/ijece.v7i6.pp3613-3621.

[11] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, “Comparing and experimenting machine learning techniques for code smell detection,” Empirical Software Engineering, vol. 21, no. 3, pp. 1143–1191, Jun. 2016, doi: 10.1007/s10664-015-9378-4.

6 Hibrid evolúciós algoritmusok a termelésütemezési feladatokban

6.1 Bevezetés

Egy másik friss kutatási téma visszavezet a termelésinformatika és a *soft-computing* algoritmusok világába. 2020 végén megcéloztam újabb evolúciós algoritmusok alkalmazását az ütemezési feladatokra. Később 2021-ben további inspirációt Prof. Kóczy Lászlóval való együttműködés adta - akivel akkoriban kerültem kapcsolatba - és az ő javaslatára kezdtem el foglalkozni a diszkrét bakteriális memetikus algoritmus (DBMA) alkalmazásaival. Később két közös cikket is publikáltunk a témában. Ennek a sikeres időszaknak az összefoglalása a következő fejezet.

6.2 A flowshop ütemezési feladat

Az ütemezési feladatokkal kapcsolatos kutatások az erőforrás tervezés, a termelékenység javítása és a működési költségek csökkentése szempontjából is jelentős szerepet kapott a termelésinformatikában. A flowshop feladat a többgépes, több operációs ütemezési problémák egyik klasszikus feladattípusa, ahol a cél, a feladatok minimális átfutási idejének elérése az optimális indítási sorrendjük változtatásával. Az optimális sorrend megtalálásához sokgépes esetben nincs egzakt algoritmus, ezért gyakran használnak különféle heurisztikákat és evolúciós algoritmusokat egyaránt. 1993-ban ehhez a problémához már benchmark dataset-et publikáltak [1], így könnyebben összehasonlíthatóvá váltak az egyes algoritmusok. A lehetséges sorrendek száma *n!* így egy nagyobb pl. 50 feladatos, 10 gépes probléma esetén esélytelen egy adott sorrend javítása, pusztán véletlen próbálgatással, még a modern hardvereken is.

Az elmúlt években a metaheurisztikus algoritmusok területének gyors növekedésének lehettünk szemtanui. Az „sima" evolúciós megközelítésekhez, mint például a didaktikailag fontos genetikai algoritmus (GA) javítására, egy sor kombinált módszert javasoltak, amelyek növelték az algoritmus hatékonyságát. Fontos lépés volt Moscato és mások ötlete [2], egy új memetikus algoritmuscsalád, a memetikus algoritmuscsalád, amely a hagyományos matematikai optimalizálási technikákat alkalmaz a helyi (lokális) kereséshez, miközben megtartja az evolúciót a globális keresésnél, és az előbbit a globális keresésbe ágyazza. Ezt a koncepciót kiterjesztették a diszkrét problémákra, és különböző egyszerű gráfelméleti és hasonló kimerítő optimalizálási technikákat alkalmaztak lokális keresésre. Ennek a kombinációnak az az előnye, hogy az evolúciós technikák általában jó eredményekhez vezetnek, de meglehetősen lassúak, míg a klasszikusabb optimalizálással lehet sokkal gyorsabbak; viszont hajlamosak „beragadni" a lokális optimumokban. Kutatásunk során arra kerestük a választ, hogy milyen lokális kereső algoritmusokat alkalmazzunk, amivel tovább növelhetjük a hatékonyságot. A [T6.2]-ben publikált megközelítésünkben a szimulált hűtéssel kombináltuk a DBMA-t és kimutattuk, hogy jobb eredményeket kapunk, mint a kombinálatlan változatokban alapváltozatokban. A [T6.1]-ben tovább finomítottuk az eljárást.

6.3 Rövid irodalmi áttekintés

Korán kimutatták, hogy a flowshop feladat *NP-complete* problémacsaládba tartozik, ha az erőforrások száma kettőnél nagyobb, a feladat nem oldható meg matematikailag zárt alakban. Általában úgynevezett *meta-heurisztikákat* alkalmaznak, amelyek a sokdimenziós keresési térben szisztematikusan „lépkednek”, olyan irányokat keresnek amely mentén a célfüggvény javul. Az ütemezési faladatoknál is gyakori, hogy a keresési tér lokális optimumokat tartalmaz, így az alap algoritmusok képtelenek megtalálni az optimumot, mert beragadnak egy lokális optimum környezetében, vagy „átugorják” a globális optimumot.

Az egyik legtöbbet hivatkozott eljárás a NEH-Tailard algoritmus, ami a Nawaz–Enscore–Ham (NEH) algoritmuson alapul [3]. A szimulált hűtés (SA) egy másik széles körben használt módszer. Az elnevezés egy analógiára utal a hőkezelési folyamatokkal, mint például az ötvözetek edzése vagy az üveg hűtése. A folyamat lényege az, hogy kezdetben magas hőmérsékleten indítjuk az algoritmust, ami lehetővé teszi, hogy nagyobb ugrásokkal lépjünk a keresési térben. Ahogy az algoritmus halad előre, fokozatosan csökkentjük a hőmérsékletet, ami a közelítőleg optimális megoldásig jut [4]. A "Particle Swarm Optimization" (PSO) egy populációs alapú optimalizációs algoritmus, amely az részecske rajok viselkedését modellezi és a kollektív intelligencia elveire épül. A PSO azokat a problémákat célozza meg, ahol a megoldási tér összetett és sokdimenziós, és nem mindig könnyű megtalálni a globális optimumot. A részecskék egyfajta kollektív tudatosságot fejlesztenek ki, és az egyének tapasztalataiból tanulva keresik meg a jobb megoldásokat. Ennek a módszernek is számos variáns létezik. A *Jaya* optimalizációs algoritmus egy paraméter nélküli optimalizálási technika, amely nem igényli semmilyen konkrét paraméter vagy vezérlő változó beállítását. Egyszerűsége és a feltárás és a kiaknázás közötti egyensúly megteremtésének képessége miatt hatékony megoldást nyújt különböző optimalizálási problémák megoldására. Lehet, hogy nem garantálja a globális optimumot, de gyakran ésszerű időn belül jó minőségű megoldásokhoz konvergál számos valós probléma esetében [5].

A genetikus algoritmusok alapvetően az biológiai evolúció egy fajon belüli alkalmazkodási folyamatát modellezik, ahol a populáció egyedei között öröklődik és változik a génállomány. Reprodukció, mutáció és keresztezés révén új egyedek jönnek létre. Ezek az új egyedek lehetnek potenciálisan jobbak a probléma megoldásában, és így az algoritmus a generációk létrehozása és versenyeztetése során próbálja megtalálni a legjobb megoldást. [6] Korán megfigyelték, hogy sok feladatban a klasszikus GA alkalmazása nem vezet elég jó eredményre ezért a módszert finomították, kombinálták úgynevezett lokális keresőkkel. [7]

6.4 Bakteriális memetikus algoritmus (DBMEA)

A bakteriális memetikus algoritmus [T6.3] egy olyan számítógépes eljárás, melyet a biológiából ismert bakteriális szaporodási mechanizmus ihletett. [8] A modell, a baktériumok szaporodásakor, a természetben megfigyelhető génkereszteződési folyamatot utánozza. Ennek

a folyamatnak két lényeges művelete van: a bakteriális mutáció és a gén-transzfer. Az algoritmus indulásakor egy kezdeti, véletlen tulajdonságokkal rendelkező populációt hozunk létre, majd a két művelet segítségével a rákövetkező generációban újabb egyedeket hozunk

létre. A második generáció leszármazottait egy célfüggvény segítségével egyesével kiértékeljük, egyes egyedeit elhagyjuk, míg másokat a tulajdonságaik alapján priorizálunk, majd a vázolt lépéseket a leállási kritérium eléréséig ismételjük.

Az első implementációkban a bakteriális evolúciós algoritmusokat (BMEA), fuzzy-szabály alapú rendszerek optimális paramétereinek kereséséhez használták. Az évek során kiderült, hogy hatékonyan használhatók más optimalizálási feladatok esetén is, például interaktív ápoló ütemezés problémára [9], automatikus adat-csoportosításra [10], és a háromdimenziós láda pakolás problémára [T6.4] egyaránt. A kezdeti definíciók szerint, a memetikus algoritmusok olyan módosított genetikus algoritmusok, melyek egy további lokális kereső algoritmust is alkalmaznak. Annak az ötlete, hogy a bakteriális evolúciós algoritmusokat kombinálják lokális kereső algoritmusokkal azért született, hogy az optimalizációs képességet jobban növeljék, a fuzzy szabályok becslésénél. A lehetséges alkalmazások tárát kiterjesztették további területekre is. Műszaki, kémiai és villamosmérnöki problémákra is teszteltek transzcendentális matematikai függvények mellett, amelyeknél a lokális kereső egy másodrendű gradiens alapú megoldás volt (a Levenberg-Marquard algoritmus). Az eredmények jobbak lettek, mint a korábbiak, még annál a megközelítésnél is, amelyet a szakirodalomban a trapéz alakú fuzzy tagsági függvények paramétereinek meghatározásához használnak. Később, elsőrendű gradiens alapú megoldásokat is teszteltek, mint lokális kereső algoritmusok és az eredmények jónak ígérkeztek [11].

A BMEA algoritmusok ötlete diszkrét, permutáció alapú problémákon is tesztelésre kerültek, ahol természetesen a lokális kereső algoritmusok is diszkrét módszerek. Az első javasolt operátor csoport az *n-opt* lokális keresők voltak, melyeket néhány szimuláció után a 2-opt és 3-opt egymásutáni használatára szűkítették. Az $n \geq 4$ eseteiben a számítási igény túl nagynak bizonyult, ezzel nagy mértékben rontva az algoritmus hatékonyságát. A *2-opt* operátort először az utazó ügynök problémára használták [12], itt a gráf két élét cserélik meg. A *3-opt* hasonló a *2-opt* operátorhoz; az egyetlen különbség az az, hogy ebben az esetben három élet cserélnek ki egy lépésben.

6.4.1 Az algoritmus bemutatása

Az első lepés a véletlenszerű kezdeti populáció generálása történik, melyben az egyes baktériumok a feladat lehetséges megoldásai (permutációk). A második lepésben a bakteriális

mutációt alkalmazzuk. A harmadik lépésben a lokális keresőt használjuk (másik nevén ez a memetikus lépés). A lokális keresés megpróbálja tovább javítani (finomítani) az egyes megoldásokat (pl: szomszéd megoldások generálásával), amíg nem talál egy jobb szomszédot.

Az alkalmazott jelölések:

- *P*: poluláció,

- x_1: aktuális megoldás,
- x_{best}: eddigi legjobb megoldás,
- f: fittness függvény,
- N_{clones}: klónok száma,
- I_{seg}: mutációs szegmensek száma,
- N_{inf}: fertőzések száma a gén-traszferben,
- I_{trans}: a gén-transzfer szegmensek hossza.

A harmadik lépés befejeződik, ha nem találunk jobb egyedet a szomszédok között. A negyedik lépésben a gén-transzfer mutációt végezzük el. Az algoritmus a 2-4 lepéseit addig ismételjük, amíg a leállási kritériumot el nem éri. Végül visszatérünk a legjobb megoldással.

1. Algorithm Diszkrét Bakteriális Memetikus Evolúciós algoritmus

procedure DBMEA
  Step 1: generate initial population P
  while termination criteria is not met **do**
    Step 2: bacterial mutation (P, N_{clones}, I_{seg})
    Step 3: local search
    Step 4: x_1 = gene transfer (P, N_{inf}, I_{trans})
    if $f(x_1) < f(x_{best})$ **then**
      Step 5: $x_{best} = x_1$
  return x_{best}

16. ábra DBMA áttekintő lépései

6.4.2 A bakteriális mutációs eljárás

A bakteriális mutációt természetesen a teljes populáción el kell végezni. Input paraméterként: a populációt (P), a klónok számát (N_{clones}) és a szegmens hosszát (I_{seg}) kell megadni. A 2. lépésben meghatározott számú klónt (N_{clones}) készítünk minden baktériumból, majd az eredeti baktériumot szegmensekre bontjuk, ez részben koherens szegmensekkel, részben laza szegmensekkel történik, előre megadott valószínűséggel. Mindkét szegmens-variáns esetén végig megyünk a szegmenseken. Kiválasztunk egy nem mutált szegmenst. Először a szegmens elemeit tükrözzük, amivel megkapjuk az első klónt, aztán véletlenszerűen keverve az elemeket, kepézzük a többit. Ezzel a módszerrel összesen N_{clones} számú klónt képezünk. A mutációs algoritmus végén a legjobb klón veszi át az eredeti egyed helyét. A koherens szegmens mutáció esetén a szegmens elemei sorfolytonosan következnek (2a. ábra). Laza szegmens mutációnál nem feltétlenül egymás melletti elemek alkotják a szegmenst (2b. ábra).

2. Algorithm Bakteriális mutációs eljárás

procedure BACTERIAL MUTATION(P, N_{clones}, I_{seg})
  for all p in P **do**
    Step 1: create a random number between 0 and 1.
    Step 2: create N_{clones} clones of p and a random number r[0..1]
    if $r \leq COHERENT\_LOOSE\_RATE$ **then**
      Step 3: cut p into coherent segments with I_{seg} length
    else
      Step 4: cut p into loose segments with I_{seg} length
    Step 5: replace p with the best of the clones and p
  return P

17. ábra A bakteriális mutáció

6.4.3 Gén-traszfer eljárás

A gén-transzfer operációt a teljes populáción elvégezzük. Először, az egyedeket fittnesz értékeik alapján rangsoroljuk. Aztán a populációt két részre osztjuk, a felsőbbrendű és alsóbbrendű részekre a korábban felállított rangsor alapján. Következő lepésként a gén-transzfer mutációt végrehajtjuk N_{inf} alkalommal egy véletlenszerűen választott felsőbb- és alsóbbrendű egyeden. A gén-transzfer mutáció során egy véletlen kiválasztott I_{trans} hosszúságú szegmenst a felsőbbrendű baktériumból transzferalunk az alsóbbrendűbe úgy, hogy ne maradjon a mutált baktériumban duplikált elem.

Paraméterek:

• p_{src}: A forrás baktérium, melynek felhasználásával transzformálunk,

• p_{dst}: A cél baktérium, melybe áthelyezzük a forrásból származó szegmenst.

3. Algorithm Gén-transzfer eljárás

procedure GENE TRANSFER(P, N_{inf}, I_{trans})
  Step 1: sort the population according to fitness values
  Step 2: divide the population into superior and inferior parts
  for i to N_{inf} **do**
    Step 3: select a random bacterium from the superior part (p_{src})
    Step 4: select a random bacterium from the inferior part (p_{dst})
    Step 5: select a random segment from p_{src} with I_{trans} length
    Step 6: copy the segment into p_{dst} in a random position
    Step 7: eliminate duplicates in p_{dst}
  return P

18. ábra A gén traszfer eljárás lépései

6.5 A Monte-Carlo fakeresés algoritmus

A Monte Carlo fa keresés (MCTS) [T6.3] algoritmusa - eredetileg táblás játékokhoz lett kifejlesztve - egy keresőfát készít a vizsgált probléma modellezéséhez. Egy adott levél, egy

kezdőállást reprezentál, aminek gyerek levelei az ellenfél lehetséges válaszlépései. A keresés során a keresőfát bővítjük egy többlépéses módszer segítségével. Minden levél tárolja a rajta elérhető nyert játszmák és az összes próbálkozás számát, amit paraméterként felhasználunk a fa bejárására használt eljárásban.

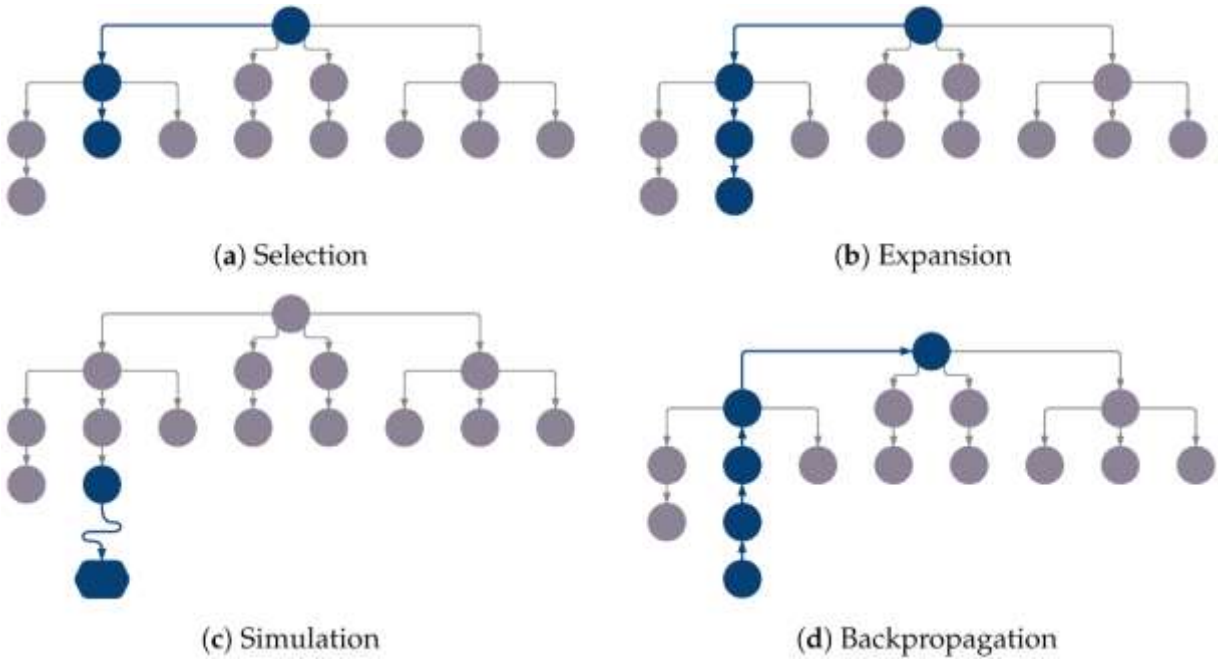
Az algoritmus az alábbi az alábbi négy lépést ismétli [41]:

1) Kiválasztás: a gyökértől kezdve fentről-lefelé rekurzívan kiválasztunk egy gyerek levelet (14a. ábra).

2) Kiterjesztés: egy következő véletlen gyermek levelet hozunk létre (14b. ábra).

3) Szimuláció: az új levélből kiindulva végig játszunk egy szimulált játszmát (14c. ábra). A játszma lépéseit jelképező gyerekeket nem hozzuk létre, hanem a kilépési feltétel alapján értékeljük a végállást: nyer, veszít vagy döntetlen (+1, -1, 0).

4) Visszacsatolás: a játszma értékelése alapján rekurzívan felfelé lépve frissítjük a levelek statisztikáit (nyert játszma és/vagy próbálkozások számának növelésével) (4d. ábra).



19. ábra A MCTS algoritmus 4 alaplépése

A kiválasztási lépésben az UCB (a 2.) formulát használjuk a gyerek levél kiválasztására.

$$UCB_j = \frac{X_j}{n_j} + C\sqrt{\frac{\ln N_j}{n_j}}, \tag{10}$$

Ahol X_j a j-edik gyerek levélen keresztül elérhető nyert játszmák számát jelöli, a C konstans szabályozza a kiválasztási stratégiát, és az N_j a j-edik szül ő levélen keresztül játszott összes játszma darabszáma. n_j a j-edik gyerek levélen keresztül játszott összes játszma darabszáma. Az algoritmus futása során, a fa fokozatosan és aszimmetrikusan növekszik a feltárási és hasznosítási stratégiák egyvelegeként. C konstans növelése/csökkentése befolyásolja a súlyt a

két stratégia között. *Feltárás* alatt azt értjük, hogy az algoritmusnak véletlenszerű választásra van szüksége arra, hogy bejárja a keresési teret. A *hasznosítás* alatt az ismert legjobb opció újrahasznosítását értjük. Az eredetileg két résztvevős táblás játékokra kifejlesztett algoritmust most ütemezési feladatokra alkalmazzuk. Az egyes levelek egy-egy permutációt, egy lehetséges ütemezési sorrendet jelképeznek. A levelek gyermekeit a szülő permutációból, módosító operációkkal hozzuk létre. A fa minden csomópontja egy lehetséges megoldás, melyek között szomszédsági reláció van. Azonos megoldást tartalmazó levelek többször is megjelenhetnek a fában, hasonlóan, mint a táblás játékok esetén.

6.6 Hibrid DBMEA algoritmus

A hatékonyság növelése érdekében érdemes algoritmusokat kombinálni, ötvözni a módszerek tulajdonságait. A DBMEA-t célszerű ott átalakítani, kiegészíteni, ahol az az 11. ábrán bemutatott algoritmusban a harmadik lépésben a lokális keresés történik. Az eredeti DBMEA algoritmus a *2-opt* és *3-opt* módszereket alkalmazza, melyek a flow shop feladatnál egyébként sem hatékonyak. A lokális kereső hatékonyságát úgy növeljük, hogy Monte-Carlo fa-keresést használunk, melyben az egyes játszmákat *szimulált hűtéssel* értékeljük ki (DBMEA + MCTS + SA). Az MCTS kiterjesztés lépésében definiáljuk a szimulált hűtés eljárás kiinduló permutációját. A játszma során jobb rosszabb, vagy egyforma permutációkat találunk. A SA alkalmazása miatt, a módszer rosszabb megoldásokat is el tud fogadni, ezzel kijutva a lokális optimumokból. A lokális minimumokból való kijutás elősegítésére, bevezetésre került továbbá egy halandósági (N_{mort}) ráta, ami azt jelzi, hogy a populáció bizonyos százaléka eldobásra kerül (meghal), és ezek helyébe új véletlenszerűen generált új egyedek kerülnek.

6.7 Futási eredmények

[13]-ban bemutatott klasszikus benchmarkot vettük alapul az algoritmusok kiértékelésére és összehasonlítására. Különböző algoritmusok átlagos teljesítményének az összehasonlításához az eddigi legjobb találathoz viszonyítjuk az eredményeket, a következőképpen.

$$\omega = \frac{C_{Bs} - C_{uB}}{C_{uB}} \cdot 100 \qquad (11)$$

Ahol C_{BS} jelöli az algoritmus által talált legjobb C_{max} értéket, C_{UB} pedig az eddigi ismert legjobb eredményt.

Az elért eredmények alapján látható, hogy az alkalmazott Hibrid Bakterialis Memetikus algoritmus a benchmark feladatban átlagosan kevesebb, mint *1%*-os eltéréssel megközelíti az ismert legjobb megoldásokat. Kilenc esetet közöltünk, ahol az ismert legjobb megoldásoknál jobb eredményt értünk el, ezekben az esetekben megadtuk az új C_{max} értéket és a permutációkat, így az eredményeink könnyen validálhatók.

6.8 Összefoglalás

A "Hibrid Evolúciós Algoritmusok a Termelésütemezési Feladatokban" című fejezet a termelésinformatika és a soft-computing algoritmusok területén végzett friss kutatásokat ismerteti, különös tekintettel a termelésütemezési feladatokra. A kutatás fókuszában a diszkrét bakteriális memetikus algoritmus (DBMA) és annak alkalmazása áll, amelyet Prof. Kóczy Lászlóval való együttműködés inspirált. A fejezet bemutatja a klasszikus flowshop ütemezési feladatokkal kapcsolatos kutatásokat, ahol az evolúciós algoritmusok és a metaheurisztikus megközelítések ötvözésével próbálják javítani a termelési folyamatok hatékonyságát. Kiemeli a bakteriális memetikus algoritmus és a Monte-Carlo fakeresési technikák kombinációjának előnyeit, amelyek hatékonyabb ütemezési eredményeket eredményezhetnek, mint a hagyományos módszerek. Az új megközelítések a nemzetközi benchmark teszteken is bizonyították hatékonyságukat, néhány esetben (9-ben a 120-ból) jobb eredményeket értünk el.

6.9 Témához kapcsolódó publikációk

[T6.1] Fazekas, B. Tüű-Szabó, L. T. Kóczy, O. Hornyák, and K. Nehéz, “A Hybrid Discrete Memetic Algorithm for Solving Flow-Shop Scheduling Problems”, ALGORITHMS, vol. 16, no. 9, p. 406, 2023.

[T6.2] A. Agárdi, K. Nehéz, O. Hornyák, and L. T. Kóczy, “A Hybrid Discrete Bacterial Memetic Algorithm with Simulated Annealing for Optimization of the Flow Shop Scheduling Problem,” SYMMETRY (BASEL), vol. 13, no. 7, 2021.

[T6.3] A. Agárdi and K. Nehéz, “Parallel Machine Scheduling With Monte Carlo Tree Search,” Acta Polytechnica, vol. 61, no. 2, pp. 307–312, 2021.

[T6.4] A. Agárdi and K. Nehéz, “The Unrelated Parallel Machines Scheduling Problem with Machine and Job Dependent Setup Times, Availability Constraints, Time Windows and Maintenance Times,” MANAGEMENT AND PRODUCTION ENGINEERING REVIEW, vol. 12, no. 3, pp. 15–24, 2021.

[T6.5] A. Anita and N. Károly, “Advanced Scheduling Model For Unrelated Parallel Machines Problem With Job-Sequence Dependent Setup Times, Availability Constraints And Time Windows,” ACADEMIC JOURNAL OF MANUFACTURING ENGINEERING, vol. 18, no. 2, pp. 20–27, 2020.

6.10 Felhasznált irodalom

[1] Eric Taillard. Benchmarks for basic scheduling problems. European journal of operational research, 64(2):278–285, 1993.

[2] Moscato, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. In Technical Report, Caltech Concurrent Computation Program Report 826; California Institute of Technology: Pasadena, CA, USA, 1989.

[3] Nawaz, M.; Enscore, E.E., Jr.; Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 1983, 11, 91–95.

[4] Van Laarhoven, P.J.; Aarts, E.H. simulated annealing. In simulated annealing: Theory and Applications; Springer: Berlin/Heidelberg, Germany, 1987; pp. 7–15.

[5] Gao, K.; Yang, F.; Zhou, M.; Pan, Q.; Suganthan, P.N. Flexible job-shop rescheduling for new job insertion by using discrete Jaya algorithm. IEEE Trans. Cybern. 2018, 49, 1944–1955.

[6] Shih-Hsin Chen, Pei-Chann Chang, TCE Cheng, and Qingfu Zhang. A self-guided genetic algorithm for permutation flowshop scheduling problems. Computers & operations research, 39(7), 2012.

[7] Lin-Yu Tseng and Ya-Tai Lin. A hybrid genetic algorithm for no-wait flowshop scheduling problem. International journal of production economics, 128(1):144–152, 2010.

[8] Norberto Eiji Nawa and Takeshi Furuhashi. Fuzzy system parameters discovery by bacterial evolutionary algorithm. IEEE Transactions on Fuzzy Systems, 7(5):608–616, 1999.

[9] Takeshi Inoue, Takeshi Furuhashi, Hiroshi Maeda, and Minoru Takaba. A study on interactive nurse scheduling support system using bacterial evolutionary algorithm engine. IEEJ Transactions on Electronics, Information and Systems, 122(10):1803–1811, 2002.

[10] Swagatam Das, Archana Chowdhury, and Ajith Abraham. A bacterial evolutionary algorithm for automatic data clustering. In 2009 IEEE Congress on Evolutionary Computation, pages 2403–2410. IEEE, 2009.

[11] János Botzheim, Cristiano Cabrita, László T Kóczy, and AE Ruano. Fuzzy rule extraction by bacterial memetic algorithms. International Journal of Intelligent Systems, 24(3):312–339, 2009.

[12] Luc Muyldermans, Patrick Beullens, Dirk Cattrysse, and Dirk Van Oudheusden. Exploring variants of 2-opt and 3-opt for the general routing problem. Operations research, 53(6):982–995, 2005.

[13] Eric Taillard. Benchmarks for basic scheduling problems. European journal of operational research, 64(2):278–285, 1993.

7 SUMMARY

Dr Nehéz Károly Róbert's habilitation thesis booklet, "Applied Informatics Research: Model Optimization and Applications of Artificial Intelligence," focuses on the main topics of each chapter:

Inventory Management Models: this part of the thesis booklet explores various inventory management models, including an in-depth discussion of the classical newsvendor model and its extensions. It presents analytical approaches and the concept of penalty costs in inventory management. The chapter further examines the application of these models in different scenarios, including multi-period extensions and specific cost models, along with methods for modelling product phase-outs.

MOOC Systems - this chapter discusses the role of MOOCs (Massive Open Online Courses) in the evolution of online education, particularly highlighting their significance during the pandemic. It delves into the challenges and solutions for automatic program evaluation in online learning environments, emphasizing the importance of active practice and immediate feedback for students. The chapter also describes the implementation and effectiveness of the MeMOOC system, a platform specifically designed for online programming courses.

Automatic Recognition of Certain Graphic Elements in CAD drawings: this chapter focuses on the automatic recognition of graphic elements, emphasising optical character recognition. The chapter reviews various methods and technologies used in this field and discusses developing and applying proprietary processes. It also reviews the use of these technologies in multi-user interactive applications, highlighting their practical applications and the evolution of this field.

Artificial Intelligence Methods in Software Quality Assurance: this chapter addresses the application of artificial intelligence in software quality assurance. It covers the prediction of software bugs and the detection and significance of "code smells" in software development. The chapter discusses the use of software metrics in this context. It examines the model creation and evaluation process, underscoring AI's role in enhancing the quality and reliability of software products.

Hybrid Evolutionary Algorithms in Production Scheduling Tasks: focuses on integrating production informatics with soft computing algorithms, highlighting the application of new evolutionary algorithms to scheduling challenges. A significant advancement was the exploration of the Discrete Bacterial Memetic Algorithm (DBMA). This chapter discusses the flow shop scheduling problem, a multi-machine, multi-operation scheduling issue in production informatics aimed at minimizing task completion times through optimal job sequences. The absence of exact algorithms for complex cases necessitates using various heuristics and evolutionary algorithms. Due to the complexity of these problems and the lack of exact algorithms for more intricate scenarios, a combination of heuristics and evolutionary algorithms, including the development of the Hybrid DBMEA algorithm, has been employed.

8 KÖSZÖNETNYILVÁNÍTÁS

Mindenekelőtt szeretnék köszönetet mondani Dr. Tóth Tibor professzor emeritusnak témavezetőmnek, aki korábbi intézetigazgatóként a PhD fokozatszerzésem óta támogatott a szisztematikus tudományos építkezésben, valamint a nemzetközi szintű publikálásban. Köszönetemet szeretném kifejezni Dr. Csáki Tibor egyetemi docens doktori témavezetőmnek, akitől sok segítséget kaptam PhD munkám során, valamint Dr. Velezdi György egyetemi adjunktusnak, aki rávezetett a gyakorlati munka fontosságára és megtanultam tőle a mérnöki munka és kutatómunka közötti szakadék áthidalásának fontosságát.

Köszönetemet szeretném kifejezni az informatika intézeti kollégáknak a szakmai konzultációkért, valamint a témavezetésem alatt álló doktorandusz kollégák, Dr. Mileff Péter, Szabó Martin és Nasraldeen Alnor Khleel, és az idén kezdett új aspiránsom, Fazekas Levente által nyújtott folyamatos szakmai motivációt. Külön kiemelem Dr. Hornyák Olivér barátom segítségét, aki percíz szakmaisága mellett baráti segítséget is nyújtott a holtpontok átvészelésében.

Végezetül kiemelt köszönetet szeretnék mondani feleségemnek, Noéminek, valamint lányomnak, Noának, hogy türelmükkel hozzájárultak a tudományos kutatómunka sok esetben munkaidőn túlmutató részének elvégzéséért. Továbbá ezúton is köszönöm szüleimnek, aki gyermekkorom óta kiemelten támogatták a szellemi fejlődésemet.

Miskolc, 2024. március 20. Dr. Nehéz Károly