Pannon Egyetem

Műszaki Informatikai Kar

Rendszer- és Számítástudományi Tanszék

mérnökinformatikus BSc

SZAKDOLGOZAT

Vendéglátói szálláshelyek csoportos foglalását megvalósító rendszer fejlesztése

Rozsenich Balázs

Témavezető: Frits Márton

2015

Feladatkiírás

Nyilatkozat

Alulírott *Rozsenich Balázs* hallgató, kijelentem, hogy a dolgozatot a Pannon Egyetem *Rendszer- és Számítástudományi tanszék* tanszékén készítettem a *mérnökinformatikus* végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Veszprém, 2015. április \_\_

Rozsenich Balázs

Alulírott *Frits Márton* témavezető kijelentem, hogy a dolgozatot *Rozsenich Balázs* a Pannon Egyetem *Rendszer- és Számítástudományi tanszék* tanszékén készítette *mérnökinformatikus* végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védésre bocsátását engedélyezem.

Veszprém, 2015. április \_\_

Frits Márton

Köszönetnyilvánítás

TARTALMI ÖSSZEFOGLALÓ

Az összefoglalónak tartalmaznia kell (rövid, velős és összefüggő megfogalmazásban) a következőket:

• téma megnevezése,

• megoldott feladat megfogalmazása,

• megoldási mód,

• elért eredmények,

• **Kulcsszavak:** (4-6 darab)

ABSTRACT

Tartalomjegyzék

[Feladatkiírás 2](#_Toc416943584)

[Nyilatkozat 3](#_Toc416943585)

[Köszönetnyilvánítás 4](#_Toc416943586)

[TARTALMI ÖSSZEFOGLALÓ 5](#_Toc416943587)

[ABSTRACT 6](#_Toc416943588)

[Tartalomjegyzék 7](#_Toc416943589)

[1. Bevezetés 10](#_Toc416943590)

[1.1. A probléma és megoldása 10](#_Toc416943591)

[2. Szálláskereső portálok 12](#_Toc416943592)

[2.1. Szallas.hu 12](#_Toc416943593)

[2.2. Booking.com 12](#_Toc416943594)

[2.3. Trivago.hu 13](#_Toc416943595)

[2.4. Konklúzió 13](#_Toc416943596)

[3. Matematikai optimalizálás 14](#_Toc416943597)

[3.1. A matematikai optimalizálás története 14](#_Toc416943598)

[3.2. Matematikai optimalizálási feladat 18](#_Toc416943599)

[3.3. Lineáris optimalizálási feladat 19](#_Toc416943600)

[3.4. Nemlineáris programozási feladat 22](#_Toc416943601)

[4. Ruby on Rails 23](#_Toc416943602)

[5. Specifikáció 24](#_Toc416943603)

[5.1. Szereplők 24](#_Toc416943604)

[5.2. Funkcionális követelmények 24](#_Toc416943605)

[5.2.1. Felhasználói fiókok 24](#_Toc416943606)

[5.2.2. Szobák szűrése 24](#_Toc416943607)

[5.2.3. Szobafoglalás 25](#_Toc416943608)

[5.2.4. Értékelés 25](#_Toc416943609)

[5.2.5. Intelligens keresés 25](#_Toc416943610)

[5.2.6. Törzsadatok 25](#_Toc416943611)

[5.2.7. Tartós címek 25](#_Toc416943612)

[5.3. Célcsoport 25](#_Toc416943613)

[6. Tervezés 27](#_Toc416943614)

[6.1. A rendszerben megjelenő fő folyamatok 27](#_Toc416943615)

[6.1.1. Szobafoglalás 27](#_Toc416943616)

[6.1.2. Foglalás visszaigazolás 28](#_Toc416943617)

[6.1.3. Intelligens keresés 28](#_Toc416943618)

[6.2. Nemlineáris optimalizálási modell 29](#_Toc416943619)

[6.2.1. Olcsó modell 32](#_Toc416943620)

[6.2.2. Közeli modell 32](#_Toc416943621)

[6.2.3. Olcsó és közeli modell 33](#_Toc416943622)

[6.3. Adatbázis tervezet 34](#_Toc416943623)

[6.4. Technológia 36](#_Toc416943624)

[6.4.1. Keretrendszer 36](#_Toc416943625)

[6.4.2. Adatbázis 37](#_Toc416943626)

[6.4.3. Optimalizálási modellezés 37](#_Toc416943627)

[6.4.4. Nemlineáris megoldó 38](#_Toc416943628)

[6.4.5. Megjelenés 39](#_Toc416943629)

[6.4.6. Autentikáció és autorizáció 40](#_Toc416943630)

[6.4.7. Geolokáció 41](#_Toc416943631)

[6.4.8. Űrlapok 41](#_Toc416943632)

[6.4.9. Képek tárolása és megjelenítése 42](#_Toc416943633)

[7. Megvalósítás 43](#_Toc416943634)

[7.1. Adatbázis kapcsolat és modellek elkészítése 43](#_Toc416943635)

[7.2. Autentikáció és autorizáció 46](#_Toc416943636)

[7.3. Szobák szűrése 46](#_Toc416943637)

[7.4. Intelligens keresés 47](#_Toc416943638)

[7.5. Szobafoglalás 52](#_Toc416943639)

[8. Felületek és használat 53](#_Toc416943640)

[8.1. Menüsáv 53](#_Toc416943641)

[8.2. Szobák 53](#_Toc416943642)

[8.3. Szálláshelyek 53](#_Toc416943643)

[8.4. Foglalások 53](#_Toc416943644)

[8.5. Intelligens keresés 53](#_Toc416943645)

[8.6. Kosár 53](#_Toc416943646)

[8.7. Adminisztrációs felületek 53](#_Toc416943647)

[9. Tesztelés 54](#_Toc416943648)

[9.1. Tesztelési környezet 54](#_Toc416943649)

[9.2. Teszt adatok 54](#_Toc416943650)

[9.3. Teszt eredmények 54](#_Toc416943651)

[10. Összefoglalás 55](#_Toc416943652)

[Irodalomjegyzék 56](#_Toc416943653)

[Ábrajegyzék 57](#_Toc416943654)

[Mellékletek 58](#_Toc416943655)

[11. Ruby on Rails fejlesztői környezet telepítése és konfigurálása 58](#_Toc416943656)

[CD Melléklet 59](#_Toc416943657)

# Bevezetés

A szakdolgozatom témája egy olyan webalkalmazás elkészítése, amely csoportok (pl.: osztályok, baráti vagy üzleti társaságok) számára teszi egyszerűbbé a több szálláshelyen történő szobafoglalás menetét és kezelését. Az alkalmazás szempontjából fontos az internetes platform, mert így lehet a legolcsóbban a legszélesebb felhasználói körnek elérhetővé tenni. A projekt munkacímének a *VAGATO* szót választottam, amelyet a katalán *vaganto* (jelentése: barangolás) szóból képeztem.

## A probléma és megoldása

A csoportos turizmus jelentős szereppel bír a turizmusban, gondoljunk csak a tavasszal és ősszel, százával kirándulni induló diákokra, a közös szórakozásra vágyó baráti társaságokra, vagy az egyéb, üzleti célból utazó társaságokra. Egy csoport számára, különösen főszezonban kivételesen nehéz mind árban, mind távolságban megfelelő szálláshelyet találni, illetve gyakran előfordul, hogy egy szálláshely nem képes megfelelő számú kapacitást kínálni. A kapacitás korlátja lehet az aktuális foglaltság, vagy – jellemzően kisebb településeken – a szálláshelyek alapvető szobakínálatának csekélysége. Ilyen helyzetekben az utazásszervező feladata az, hogy összegyűjtse a szálláshelyek ajánlatait és az idővel versengve kalkulációk útján kiválassza a megfelelő szálláshelyek megfelelő szobáit.

A szálláshelyek kiválasztása után az utazásszervező szembesül a következő problémával. Minden szálláshely egyedileg kezeli a foglalásokat, az utazásszervezőnek minden szálláshellyel külön-külön kell megegyeznie. Ez rengeteg, egymástól független ügyintézést és papírmunkát jelent és jelentősen megbonyolítja a folyamatot.

Az általam tervezett webalkalmazás a fent vázolt problémákat igyekszik feloldani és használható megoldást kínálni. A koncepció az, hogy a jelenleg szálláshely orientált piacot meg kell fordítani és a középpontba a szobákat kell helyezni. A szobának, csakúgy, mint a légkondicionálás vagy az ellátás, csak egy tulajdonsága az, hogy mely szálláshelyhez tartozik. A szálláshelyek adta kötöttségek feloldásával már könnyű elképzelni egy olyan portált, ami a szobákat, mint egy *webshop*-ban, termékekként sorolja fel. A szobák a szálláshelyektől függetlenül kereshetők, szűrhetők és foglalhatók. A szobák e fajta individuális termékként való kezelése a kulcs ahhoz, hogy az utazásszervező olyan foglalásokat tudjon összeállítani, amiben egyszerre jelenik meg több szálláshely több szobája egy közös felületen.

A szobafoglalás folyamatát tehát most már el lehet képzelni úgy, hogy az utazásszervező a portált böngészve, egy virtuális kosárba helyezi a kellő szobákat. A böngészés végén a kosarában lévő szobákat egy foglalássá egyesíti és a vendégadatok megadása után véglegesíti azt.

A szobák kiválasztásának folyamata bonyolult, azonban jól automatizálható. A keresés szempontjai kitérnek a felszereltségre, a szolgáltatásokra és az elérhetőségre. Ezek a feltételek gyorsan és egyszerűen szűrhetők úgy, hogy az utazásszervező egy űrlapon megjelöli a kívánalmakat. A nagyobb nehézséget az ár, a minőség és a távolság feltételei adják. Az utazásszervező olyan szobákat akar, amik olcsók, ugyanakkor nincsenek távol egymástól; vagy a távolság nem számít, de legyenek minél jobb értékelésű szálláshelyeken. Az efféle szempontokhoz már nem elég szimplán sorrendbe állítani a szobákat és kiválasztani az első *N* darabot. Az optimális megoldás kísérletezés útján kézzel is elvégezhető, azonban kimondottan időigényes feladat. A webalkalmazásnak tehát rendelkeznie kell egy olyan funkcióval, ahol a kényelmi szempontok és a csoport létszáma szerint egy ár, távolság illetve minőség szerint optimális megoldást kap az utazásszervező arról, hogy mely szobákat kell lefoglalnia. A felvázolt funkciót a rendszerben *intelligens keresés*nek neveztem el.

# Szálláskereső portálok

Ebben a fejezetben a magyar szálláskereső piac legnépszerűbb portáljait vizsgálom meg a szerint, hogy milyen lehetőségeket kínálnak a szobák, illetve szálláshelyek keresésére, szűrésére. A vizsgálat tárgya továbbá, hogy mennyire támogatják a csoportos szálláskeresés 1.1 fejezetben bemutatott problémáit.

## Szallas.hu

A szallas.hu egy magyar alapítású és fejlesztésű szálláskereső portál, amely 2007 óta üzemel. A szallas.hu tekinthető a magyar szálláskereső piac legnépszerűbb szereplőjének. A szállásadók részére egységes megjelenést és könnyű foglalást ígér jutalékért cserébe.

A szálláskeresés során részletesen megadhatók a keresés feltételei hely, ár és szolgáltatások terén is. A találati listában szálláshelyek láthatók, egy szálláshelyet kiválasztva válnak láthatóvá az ajánlott szobák. Az utazó személyeket 30 felnőtt és 10 gyerek számosságban maximalizálták a keresés során. Egy foglalás csak egy szálláshely kínálatát tartalmazhatja.

A portál rendelkezik értékelési rendszerrel.

## Booking.com

A booking.com egy nemzetközi szálláskereső portál, amely 2011-ben lépett be a magyar szálláskereső piacra. A szallas.hu közvetlen riválisának tekinthető, szolgáltatásaik megegyeznek. ~~A szálláskeresők körében alacsonyabb népszerűséggel bír mint a szallas.hu.~~

A szallas.hu-hoz hasonlóan ezen a portálon is részletesen lehet szűrni a szálláshelyek tulajdonságait. A találatok között szintén a szálláshelyek jelennek meg, amelyeknek részletes leírásában tekinthetők meg a szobák. A foglalásban csak egy szálláshely szobái szerepelhetnek. A keresés során maximálisan 30 felnőtt és 10 gyerek választható.

A portál rendelkezik értékelési rendszerrel.

## Trivago.hu

A trivago.hu a Trivago nemzetközi szálláskereső szolgáltatás Magyarországra készült változata. A működése eltér az 2.1 és 2.2 fejezetekben tárgyalt portálokétól, ugyanis a Trivago csak összegyűjti más szálláskereső portálok ajánlatait és azok közül keres.

A keresési feltételekkel nagyvonalúan bánik, nem lehet elég részletesen beállítani a kívánalmakat. Lehet szűrni a teljes foglalás ára és a talált szálláshelyek városközponttól számított távolsága alapján. Az előző fejezetekben megvizsgált portálokhoz hasonlóan ez a rendszer sem képes a szobákat vegyesen ajánlani. A keresési találatok mindig egy-egy szálláshelyre vonatkoznak. A csoportos szálláskeresést csak korlátozottan támogatja. Az utazó személyek kiválasztásakor maximum 16 felnőtt és 16 gyerek választható.

## Konklúzió

A magyar szálláskereső piac portáljai jó felületet nyújtanak az egyéni utazók számára. A vizsgált portálok előnyben részesítik a szálláshelyeket és jellemzően egy szálláshelyre koncentrálják ajánlataikat. A keresési szempontokat mindhárom portál esetében kielégítőnek találtam. Az utazó személyek száma a keresés során mindenütt korlátozott. Egyik portál sem képes több szálláshelyről származó szobákat egy foglalásként kezelni.

A kutatásom során nem találtam olyan szálláskereső portált, amely funkcionalitásában közvetlen vetélytársa vagy alternatívája lehetne az általam felvázolt rendszernek.

# Matematikai optimalizálás

A matematikai optimalizálás a modernkori matematika egyik legfontosabb és leggyorsabban fejlődő ágazata. A matematikai optimalizálás az alternatívák halmazán történő legjobb választás problémájával foglalkozik. A problémát korlátozások és célok írják le. A cél meghatározza azokat a feltételeket, amiknek a legjobb választásnak meg kell felelnie. Átfogalmazva a matematikai optimalizálás egy valós függvény maximum vagy minimum értékének meghatározásával foglalkozik.

Optimalizálás helyett gyakran használatos a programozás megnevezés. A kifejezés nem egyenlő a számítógépes programozással. Az elnevezés Dantzig-tól származik, aki az 1940-es években az amerikai hadseregnél foglalkozott az ott programnak nevezett tréning és logisztika megszervezésének problémáin.

A következő fejezetekben bemutatom az optimalizálás kialakulását, a története során megemlítendő fontos személyeket és eseményeket, illetve kitérek a vonatkozó optimalizálási osztályokra.

## A matematikai optimalizálás története

Az optimalizálási feladatok, ha nem is nevezték nevén őket, már régóta foglalkoztatja a matematikusokat és gondolkodókat. Minden korban minden nagy birodalom és városállam vezetői szembesültek a mezőgazdaság és élelmezés, az élelmiszerelosztás problémáival. A hadjáratok során szintén nagy szükség volt kielégítő, de nem pazarló, vagyis végső soron optimális hadtáprendszer kialakítására. Optimalizálási feladatok megoldásával tehát már az ókorban és a középkorban is foglalkoztak a kor tudósai, matematikusai és mérnökei, jóllehet ők maguk ezzel nem voltak tisztában.

Kr.e. 300 körül Eukleidész, görög matematikus geometriai kutatásai során megoldotta a két pont legrövidebb távolságának és az élek teljes hosszával legnagyobb területet lefedő geometriai alakzat jelentette optimalizálási feladatokat. A kutatásának eredménye, hogy két pont között a legkisebb távolság az egyenes, míg a legnagyobb lefedhető terület korlátozott teljes él hosszal a négyzet. Kr. e. 100 körül Hérón, szintén görög matematikus és gépész *Catoptrica* című művében bebizonyította, hogy a tükörben tükröződő tárgyak fénye a lehető legrövidebb utat járja be. E felfedezés matematikai alapokra helyezve szintén optimalizálási feladatra vezethető vissza.

A variációszámítás kialakulásáig csak pár optimalizálási feladatot vizsgáltak a tudósok. A 17. és 18. században több említésre méltó esemény is köthető a matematikai optimalizáláshoz.

1615-ben Kepler kitalálja a kor szempontjai szerinti boroshordó optimális méretét. Ezen kívül megalkotja a titkárnő probléma, a dinamikus programozás egy népszerű alkalmazásának korai formuláját, amikor új feleséget keres magának. 1638-ban Galilei másodfokú egyenletekkel próbálja leírni a függő lánc vagy kötél alakját, de kísérletei kudarcot vallanak. Galilei ott ejtett hibát, hogy azt feltételezte a lengő lánc egy hiperbolát formál. Az 1690-ben Jacob Bernoulli által megfogalmazott láncgörbe probléma megoldása szintén matematikai optimalizálásra vezethető vissza. 1646-ban Fermat megmutatja, hogy egy függvény szélsőértékeinél annak gradiense eltűnik. 1657-ben általánosítja Hérón a fény útjára tett megállapítását, miként a fény bármely két pont között a lehetséges legrövidebb úton halad. Az 1660-as és 1670-es években Newton és Leibniz megalkotják a matematikai analízist, amely a variációszámítás alapjává válik. Ezt követően felgyorsulnak az optimalizálás területén tett felfedezések és áttörések eseményei. 1687-ben Newton a legkisebb légellenállású testet keresi, ami egy minimalizálási problémához vezet. 1696-ban Johann és Jacob Bernoulli a brachistochron probléma kutatása során megteszik az első lépéseket a variációszámítás megszületéséhez.

1712-ben König megmutatja, hogy a méhek által képzett méhsejt forma alakja optimális. Az eredményt a Francia Tudományos Akadémia isteni jelként aposztrofálja. 1740-től kezdve Euler publikációi nyomán globális figyelmet kapnak a variációszámítás területén végzett kutatások. 1746-ban [Maupertuis](http://hu.wikipedia.org/w/index.php?title=Pierre_Louis_Maupertuis&action=edit&redlink=1) megfogalmazza a legkisebb hatás elvét, amit arra a feltevésére alapoz, hogy a természetes mozgás szükségszerűen minimalizál valamilyen mennyiséget. 1754-ben a 19 éves Lagrange megteszi első felfedezéseit a variációszámítás területén. 1760-ban megfogalmazza Plateau minimális felületekre vonatkozó problémáját. 1930-ban egymástól függetlenül Jesse Douglass és Radó Tibor is megoldást talál a problémára. 1784-ben Monge elkezdi vizsgálni a szállítási problémát, amely egy népszerű optimalizálási feladat.

A 19- században Weierstrass, Steiner, Hailton és Jacobi a variációszámítás területén végzett mélyebb kutatásai nyomán megjelennek az első optimalizálási algoritmusok.

1806-ban Legendre bemutatja a legkisebb négyzetek módszerét, amelyet Gauss is magáénak tulajdonít. A módszer lényege az eltérések négyzetösszegének minimalizálása. 1826-ban Fourier lineáris programozási problémát fogalmaz meg mechanikában és valószínűség számításban felmerülő problémák megoldására. 1846 Faustmann kidolgoz egy formulát az erdők újratelepítésével realizálható bevétel maximalizálására. 1924-ben Bertil Ohlin megoldja Faustmann formuláját, mégha néhány erdész állítólagosan már az 1860-as években megoldotta azt. 1847-ben Cauchy megalkotja a gradiens módszert, amely egy optimalizálási algoritmus. 1857-ben Gibbs megmutatja, hogy a kémiai egyensúly egy energia minimum.

A közgazdaságtanban, az 1870-es években kialakuló határhaszon-elmélet Walras és Cournot munkája nyomán a közgazdászok figyelme a fogyasztói szükséglet maximalizálásra terelődik. Az optimalizálás a közgazdaságtan szerves részévé válik.

A variációszámítás és az optimalizálás területei igazi fejlődést a 20. században mutatnak. A század második felétől, a kutatások az elektronikus számítógépek megjelenésével felgyorsulnak.

1902-ben Farkas kidolgozza a Farkas-lemmát. A lemma jelentőségét csak 1950-ben fedezi fel két amerikai matematikus, Kuhn és Tucker. A felfedezés után a lemma a lineáris optimalizálás alaptételévé válik. 1905-ben Jensen kialakítja a konvexitás fogalmát és bemutatja az első konvex függvényeket. Minkowski 1911-ben mutatja be első eredményeit konvex halmazokon végzett vizsgálatainak. 1917-ben megjelenik az első optimalizálással foglalkozó kiadvány, amelynek címe *Theory of Maxima and Minima*, szerzője Harris Hancock. 1925-ben Morse, elméletének publikálásával általánosítja a variációszámítás területét. A Morse elmélet a modern matematikai fizika egyik legfontosabb tétele. 1928-ban Ramsey a variációszámítást használja az optimális gazdasági növekedési vizsgálataihoz. Munkássága az 1950-es években kerül újra elő, az optimális növekedési elmélet fejlesztése során. 1932-ben Menger általánosan megfogalmazza az utazó ügynök problémáját. 1939-ben Kantorovich publikálja lineáris programozási modelljét és megoldó algoritmusát a problémára. Később, 1975-ben Kantorovich és Koopmans a munkájukért Közgazdasági Nobel-emlékdíjat kapnak.

A II. világháború után az optimalizálás az operációkutatással párhuzamosan fejlődik. Az operációkutatás legnagyobb alakja Neumann, aki 1944-ben Morgensternnel együtt szekvenciális döntési problémákat oldanak meg dinamikus programozás alkalmazásával. 1947-ben az Amerikai Légierőnél dolgozó Dantzig a lineáris programozási feladatokat megoldó Szimplex módszert, ugyanebben az évben alakítja ki Neumann a lineáris programozási problémák dualitás-elméletét. 1949-ben megtartják az első nemzetközi optimalizálásról szóló matematikai konferenciát Chicagoban *International Symposium on Mathematical Programming* címmel. 1951-ben Kuhn és Tucker, John (1948) és Karush (1939) után újra felfedezik nemlineáris problémák optimalitás korlátait. 1954-ben Ford és Fulkerson hálózati problémák körében végzett kutatási nyomán kialakul a kombinatorikus optimalizálás.

Az 1950-es évek második felétől az űrverseny ad újabb lökést az optimalizálás, a szabályozáselmélet megjelenésével pedig főleg az optimális szabályozás elméletének területén. 1956-ban Pontryagin és kutatócsoportja bemutatja a Maximum-elvet. A következő évben Bellman publikálja az Optimum-elvről szóló munkáját. 1960-ban Zoutendijk módszereket mutat be, amikkel a Szimplex módszer általánosítható és alkalmazható nemlineáris problémákon. Ugyanekkor Rosen, Wolfe és Powell is hasonló eredményekről számol be. 1963-ban Wilson elsőként találja fel a szekvenciális kvadratikus programozást. A módszert később Han (1975) és Powell (1977) is sajátjaként mutatja be. 1984-ben Karmarkar lineáris programozási problémákhoz kifejlesztett polinomiális idejű algoritmusa fellendülést hoz a belső pont módszerek használatában. Az 1960-70-es években kialakuló komplexitáselmélet érezhető hatást gyakorol az optimalizálás területén végzett kutatásokon. Az 1980-as évektől elérhetővé váló egyre olcsóbb és hatékonyabb számítógépek a globális optimalizálás és a nagyméretű problémák megoldására tereli a hangsúlyt. Az 1990-es években a belső pont módszereket kiterjesztik a szemidefinit optimalizálás területére.

## Matematikai optimalizálási feladat

A matematikában és a számítástudományban optimalizálási problémának nevezünk egy feladatot, ha a cél egy probléma lehetséges megoldásai közül a legjobbat kiválasztani. Egy optimalizálási feladat a következő formában írható fel:

Adott egy függvény, ami az *A* halmazból a valós számokba képez

Keressük *x0*-t úgy, hogy minimalizálás vagy maximalizálás esetén.

A fenti formulával sok, valós életbeli probléma általánosan modellezhető. A fizika illetve a gépi látás területén a fenti formulát az energiaminimalizálás modellezésére használják, ahol *f* a modellezett rendszer energiája.

Az *A* halmaz jellemzően az Euklidészi tér részhalmaza, ahol *A* elemeinek egy sor egyenlőségi és egyenlőtlenségi feltételnek kell megfelelniük. *A* az *f* függvény értelmezési tartománya, más szavakkal a keresési terület vagy választási halmaz. *A* elemeit lehetséges megoldásoknak nevezzük. Az *f* függvénynek több elnevezése létezik. Általában célfüggvénynek nevezzük, minimalizálás esetén használatos a költségfüggvény, maximalizálás esetén a hasznossági függvény, egyes alkalmazási területeken az energiafüggvény elnevezés. Egy lehetséges megoldást, ami minimalizálja vagy maximalizálja (céltól függően) a célfüggvényt optimális megoldásnak nevezünk.

A matematikában általános elfogadott, hogy minden optimalizálási problémát minimalizálásként kell felírni. Általánosságban, ha a célfüggvény és a megoldási halmaz nem konvex, a minimalizálási problémának több lokális minimuma is létezhet. Egy lokális minimum az az *x\** pont, aminek létezik olyan környezete ahol nem található nálánál kisebb érték. Formálisan:

A lokális maximum a fenti formulához hasonlóan definiálható.

A nagy számban jelen lévő nem-konvex problémákat megoldó algoritmusok nem tudnak különbséget tenni a lokális és globális minimum között, és a probléma megoldásának a helyi minimumot adják. Az alkalmazott matematika és a numerikus analízis globális optimalizálás ágazata foglalkozik olyan determinisztikus algoritmusok kifejlesztésével, amelyek véges időn belül képesek a valós minimális megoldáshoz konvergálni.

## Lineáris optimalizálási feladat

A lineáris optimalizálás a matematikai optimalizálás egy speciális esete. A lineáris optimalizálás módszerével megoldhatók azon optimalizálási feladatok, ahol a célfüggvény lineáris függvény és a korlátozások lineáris egyenlőségek vagy egyenlőtlenségek. A lineáris optimalizálási feladat megoldási halmaza egy konvex politóp, amit véges sok fél-tér határoz meg, melyek mindegyikét lineáris egyenlőtlenségek határoznak meg. A probléma célfüggvénye a poliéderen értelmezett valós értékű affin transzformáció. A lineáris optimalizálási algoritmus a poliéderen keresi azt a pontot, ahol a célfüggvény értéke optimális.

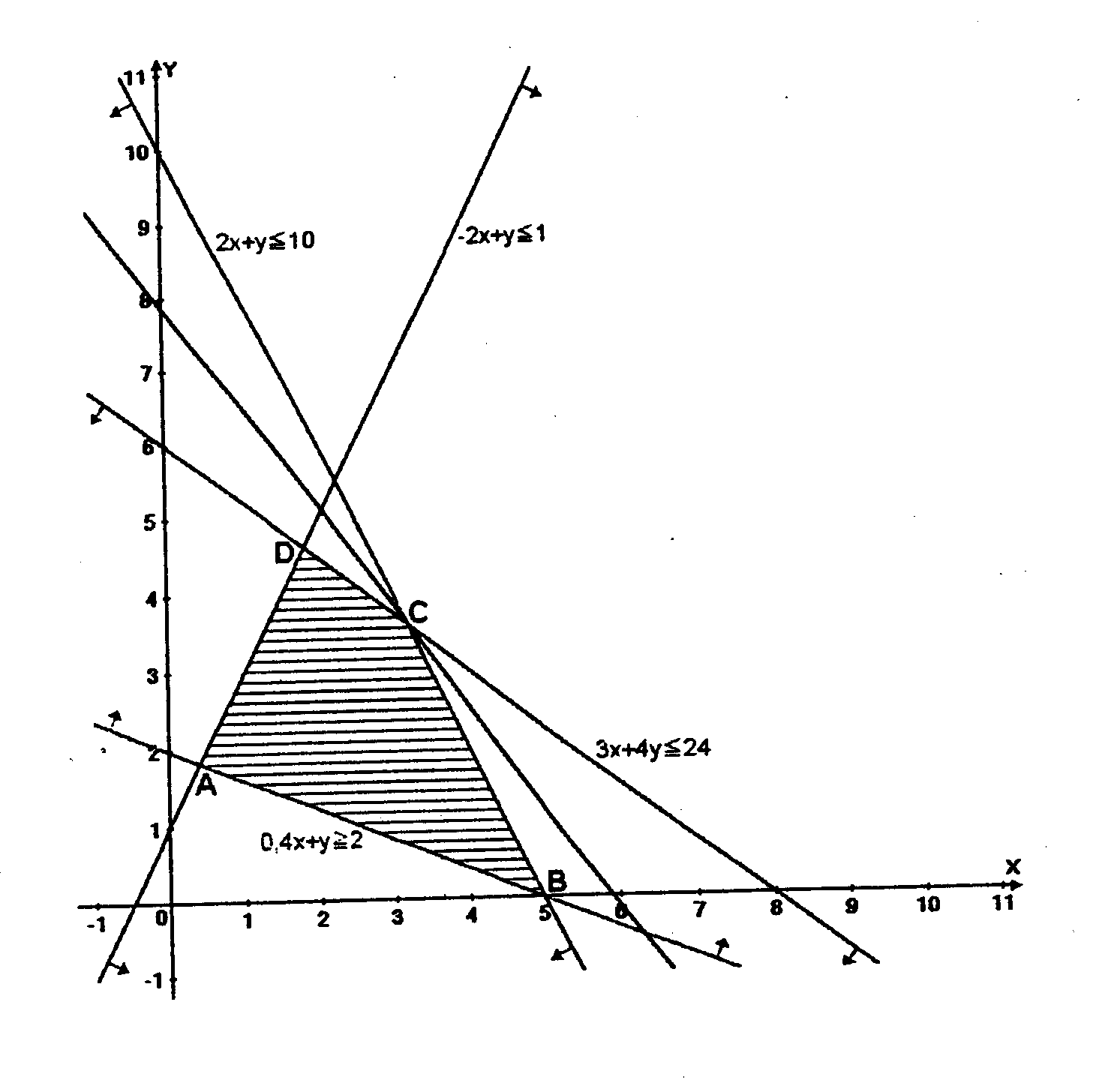
A lineáris programozási feladat általános mátrix alakja a következőképen írható fel:

vagy másként:

ahol , , . A formulában *x* a változók vektora, *c* a célfüggvény együtthatóinak vektora, *b* kapacitásvektor, *A* pedig a technikai együtthatómátrix. Az és korlátozások határozzák meg azt a konvex politópot ami felett az a célfüggvény optimalizálandó.

Az optimális megoldás létezése nem szükséges feltétel. Tekintsük a következő két korlátozást *x*-re: és . Ez esetben a korlátozások által képzett tereknek nincs metszette, az optimalizálási feladat megoldhatatlan. Egy másik példában feltételezzük, hogy a korlátozások által képzett polítóp korlátlan a célfüggvény gradiensének irányában (amely esetben a célfüggvény gradiense megegyezik a célfüggvény együttható vektorával). Ekkor belátható, hogy az optimális megoldás véges időn belül nem elérhető. Azonban a lineáris optimalizálás alaptétele kimondja, hogy ha egy lineáris problémának van optimális megoldása, akkor, az megtalálható a konvex poliéder sarokpontjainak vizsgálatával.

A lineáris optimalizálási feladatok megoldására alapvetően három módszert dolgoztak ki a matematikusok: grafikus megoldás, Szimplex módszer, belsőpontos módszer.



3.1 ábra Lineáris optimalizálási probléma grafikus megoldása. Forrás: []

A grafikus megoldás két változóval leírható problémák megoldására alkalmas. A megoldás során a korlátozásokat egyenlőségekké kell alakítani, és azokat a két dimenziós koordinátarendszerben ábrázolni. Az eredeti egyenlőtlenségek a megfelelő síkterületek megjelölésével (pl. satírozás) ábrázolható. A jelölt síkterületek metszete alkotja a megoldáshalmazt. A megoldáshalmaz pontjai között keresendő az, amelyre a célfüggvény értéke optimális. Lineáris probléma grafikus megoldását szemlélteti a 3.1 ábra.

A Szimplex módszer az optimalizálási feladatot lineáris egyenletrendszerek megoldására vezeti vissza és elsősorban maximalizálási feladatok megoldására alkalmas. Kiindulásként egy olyan táblázatot határoz meg, amelynek bal felső részében a technológiai együtthatók mátrixát, a jobb felső részében a kapacitásvektort, az alsó sora pedig a célfüggvény együtthatóit szerepelteti. Erre példa az alábbi ábra:

3.2 ábra Kiinduló táblázat képzése lineáris egyenletrendszerből Szimplex módszerhez

|  |  |  |  |
| --- | --- | --- | --- |
|  | *x1* | *x2* | *b* |
| *u1* | 1 | 2 | 10 |
| *u2* | 2 | 2 | 12 |
| *u3* | 2 | 0 | 8 |
| *-z* | 2 | 3 | 0 |

A módszer lényege, hogy a technológiai együtthatók mátrixából generáló elemet választunk. A generáló elem választásának szabályai:

1. a kiválasztott elem oszlopában a célfüggvény együtthatója nem lehet negatív, célszerű a legnagyobb elemet választani;
2. csak pozitív szám választható;
3. abból a sorból kell generáló elemet választani, ahol a legkisebb.

A generáló elem kiválasztása után a következő lépéseket kell tenni:

1. a generáló elem sorának elemit osztjuk a generáló elemmel;
2. a generáló elem oszlopának elemeit osztjuk a generáló elem (-1)-szeresével;
3. a generáló elemet a reciprokával helyettesítjük;
4. a táblázat fennmaradó elemeit bázistranszformációs lépésekkel adjuk meg.

Addig kell újabb generáló elemet választani és a fenti lépéseket végrehajtani, amíg a generáló elem választásának szabályai szerint lehetséges annak kiválasztása.

## Nemlineáris programozási feladat

# Ruby on Rails

# Specifikáció

A feladat teljesítéséhez egy webalkalmazás tervezése és implementálása volt a cél. A webes technológia választásának oka, hogy a már megszokott és ismert szálláskereső portálokhoz hasonuljon. Ezen kívül az internetes platformra való fejlesztéssel lehet a legolcsóbban és leggyorsabban a legszélesebb felhasználói kört elérni. A manapság rendelkezésre álló úgynevezett *responsive*, magyarul alkalmazkodó web design-ok alkalmassá tesznek egy weboldalt arra, hogy egyszerre legyen áttekinthető és kezelhető minden képernyőméreten.

## Szereplők

A tervezett rendszerben négy felhasználói szerepkör különül el, amelyek a következők:

1. **Látogató**: bejelentkezés nélkül böngészi a portál publikus tartalmát.
2. **Szálláskereső**: bejelentkezés után szobát keres és foglal
3. **Szállásadó**: bejelentkezés után szobákat hirdet, foglalásokat kezel
4. **Adminisztrátor**: bejelentkezés után a rendszer törzsadatait és beállításait kezeli

## Funkcionális követelmények

A fejezet a webalkalmazással szemben támasztott követelményeket és elvárásokat taglalja.

### Felhasználói fiókok

A rendszernek tudnia kell kezelnie az 5.1-ben meghatározott felhasználói szerepköröket. Minden bejelentkezéshez kötött szerepkörnek tudnia kell regisztrálni, bejelentkezni és kijelentkezni a rendszerből. Adminisztrátort csak adminisztrátor regisztrálhat. A rendszernek tudnia kell szerepkörökként eltérő tartalmat megjeleníteni.

### Szobák szűrése

A látogatónak és a szálláskeresőnek lehetőséget kell biztosítani a szobák szűrésére. A szűrési feltételek között szerepelnie kell a szálláshely szolgáltatásainak, a szoba felszereltségének, a szoba elérhetőségét jelző kezdő- és végdátumnak, a szoba típusát jelző ágyak számának valamint a városnak.

### Szobafoglalás

A szálláskereső csak a kiválasztott időszakban a rendszerben elérhetőként nyilvántartott szobákat foglalhatja le. A foglalás véglegesítése előtt a szálláskeresőnek minden vendég adatát meg kell adnia.

A szobafoglalásról minden szállásadónak egyénileg kell visszajelzést készítenie. A szobafoglalást el lehet fogadni és vissza lehet utasítani. Egy foglalás akkor tekinthető teljesíthetőnek, ha minden szállásadó pozitív visszajelzést küldött. A foglalás nem teljesíthető, ha legalább egy szállásadó negatív visszajelzést küldött.

A szobafoglalások a szálláskereső és a szállásadó részéről is bármikor visszakereshetők és megtekinthetők.

### Értékelés

A teljesült szobafoglalások esetén, az utazás befejező dátumát követően a szálláskereső értékelheti a meglátogatott szálláshelyeket.

### Intelligens keresés

Az intelligens keresés funkció ár és távolság, vagy ezek kombinációja szerint képes automatikus ajánlást készíteni. A választható szempontok mellett figyelembe kell vennie a szálláshelyek értékeléseit és törekednie kell a jobb értékelésűek ajánlására.

### Törzsadatok

Az adminisztrátornak a rendszerben megjelenő törzsadatokat tudnia kell szerkeszteni és bővíteni.

### Tartós címek

A rendszerben megjelenő oldalak címeit és a keresések eredményoldalaira mutató címeket úgy kell kialakítani, hogy azok bármikor újra meglátogathatóak és linkelhetőek legyenek.

## Célcsoport

A webalkalmazás felhasználói célcsoportjaként a szállásadó szerepkör részéről a jellemzően vidéki, alacsony kapacitású panziókat és apartmanokat azonosítottam. Számukra a rendszer ugyanúgy a foglalások egyszerű kezelhetőségét nyújtja, mint a szálláskeresők számára. A szálláskereső szerepkör szempontjából a célcsoport tagjaiként az iskolai kirándulásokat szervező osztályfőnök, a baráti társaságok, illetve az üzleti célból szállást kereső szervezőket tekintem.

# Tervezés

A fejezet a feladat megvalósításához szükséges tervezés eredményét mutatja be. A fejezet kitér az alkalmazásban megjelenő folyamatok tárgyalására, bemutatja az intelligens keresés működéséhez szükséges optimalizálási modelleket. A fejezet második felében a tervezett adatbázis entitásai és a megvalósítás során felhasznált technológiákról lesz szó.

## A rendszerben megjelenő fő folyamatok

Ez a fejezet a rendszerben megjelenő fő interakciós és háttérfolyamatokat mutatja be.

### Szobafoglalás

A szobafoglalás folyamatában a bejelentkezett szálláskereső valamelyik keresési mechanizmust választva feltölti a virtuális kosarát a foglalni kívánt szobákkal. A kosár feltöltése után a szálláskereső véglegesíti a foglalását, megadja a foglalásban részt vevő vendégek adatait és a foglalást elküldi. Az alábbi ábra a folyamat lépéseit részletesen mutatja be.



6.1 ábra Szobafoglalás folyamata

### Foglalás visszaigazolás

A rendszerbe érkező szobafoglalásokat a szállásadóknak külön-külön vissza kell igazolniuk. A foglalás állapota csak akkor változhat meg, ha minden szállásadó megtette visszajelzését. Az alábbi ábra bemutatja a visszaigazolás folyamatát a foglalás szempontjából.



6.2 ábra Foglalás visszaigazolás folyamata

Miután minden szállásadó visszaigazolta a rá vonatkozó szobákat, a rendszer új állapotba lépteti a foglalást. A foglalás teljesíthető állapotúvá válik, ha minden szállásadó pozitív visszajelzést adott. A foglalás nem teljesíthető állapotú lesz, ha legalább egy szállásadó negatív választ adott.

### Intelligens keresés

Az intelligens keresés háttérfolyamatát a rendszer a szálláskereső által megadott keresési feltételek alapján végzi el. A folyamat lépéseit részletezi az alábbi ábra.



6.3 ábra Intelligens keresés háttérfolyamata

A rendszer nem tud közvetlenül kommunikálni a nemlineáris feladatmegoldó eszközzel ezért előbb a szűrési feltételek szerint kiválogatott szobák alapján elkészíti az optimalizáláshoz szükséges adatmodellt és azt, az adatbázisból kiolvasott modellel együtt fájlba írja. Ezután parancssorból végzi a nemlineáris megoldó futását és az eredmények kiolvasását.

## Nemlineáris optimalizálási modell

A nemlineáris optimalizálás során a cél az, hogy az ár, a távolság, illetve a minőség szempontjából optimális megoldást kell találni. A minőség, vagyis a szobák a szálláshelytől örökölt értékelése minden modellben megjelenik, hiszen cél az is, hogy a szálláskereső számára nem csak racionálisan, de emocionálisan is elfogadható megoldást kínáljon a rendszer. Az ár és a távolság választható külön-külön és együttesen is. Tehát három különféle modellt kellett kialakítanom.

A modellek kialakítása során figyelembe kellett vennem, hogy a különböző szempontokhoz különböző nagyságrendű és szórású értékek tartoznak. Az ár jellemzően tízezres nagyságrendű érték. A távolság, amennyiben a keresés egy városra terjed ki a pár tíz kilométernél nem nagyobb, míg város meghatározása nélkül több száz kilométer is lehet. Az értékelés egy 1-től 10-ig terjedő skálán számított átlagos érték. A nemlineáris modellben a célfüggvény a kifejezés minimalizálására törekszik. Ezáltal belátható, hogy a nagyobb nagyságrendű értékektől fog függni a megoldás. Ez nem megfelelő, a megoldás szempontjából minden szempontnak egyenlően kell teljesülnie.

A különböző nagyságrendű értékeket két módszerrel tettem összehasonlíthatóvá. Az első módszerem az, hogy az ár és távolság értékeket nem közvetlenül használom fel az adatmodellben. Az adatmodellbe jegyzés előtt növekvő sorrendbe állítom őket, és minden különböző értéket 1-től egyesével növelve kategóriákba sorolom, ahogy az alábbi ábrákon is látható.



6.4 ábra Árak kategorizálása (Ft)



6.5 ábra Távolságok kategorizálása (km)

Az ár és távolság értékekből annyi kategóriát különböztetek meg, ahány különböző érték megjelenik a kiértékelés során. Mivel ez jellemzően nem haladja meg a 20-30-as számosságot, ezért az értékelésekkel is jobban összevethető. Hangsúlyos előny továbbá, hogy megszűnik a sokaság gyakran előforduló kiugró szórása, ami a következő módszer előnyére is válik.

A fenti módszerrel kialakított kategóriák legnagyobb értéke akár a duplájával is meghaladhatja az értékelések legnagyobb, 10 értékét, azonban a sokaságok szórása közel hasonló értékekkel bír. A célfüggvényben tehát úgy döntöttem, hogy nem a puszta összegeket tekintem, hanem változók által kijelölt ár- és távolságkategóriák, valamint az értékelések sokaságainak speciális relatív szórását. A relatív szórás azért speciális, mert nem a középértékhez közelítem, hanem az ár- és távolságkategóriák esetében a legkisebb, 1 értékhez, míg az értékelések esetében, a legnagyobb 10 értékhez. A relatív szórás eredménye egy százalékos szám. A célfüggvény tehát három százalékérték összegét minimalizálja. Az alábbi képlet az alkalmazott relatív szórási képletet mutatja be, ahol *si* a vizsgált sokaság egy értéke, *xi* a bináris súly, *smin* a vizsgált sokaság lehetséges legkisebb értéke.

6.1 képlet Speciális relatív szórás képlet

A fenti módszerekkel el tudtam érni, hogy több, különböző nagyságrendű sokaságot összehasonlítsak és a célfüggvény kiértékelésekor az algoritmus azokat egyenlőként kezelje.

A 6.1 ábrán bemutatott képlet miatt szükséges, hogy az optimalizálást nemlineáris megoldóval végezze a rendszer. A linearitást a bináris súllyal – ami változóként szerepel a modellben – való szorzással lépi át a modell.

Mindegyik modell bináris változókat használ, amik azt mutatják, hogy mely szobákat kell a megoldáshalmazba beválasztani. A modellekhez alapvetően két adathalmazra van szükség. Az első a szobák adathalmaza, amely minden eleméhez legalább kettő paraméter tartozik: kapacitás és értékelése. A második adathalmaz a vendégek száma. Az optimalizálási szempontok szerint a szobák további paraméterekkel bővülnek. Az alábbi ábra a szobák halmazának egy elemét és a hozzá kapcsolódó változót és paramétereket mutatja be.



6.6 ábra A modellben megjelenő szoba objektum és a hozzá kapcsolódó változó és paraméterek

Mindhárom modellben egy alapvető korlátozást vezettem be, egyértelmű módon azt, hogy a kiválasztott szobák kapacitása egyenlő kell, hogy legyen a vendégek számával.

6.2 képlet Korlátozás a vendégek száma alapján

A fenti képletben *fi* bináris változó, *ci* az *i*-edik szoba kapacitása, *v* pedig a vendégek száma.

A felírt matematikai modelleket nemlineáris bináris egészértékű optimalizálási feladatokra vezettem vissza. A további vizsgálataim során megállapítottam, hogy a célfüggvények kvázikonvex kifejezések.

### Olcsó modell

Az olcsó modell azokat a szobákat adja eredményül, amelyek a legolcsóbbak és a lehető legmagasabb értékeléssel bírnak.



6.7 ábra Az olcsó modellhez szükséges paraméterek

Ahogy azt a fenti ábra is mutatja, ehhez a modellhez a szoba halmaz paraméterlistáját ki kell egészíteni a szobák árával.

6.3 képlet Az olcsó modell célfüggvénye

A modell célfüggvényét a fenti ábra mutatja be, ahol *pi* az i-edik szoba ára, *pmin* a legalacsonyabb szobaár a sokaságban, *ri* az i-edik szoba átlagos értékelése *rmax* pedig a lehetséges legmagasabb értékelés.

### Közeli modell

A közeli modell azokat a szobákat választja ki, amelyek egymáshoz képest a legközelebb helyezkednek el és a lehető legmagasabb értékeléssel bírnak.



6.8 ábra A közeli modellhez szükséges paraméterek

A távolságok tárolásához egy, a szobák halmazán képzett Descartes szorzatból kialakított mátrixra van szükség, ahol a távolság paraméterként jelenik meg, ahogy az a fenti ábrán is látható. A közös szálláshelyen lévő szobák távolsága 0.

6.4 képlet A közeli modell célfüggvénye

A modell célfüggvényét a következő ábra mutatja be, ahol *dij* az i-edik és j-edik szoba távolsága, *dmin* a legalacsonyabb távolság a sokaságban, *ri* az i-edik szoba átlagos értékelése *rmax* pedig a lehetséges legmagasabb értékelés.

### Olcsó és közeli modell

Az olcsó és közeli modell egyesíti a 6.2.1 és 6.2.2 fejezetekben taglalt modelleket, vagyis az egymáshoz legközelebb eső legolcsóbb és lehető legmagasabb értékeléssel bíró szobákat adja eredményül.



6.9 ábra Az olcsó és közeli modellhez szükséges paraméterek

Az összevont modellnek szüksége van minden, az előző két fejezetben tárgyalt kiegészítő paraméterre, ahogy az a fenti ábrán is látható.

6.5 Az olcsó és közeli modell célfüggvénye

Az egyesített célfüggvényt mutatja a fenti ábra, ahol *dij* az i-edik és j-edik szoba távolsága, *dmin* a legalacsonyabb távolság a sokaságban, *pi* az i-edik szoba ára, *pmin* a legalacsonyabb szobaár a sokaságban, *ri* az i-edik szoba átlagos értékelése *rmax* pedig a lehetséges legmagasabb értékelés.

## Adatbázis tervezet

A rendszer működéséhez 18 adatbázistáblát terveztem meg. Ezek közül négy, a *User*, *Admin*, *Guest* és *Owner* táblák a felhasználók adatainak tárolásához és azonosításukhoz szükséges. Az *Admin*, *Guest* és *Owner* táblák és a *User* tábla között polimorfikus kapcsolat áll fenn. Ez azt jelenti, hogy a *User* tábla kapcsolódik a három felhasználói tábla egyikéhez, a *role\_id* attribútumban külső kulcsként tárolva annak elsődleges kulcsát, és a *role\_type* mezőben tárolva a hivatkozott tábla nevét. Tehát egy *User* entitás rendelkezik egy szerepkörrel, amit az *Admin*, *Guest* vagy *Owner* táblákkal képzett kapcsolat azonosít.

Hasonlóan a felhasználói táblákhoz, az *Address* táblához is polimorfikus kapcsolatot terveztem. Címe a szálláskereső (*Guest*) felhasználónak és egy szálláshelynek (*Accommodation*) van. Az *Address* tábla az *addressable\_id* attribútumban tárolja külső kulcsként annak az entitásnak az elsődleges kulcsát, amihez a cím tartozik, és az *addressable\_type* mező azonosítja a hivatkozott tábla nevét.

A 6.10 ábra bemutatja az adatbázis összes tábláját és azok kapcsolatát.

A *Booking* tábla tárolja a szobafoglalásokat. Minden szálláskereső (*Guest*) felhasználó rendelkezik legalább egy *Booking* példánnyal, ami a virtuális kosaraként van dedikálva. A *Booking* tábla *guest\_id* külső kulcsa azonosítja a foglalás tulajdonosát. A *Booking* táblához két kapcsolótáblát terveztem.

A *BookingRoom* kapcsolótábla a foglaláshoz tartozó szobákat tárolja. Mivel a szobák a *Room* táblában nem egyedi példányok, hanem szobatípusok, amelyek a *num\_of\_this* mezőben jelölik számosságukat, ezért minden, a foglalásba beválasztott szobához hozzárendelek egy, a foglalás során egyedi azonosítót, amit az *index* mező tárol. Ez az egyedi azonosító teszi lehetővé, hogy a foglalásban szereplő vendégeket a szobához lehessen csatolni.



6.10 ábra Az adatbázis entitásai és kapcsolatuk

A *BookingGuest* kapcsolótábla a foglalásban szereplő vendégeket, és azok elhelyezését tárolja A *room\_index* mező a *BookingRoom* táblában bejegyzett *index* mezőre hivatkozik, és azt jelenti, hogy a vendég melyik szobában kerül elhelyezésre. Ezen kívül, a *bed* mező jelenti azt, hogy a vendég a szobában melyik ágyon kap helyet. A *bed* mező főként a weboldalon megjelenő űrlapok mezőinek azonosításakor használatos. A foglalás számából, a szobaindexből és az ágy sorszámából képzett azonosító egyedi a teljes rendszerben.

A szálláshelyek szolgáltatásait az *Accommodation* táblához az *AccommodationServiice* kapcsolótáblán keresztül hozzárendelt *Serviice* entitásokkal lehet tárolni. A *Serviice* tábla neve azért tartalmaz két *i* betűt, mert a Ruby on Rails keretrendszer egy védett kulcsszava a *service*. A *category* szintén védett szó, ezért kell a szálláshely kategóriákat tároló táblát *Categry*-nek nevezni. A szobák (*Room*) felszereltségét az *Equipment* tábla tárolja és az *EquipmentRoom* táblán keresztül kapcsolódnak a szobákhoz.

Az értékeléseket a *Comment* tábla tárolja. A *Comment* táblában hivatkozás van a foglalásra az *booking\_id* külső kulccsal, mert egy vendég több foglalás útján újra és újra tehet értékelést. Az értékelés egy szálláshelyről szól, azért szükséges az *accommodation\_id* külső kulcs. A *guest\_id* külső kulcs azonosítja az értékelő vendéget.

A rendszer beállításait és paramétereit a *Property* tábla tárolja. A *Property* táblában az sorok kulcs-érték pároknak tekinthetők. A *group* mező az összetartozó bejegyzéseket jelöli.

## Technológia

Ebben a fejezetben a fejlesztés során felhasznált technológiák kerülnek bemutatásra.

### Keretrendszer

Az 5. fejezetben meghatározott cél, hogy a feladatot egy webalkalmazás képében oldjam meg. Ehhez több programozási nyelv még több keretrendszere közül válogathattam. A tapasztalataim alapján a Ruby on Rails keretrendszert választottam, amit a 4. fejezetben részletesen bemutatok.

A Ruby on Rails keretrendszerben való fejlesztés előfeltétele, hogy telepítve legyen a Ruby nyelvcsomag a számítógépre. A Ruby nyelvcsomag telepítése után telepíteni a kell a *rails* gem-et, aminek segítségével lehetővé válik Ruby on Rails projektek készítése és futtatása.

A Ruby on Rails keretrendszerben való fejlesztéshez szükséges fejlesztői környezet Ubuntu Linux operációs rendszerre való telepítését és konfigurálását mutatja be a [11] melléklet.

### Adatbázis

A webalkalmazás adatbázis megvalósításának a PostgreSQL-t választottam. A PostgreSQL egy több mint 15 éves múltra visszatekintő, nyílt forráskódú megbízhatónak és stabilnak tartott adatbázis motor. Magába foglalja a legtöbb az SQL:2008 szabványban meghatározott adattípust. Szinte minden népszerű programozási nyelvhez létezik kommunikációs interfésze. Kiválóan illeszkedi a Ruby on Rails környezetbe.

A PostgreSQL adatbázis kétszer is elnyerte a *Linux New Media Award For Best Database* díjat, a *Linux Journal* újságírói pedig ötször is neki ítélték az *Editors' Choice Awards for Best Database* díjat.

A PostgreSQL adatbázis használatához a rendelkezni kell egy lokális vagy távoli kiszolgálóval, amihez a Ruby on Rails alkalmazásban a *pg* gem telepítésével lehet kapcsolódni.

### Optimalizálási modellezés

Az AMPL egy modellező eszköz, amivel az optimalizálás teljes életciklusát le lehet fedni. Az AMPL része egy részletes és jól dokumentált modellező nyelv. A modellező nyelv használatával az optimalizálási feladat minden eleme leírható kezdve az adattól, a korlátozásokon át a célfüggvényekig. Emellett a nyelv gazdag programozási lehetőségeket kínál ciklusok és elágazások használatával. Rendelkezik számos beépített matematikai függvénnyel és operátorral. Képes az adatot a modelltől elválasztani és ezáltal paraméterezhető modelleket előállítani. Az AMPL-hez modulárisan illeszthetők különféle, az igények szerint választott lineáris és nemlineáris megoldók. Az AMPL mindhárom (Windows, UNIX, Linux) népszerű operációs rendszert támogatja.

Az AMPL nem egy ingyenes eszköz, de kínál lehetőséget a kipróbálásra. A legegyszerűbben hozzáférhető verzió az AMPL Demo Version, amely nem funkcionalitásban, hanem teljesítményben van korlátozva. Az AMPL Demo Version egy lineáris modellezési feladatnál 500 változót és 500 korlátozást, míg nemlineáris feladat esetén 300 változót és 300 korlátozást képes feldolgozni. A Demo Version-ön kívül létezik egy 30 napos teljes próbaverzió diákoknak. A 30 nap nem hosszabbítható meg és számítógépenként korlátozott. A szakdolgozatomban az AMPL Demo Version-t használtam.

Az AMPL telepítő fájljai letölthetők az AMPL weboldaláról.

### Nemlineáris megoldó

Az AMPL modellező eszközhöz számos lineáris és nemlineáris megoldó is választható. Ezek egy részéért fizetni kell, de vannak nyílt forráskódú, ingyenes megoldók is. A nemlineáris megoldók közül három ingyen letölthető csomagot kínálnak: az Ipopt-ot, a Bonmin-t és a Couenne-t. Mindhárom termék a COIN-OR projekt része, de különböző tulajdonságokkal bírnak.

Az Ipopt csak folyamatos nemlineáris problémákat tud megoldani belső pont módszerrel. Mivel az általam felírt probléma diszkrét bináris változókat használ, ezért nem alkalmazható. Megpróbáltam azonban futtatni, és a megoldó képes optimális megoldást találni, de figyelmen kívül hagyja változók bináris korlátozását.

A Bonmin folytonos és diszkrét változójú konvex problémák globális optimumát szolgáltatja és heurisztikus úton képes nemkonvex problémák megoldására is.

A Couenne megoldó a konvexitástól függetlenül képes megoldani folytonos vagy diszktér változókkal rendelkező nemlineáris problémákat.

Az általam felírt problémához tehát használhattam a Bonmin és Couenne megoldókat, de a fent bemutatott tulajdonságok miatt először a Couenne-t választottam. A kísérletek azonban rácáfoltak az elképzelésekre. A Couenne megoldó, az Ipopt-hoz hasonlóan figyelmen kívül hagyta a változók bináris jellegét és helytelen megoldásokat adott. Ezzel szemben a Bonmin használata jónak bizonyult, bármilyen adathalmazon jó megoldást kínált.

A Bonmin megoldó négy különböző megoldó algoritmust tartalmaz, amelyek a következők:

* B-BB: Nemlineáris programozás (NLP) alapú korlátozás és szétválasztás (branch and bound) algoritmus
* B-OA: Külső közelítéses algoritmus kimondottan vegyes-egész (mixed-integer) nemlineáris problémák megoldására fejlesztve
* B-QG: Quesada és Grossmann korlátozás és vágás (branch and cut) algoritmusa
* B-Hyb: Egy hibrid, külső közelítés alapú korlátozás és vágás algoritmus

Ezek közül kísérleti úton választottam ki a B-OA algoritmust, mert szignifikáns különbséget mutatott futási időben a többi algoritmushoz képest.

A nemlineáris megoldó telepítő fájljai letölthetők az AMPL weboldaláról.

### Megjelenés

A webalkalmazás célcsoportját tekintve fontos, hogy a megjelenő felületek átláthatóak, a szemnek kellemesek és divatosak legyenek. Manapság nem szükséges, hogy egy weboldalhoz a fejlesztője egyedi megjelenésű gombokat, szövegmezőket és egyéb alkotóelemeket tervezzen.

A gyors fejlesztés és a trendek egyszerű követése hívta életre az általános webes megjelenítési csomagokat (angolul: *UI kit*). Ezek olyan ingyenes vagy megvásárolható csomagok, amelyek egy egységes kinézetet biztosító megjelenést ígérnek. A csomagok jellemzően CSS és Javascript fájlokat rejtenek. A CSS fájlokban meghatározott osztályokat az egyszerű HTML elemeken kell alkalmazni. A HTML elemek átmaszkolásán túl a legtöbb csomaghoz tartozhatnak animációk, ikonok, összetett építőelemek (pl.: legördülő menü, lebegő ablak, menüsáv) és komplett, az elrendezést segítő rácsszerkezetek (*grid system*). A legtöbb megjelenítési csomag a rácsszerkezetek használatával képes automatikusan a megjelenést bármilyen méretű képernyőhöz igazítani.

A webalkalmazás elkészítéséhez a Bootstrap nevű megjelenítési csomagot választottam. A Bootstrap a Twitter által kifejlesztett és nyílt forráskódúvá tett megjelenítési csomag. Mivel a Twitter is Ruby on Rails keretrendszert használ, ezért a Bootstrap remekül illeszkedik az én környezetembe is. A Bootstrap a HTML szabvány minden kezelőelemét egyedivé teszi, ezen kívül rendelkezik egy 12 oszloppal operáló rácsszerkezettel, valamint többféle, az összetartozó elemeket egybezáró konténer-elemmel.

A kellemes megjelenés mellett fontos volt, hogy az egyes űrlapelemek és gombok funkcióit ikonok jelezzék a könnyebb érthetőség miatt. A Bootstrap csomag rendelkezik egy korlátozott ikonkészletettel, azonban az általam választott FontAwesome csomag több, mint 500 ikonja gazdagabb megjelenést biztosít.

A fenti csomagokat Ruby gem-ek telepítésével lehet Ruby on Rails alkalmazásban használni. A Bootstrap csomaghoz a *bootstrap-sass* gem-et, a FontAwesome csomaghoz a *font-awesome-sass* gem-et kell telepíteni.

### Autentikáció és autorizáció

Az 5.2.1-ben meghatározott követelmények valamilyen autentikációs és autorizációs modul kialakítását teszik kötelezővé. A Ruby on Rails alkalmazásokhoz több kész megoldás is kínálkozik. Ezek közül én a Devise nevű implementációt választottam.

A Devise egy Warden alapú, a Rails alkalmazások számára készült autentikációs és autorizációs megoldás. A Warden a Rack környezetet használó Ruby alkalmazásoknak autentikációs megoldást. A Devise teljes egészében támogatja a Rails alkalmazások MVC (Model-View-Controller) architektúráját. Képes egyszerre több felhasználót beléptetve tartani és kezelni. Ezen kívül moduláris felépítésű, így konfigurálható az alkalmazásban való használata. A fő szolgáltatásai, hogy a felhasználókat adatbázisból azonosítja, a jelszavakat BCrypt algoritmussal titkosítva tárolja, kezeli az elfelejtett jelszavakat, képes email értesítéseket küldeni, használ email és jelszó validálást, valamint, hogy használatával időben korlátozhatók a munkamenetek.

A Devise használatához a *devise* gem-et kell telepíteni.

### Geolokáció

A szálláskeresők számára előnyös, ha szobák és szálláshelyek böngészése közben a szálláshelyek címeik szerint megjelenítésre kerülnek térképen is. A térképes megjelenítés segítségével a felhasználó könnyebben elhelyezi a szálláshelyet a környezetében.

A szálláshelyek térképen való megjelenítéséhez az első lépés a regisztráció során felvitt teljes cím leképzése koordinátákká. Ehhez egyszerű és kézenfekvő megoldást kínál a Gecoder nevű megoldás. A Geocoder a geokódolásra megjelölt modelleket az adatbázisba mentés előtt megvizsgálja, és a megadott mezők alapján meghatározza a címhez tartozó koordinátákat, amiket a modell *latitude* és *longitude* mezőibe ment. Alapértelmezetten a Google térkép szolgáltatását használja.

A koordináták megjelenítéséhez a Google Maps-et szerettem volna használni népszerűsége miatt. A Google Maps Ruby on Rails alkalmazásokba való egyszerű integrálását ígéri a Gmaps4Rails nevű megoldás. Használatával nem kell az alkalmazáshoz API kulcsot regisztrálni, a legtöbb konfigurációt elrejti, és kényelmes interfészt biztosít a térkép személyre szabására. A térképen megjelenítendő pontokat egy JSON tömbbe foglalva várja.

A fent bemutatott szolgáltatások használatához a *geocoder* és a *gmaps4rails* gem-eket kell telepíteni a Ruby on Rails alkalmazásban.

### Űrlapok

A Ruby on Rails keretrendszer alapértelmezett űrlapsegédjei megkönnyítik a modellekhez kapcsolódó űrlapok elkészítését, azonban még egyszerűbbé tehető a Simple Form nevű megoldás használatával.

A Simple Form egy olyan űrlapsegéd, amely használatával az űrlapok elkészítésekor nem kell foglalkozni a beviteli mezők típusával, mert azt a Simple Form maga választja ki a beviteli mezőhöz tartozó attribútum alapján. Ezen kívül a telepítéskor megadott opció eredményeként a beviteli mezőkbe integrálja a megfelelő Bootstrap osztályokat.

A webalkalmazás rendszeresen visszatérő eleme a dátumválasztó beviteli mező. Legtöbbször a szobák elérhetőségének vizsgálatakor és a foglalás időtartamának beállításához használatos. Egy jól használható, intuitív megoldás a Bootstrap 3 Datepicker nevű kiegészítő. Használatával a beviteli mező alatt vagy fölött megjelenik egy ablak, amiben akár másodpercre pontos időpontok is kiválaszthatók. Ahogy az a nevéből is sejthető, megjelenése jól illeszkedik a Bootstrap stílusához. A Bootstrap 3 Datepicker a MomentJS Javascript könyvtér használja a dátumkezeléshez.

Az intelligens keresés felületén a felhasználónak ki kell választania, hogy milyen szempont szerint (ár, távolság) keressen a rendszer a szobák között. Ehhez az egyszerű jelölőnégyzet helyett valamilyen látványosabb elemet választottam. A Bootstrap Switch nevű megoldás a jelölőnégyzetekből nagy, színes felirattal ellátott kétállású kapcsolókat készít.

A Simple Form használatához a *simple\_form*, a Bootstrap 3 Datepickerhez a *bootstrap3-datetimepicker-rails* és *momentjs-rails*, a Bootstrap Switch kiegészítőhöz pedig a *bootstrap-switch-rails* gem-eket kell telepíteni.

### Képek tárolása és megjelenítése

A szálláskeresők számára előnyös, ha a szobákról és a szálláshelyekről képeket is láthat. A képek növelik a szálláskereső bizalmát és szűrőként is funkcionálnak. A Ruby on Rails alkalmazásokban megjelenő modellekhez a Paperclip nevű megoldással lehet hatékonyan képeket és egyéb fájlokat csatolni.

A Paperclip kiegészítő a fájlok modellekhez való csatolásán kívül elvégzi azok típus szerinti validációját, a képeket képes átméretezni és előnézeti képeket készíteni. Képek tárolásához szükség van az ImageMagick nevű képfeldolgozó könyvtárra.

A Paperclip kiegészítő használatához a *paperclip* gem-et kell telepíteni.

# Megvalósítás

## Adatbázis kapcsolat és modellek elkészítése

A fejlesztés során a fejlesztő gépen egy lokális adatbázist üzemeltettem, ehhez kapcsolódott a készülő webalkalmazás. Ruby on Rails környezetben a konfigurációs feladatok egyike az adatbázis kapcsolat beállítása. A Rails alkalmazások megkülönböztetnek fejlesztési (*development*), tesztelési (*test*) és éles (*production*) környezeteket. Ezek, mint névterek léteznek egy Rails alkalmazásban és hozzájuk konfigurációs beállítások és környezeti változók rendelhetők.

Az adatbázis konfigurációját az alkalmazás gyökérkönyvtárában lévő *config* mappában található *database.yml* fájlban kell elvégezni. Egy tipikus konfiguráció a következőképpen néz ki:

default: *&default*

adapter: postgresql

encoding: utf8

password: m3dw3

username: \*\*\*\*\*

host: localhost

development:

<<: \*default

database: vagato

test:

<<: \*default

database: vagato\_test

production:

adapter: postgresql

encoding: utf8

database: vagato

username: deploy

password: \*\*\*\*\*

host: localhost

A fenti konfigurációból is látszik, hogy az egyező tulajdonságokat nem kell minden környezetben újra definiálni. A YAML típusú konfiguráció megengedi korábban definiált objektumok használatát későbbi bejegyzések tartalmaként.

A fenti konfigurációban egyedül az *encoding*, az adatbázis karakterkódolását beállító bejegyzés nem kötelező, minden más elengedhetetlen a sikeres kapcsolódáshoz és adatmanipulációhoz. Az *adapter* az adatbázis típusa szerinti interfészt azonosítja, a *host* az adatbázis elérési címe. A *username* és *password* mezők a védett adatbázisok bejelentkezési adatait tárolják.

Az adatbázis kapcsolat beállítása után, ha még nem tettük meg létre kell hozni az adatbázis példányt. Ezt megtehetjük közvetlenül a kiszolgálón, a választott adatbázis saját mechanikájával, azonban a Rails környezet nyújt egy egyszerű megoldást. A Rake parancssori eszköz *db* névtere rendelkezik több, adatbázis műveleteket végrehajtó paranccsal. Egy adatbázis konfiguráció érvényesítéséhez a következő parancsot kell a parancssorban végrehajtani:

rake db:create

A Ruby on Rails alkalmazásokban a modellek és az adatbázis megfelelő táblái között nagyon szoros kapcsolat áll fenn. A modelleket, más nyeli megvalósításoktól eltérően, nem kell teljesen deklarálni. Ez azt jelenti, hogy nem szükséges felsorolni az attribútumokat, mert azokat a Rails motor az adatbázis tábláiból azonosítja és az oszlopneveket használja. Egy Ruby on Rails alkalmazásban nem kell az adatelérési műveletek implementációjával foglalkozni, mert Az Active Record nevű szolgáltatás kész megoldást nyújt bármilyen adatmanipulációs művelet elvégzésére. Az egyszerű beillesztések, törlések és frissítések mellett képes bonyolult kapcsolatok és egybeágyazott, összekapcsolt lekérések végrehajtására is. Minden, az *ActiveRecord::Base* osztályból származó modell osztályon végrehajthatók adatbázis műveletek.

Az adatbázis szerkezeti változásait úgynevezett migrációs fájlokban kell bejegyezni. Minden migrációs fájl egy olyan Ruby osztály, amely az *ActiveRecord::Migration* osztályból származik. A fájl nevének tükröznie kell a tartalmában leírt változást. A szobákat tároló *rooms* tábla – kapcsolódó modell: *Room* – létrehozása a következő migrációs kód futtatásával érhető el:

**class** CreateRooms < ActiveRecord::Migration

def change

create\_table :rooms **do** |t|

t.string :name

t.integer :accommodation\_id

t.integer :num\_of\_this

t.integer :capacity

t.text :description

t.timestamps

end

end

end

A migrációs fájlokat el lehet készíteni kézzel, menteni az alkalmazás gyökérkönyvtárában a *db*, azon belül a *migrate* mappába kell. A kézi szerkesztést könnyíti meg a Rails környezet másik népszerű szolgáltatása, a generátorok. Sokféle generátor létezik, ezek közül egy, amelyik modelleket hoz létre. Ahhoz, hogy a fent bemutatott *Room* modellt, és annak *rooms* tábláját létrehozhassuk a következő parancsot kell futtatni a parancssorban:

rails generate model Room name:string description:text capacity:integer num\_of\_this\_integer accommodation\_id:integer

A fenti parancs sikeres futtatásakor a generátor létrehozza a korábban bemutatott migrációs fájlt. Ezen kívül a modellt bejegyzi az *app/models* mappában *Room.rb* névvel és a következő tartalommal:

**class** Room < ActiveRecord::Base

end

A változtatások lejegyzése után frissíteni kell az adatbázist, hogy a azok életbe lépjenek. Ehhez újra a Rake eszközt kell elővenni és annak *db:migrate* parancsát, ahogy az alább látható:

rake db:migrate

A modellek létrehozása után a következő feladat a kapcsolatok kialakítása. A fenti *Room* modell létrehozásánál már sejthető volt az accommodation\_id mező a kapcsolat az Accommodation (szálláshely) modellel. Az Active Record szolgáltatás a táblák id mezője alapján azonosítja a kapcsolatokat és fontos, hogy a hivatkozó mező a {tábla\_név}\_id formátumot kövesse. Az Active Record hatféle kapcsolatot különböztet meg:

1. belongs\_to: egy-egy kapcsolat. A modell az hivatkozott modellhez tartozik.
2. has\_one: egy-egy kapcsolat. Egy *belongs\_to* kapcsolat másik oldala. A modellnek nincs idegen kulcsa a hivatkozó irányába.
3. has\_many: egy-sok kapcsolat. Egy *belongs\_to* kapcsolat másik oldala. A modellnek nincs idegen kulcsa a hivatkozó irányába.
4. has\_many :through: sok-sok kapcsolat. A kapcsolat kapcsolótáblán keresztül valósul meg. A hivatkozott modellekben nincs idegen kulcs, amely a kapcsolótáblára vagy a másik félre mutatna.
5. has\_one :through: egy-sok kapcsolat. Egy másik modellt használ kapcsolótáblaként.
6. has\_and\_belongs\_to\_many: sok-sok kapcsolat. Direkt kapcsolótáblás kapcsolat, ahol a kapcsolótábla nem jelenik meg a modellként.

A 6.10 ábra alapján a Room modell kapcsolatait az alábbi módon kellett bejegyezni:

**class** Room < ActiveRecord::Base

belongs\_to :accommodation

has\_one :price

has\_many :bookings\_rooms

has\_many :bookings, through: :bookings\_rooms

has\_many :bookings\_guests

has\_many :guests, through: :bookings\_guests

has\_and\_belongs\_to\_many :equipments

end

## Autentikáció és autorizáció

## Szobák szűrése

## Intelligens keresés

Az intelligens keresés funkcióhoz a 6.2 fejezetben bemutatott optimalizálási modelleket kellet az AMPL modellező eszköz számára feldolgozható formátummá alakítani. Az optimalizálási modelleket az AMPL modellező nyelvvel lehet deklarálni és hozzájuk adatot csatolni. Az AMPL képes arra, hogy az optimalizálási modelleket és az adatot szétválassza, így azok megadhatók két külön bemenetként.

Az AMPL számára értelmezhető optimalizálási modellek elkészíthetők egy egyszerű szövegszerkesztő segítségével. Az elkészült fájlok kiterjesztése ajánlás szerint *.mod*. A 6.2.3 fejezetben bemutatott, az ár és távolság szerint egyaránt optimalizáló modell az alábbi formában került leírásra:

param min\_distance = **1**;

param min\_price = **1**;

param guests >= **0**;

set **ROOMS**;

param capacity {**ROOMS**} >=**0**;

param stars {**ROOMS**} >=**0**;

param price {**ROOMS**} >= **0**;

param distance {**ROOMS**,**ROOMS**} >= **0**;

var **Occupation** {**ROOMS**} binary;

var occupied\_rooms = sum {i **in** **ROOMS**} **Occupation**[i];

subject to Accommodation:

sum {i **in** **ROOMS**} **Occupation**[i]\*capacity[i] = guests

minimize OPTIMUM:

sqrt(( sum{i **in** **ROOMS**} **Occupation**[i] \* (price[i] - min\_price)^**2**) / occupied\_rooms + **1**) / min\_price +

**if** occupied\_rooms > **1** **then** (

sqrt(( sum{(i,j) **in** {**ROOMS**,**ROOMS**}} (**Occupation**[i] \* **Occupation**[j]) \* ((distance[i,j] - min\_distance)^**2**)) / occupied\_rooms + **1**) / min\_dist

) +

sqrt(( sum{i **in** **ROOMS**} **Occupation**[i] \* ((stars[i] - **10**)^**2**)) / occupied\_rooms + **1**) / **10**;

A modell első felében a paraméterek és változók vannak rögzítve, a második felében a korlátozások és a célfüggvényt.

A *min\_distance* és *min\_price* paraméterek nem változtathatók, mert a 6.2 fejezetben bemutatott kategorizálás után a legkisebb érték mindig 1. A *guests* paraméter a vendégek számát tárolja. Ezt követi a szobák halmazának deklarálása *ROOMS* névvel. A szobák halmazához tartozik a *capacity*, mint kapacitás, a *price* mint ár és a *stars*, mint átlagos értékelés paraméter. A szobák közti távolságot a *distance* paraméter tárolja, amely a szobák halmazából képzett két dimenziós mátrixhoz tartozik. A modell fő változója *Occupation* névre hallgat és a *ROOMS* halmaz felett van értelmezve. Az *occupied\_rooms* egy segédváltozó, a célfüggvény átláthatóbbá tétele, és a rendundancia elkerülése miatt került bevezetésre.

A modellnek egy korlátozása van, mégpedig, hogy a kiválasztott szobák kapacitásának egyenlőnek kell lennie a vendégek számával. Ezt a korlátozást fogalmazza meg az *Accommodation* nevű kifejezés. A modell utolsó eleme az *OPTIMUM* elnevezésű célfüggvény. A 6.2.3 fejezetben bemutatott célfüggvényhez képest az implementáció alkalmaz egy kiegészítést. A nemlineáris megoldó az optimalizálás során választhatja azt, hogy az összes változó 0. Ekkor a gyökvonás operátor hibát jelez, és a folyamat leáll. Ezt elkerülendő, a gyökjel alatti kifejezéshez hozzáadok egyet, ami az eredményt nem befolyásolja, de segítségével a megoldó minden esetben helyesen fut.

Az optimalizálási modell megszerkesztése után össze kell állítani azt az adathalmazt, amelyen az optimalizálást el kell végezni. Amint, azt korábban említettem, az adatokat a modelltől függetlenül, egy másik fájlban is meg lehet adni. Egy ilyen adatfájl kiterjesztése ajánlás szerint *.dat*. Egy, a modellhez illeszthető adatfájl tartalma a következőképpen néz ki:

param guests := **8**;

set ROOMS:= **R1** **R2** **R3** **R4** **R5** **R6**;

param: capacity stars price :=

**R1** **2** **8** **15000**

**R2** **3** **7**.**5** **27000**

**R3** **2** **8**.**2** **16000**

**R4** **2** **8**.**2** **16000**

**R5** **4** **9** **15000**

**R6** **2** **8**.**1** **14000**;

param distance:

**R1** **R2** **R3** **R4** **R5** **R6** :=

**R1** **0** **6** **5** **2** **2** **4**

**R2** **6** **0** **8** **4** **5** **9**

**R3** **5** **8** **0** **6** **4** **7**

**R4** **2** **4** **6** **0** **1** **7**

**R5** **2** **5** **4** **1** **0** **6**

**R6** **4** **9** **7** **7** **6** **0**;

Az adatfájlban a modellben meghatározott paraméterekhez rendelünk értéket, tehát ugyanazokat az elnevezéseket kell alkalmazni. Fontos továbbá, hogy a mátrixok esetében az első index a sornak, a második index az oszlopnak felel meg, illetve egydimenziós halmaz esetén a sor a halmaz eleme és az oszlopok a hozzá tartozó paraméterek.

Ahhoz, hogy az AMPL-lel a fent definiált modell szerint a meghatározott adathalmazt optimalizálni lehessen még szükség van pár sornyi konfigurációra, amit a modell fájlban is megtehettem volna, de az egyértelműség kedvéért egy harmadik fájlban tettem meg. Ez a harmadik fájl a futtatási konfiguráció, kiterjesztése *.solve* és tipikus tartalma a következő:

option solver bonmin;

model vopt.mod;

data vopt.dat;

solve;

option display\_1col **1000000**;

display **Occupation**;

A konfiguráció első sora beállítja, hogy mely megoldóval történjen az optimalizáció. Jelen esetben ez a Bonmin. A második sor a modell fájlt, a harmadik sor pedig az adat fájlt csatolja a feladathoz. A negyedik *solve* parancsának hatására az AMPL levezényli az optimalizálást, annak folyamatát kijelzi a parancssorban. Az utolsó két sor a végső eredmény kijelzését állítja be. Először az kerül meghatározásra, hogy az eredményhalmazt egy oszlopban jelezze maximum egymillió sorig. Végül a *display* parancs paramétereként az AMPL utasítást kap arra, hogy az *Occupation* nevű változóhalmazt írja a képernyőre.

Az optimalizálást a parancssorból lehet elindítani, és az eredményt is ott fogjuk megtekinteni. A művelet elvégzéséhez a következő parancsot kell kiadni:

ampl vopt.solve

A művelet teljes kimenetét hely hiányában megcsonkítottam. A végső eredményt a teljes kimenet végén, az alább látható módon jelzi ki a megoldó. Értelemszerűen az egyesek az elfoglalt szobákat, míg a nullák az optimális eredményből kihagyott szobákat jelzik.

**Occupation** [\*] :=

**R1** **0**

**R2** **0**

**R3** **0**

**R4** **0**

**R5** **1**

**R6** **0**

**R7** **1**

**R8** **1**

**R9** **1**

;

Mivel az AMPL egy önálló, parancssori program és nincs interfésze egy Ruby program számára. A Ruby nyelvben, és így a Rails alkalmazásban is elérhető viszont több lehetőség parancssori műveletek futtatására. Ezek közül kettőt alkalmaztam:

1. system("[command]"): a parancsot egy új környezetben hajtja végre, nem szakítja meg a hívó folyamatot. Visszatérési értéke *true*, ha a futás sikeres, *false* különben.
2. `[command]`: a parancsot szintén egy új környezetben hajtja végre. Visszatérési értéke a művelet képernyőkimenete.

A Ruby programnak az optimalizáló futtatása előtt össze kell gyűjtenie az adatokat és fájlba írnia. Az optimalizálás közvetlen előkészületeit az *OptDataHelper* nevű segédosztály műveletei végzik. Az eddigi példákat követve, az olcsó és közeli optimalizálás elvégzéséhez a következő statikus metódust kell meghívni:

def self.find\_cheap\_and\_close\_solution(rooms, distances,

guests)

problem = generate\_problem\_name

File.open("smartfilter/tasks/#{problem}.dat", 'w') **do**

|data|

write\_rooms\_set(rooms, data)

write\_base\_params(rooms, guests, data)

write\_capacity\_and\_stars\_and\_price\_params(rooms, data)

write\_distance\_params(rooms, distances, data)

end

run\_solver\_on(problem, PROPERTIES['smartfilter - models']['cheap\_and\_close'], rooms)

end

A kódból látható, hogy az alkalmazás gyökérkönyvtárában létrehoztam egy *smartfilter* nevű mappát, amin belül a *tasks* mappa tárolja az optimalizálási feladatok fájljait. A *tasks* mappa mellett található még itt egy *models* nevű mappa, ami biztonsági mentést tárol az optimalizálási modellekről.

A *generate\_problem\_name* metódus egy egyedi azonosítót képez, amivel a feladat azonosítható. Minden, a feladathoz tartozó fájl neve ez lesz, különbség csak a kiterjesztésükben lesz. Az azonosító tartalmazza a végrehajtás dátumát és két 1 és 100 közötti véletlen számot.

A fent bemutatott kód fő feladata, hogy a korábban ismertetett formában fájlba írja az optimalizáláshoz szükséges adatokat. A különböző paraméterek bejegyzését a *write\_capacity\_and\_stars\_and\_price\_params, write\_rooms\_set*, *write\_base\_params* és *write\_distance\_params* metódusok végzik.

Az adatfájl elkészítése után a metódus elindítja az optimalizálást a *run\_solver\_on* metódushívással. A *run\_solver\_on* metódus második paramétere egy konfigurációs fájlból kiolvasott érték, amely egy *Property* entitás *key* mezőjét azonosítja. Ennek oka, hogy az optimalizálási modelleket adatbázisban tárolom, hogy az adminisztrációs felületen egyszerűen lehessen módosítani őket. A *run\_solver\_on* metódust mutatja be az alábbi kód:

def self.run\_solver\_on(problem, model, rooms)

write\_solver\_script(problem, model)

command = "ampl smartfilter/tasks/#{problem}.solve > smartfilter/tasks/#{problem}.solution"

has\_run = system(command)

**if** has\_run

lines = `cat smartfilter/tasks/#{problem}.solution | wc -l`

**if** lines.to\_i <= rooms.size+**3**

puts 'NOT FEASIBLE'

**return** Array.new

**else**

system("cat smartfilter/tasks/#{problem}.solution | tail -n#{rooms.size+4} | head -n#{rooms.size+2} > smartfilter/tasks/#{problem}.raw\_solution")

OptSolutionHelper.read\_solution\_for(problem, rooms)

end

end

end

A fenti metódus első feladata, hogy elkészítse futtatási konfigurációt. Ehhez a *write\_solver\_script* metódust használja, amely a korábban bemutatott futtatási beállítások mellett az adatbázisból elkért optimalizálási modellt is a fájlhoz fűzi. A futtatási konfiguráció elkészülte után összeállításra kerül *command* néven az a parancs, amelyet a parancssorban futtatva elindítható az optimalizálás. A parancs nem csak az optimalizálást indítja el, hanem a művelet minden kimenetét egy *.solve* kiterjesztésű fájlba irányítja. Ebből, a *.solve* kiterjesztésű fájlból kell kiolvasni az eredményt.

A *command* parancsot a *system* metódussal futtatva, a *has\_run* változó felveszi a futás sikerességének értékét. Ezt ellenőrzi a következő elágazás. Mivel a nemlineáris megoldó akkor is képez kimenetet, ha nincs optimális megoldás, ezért ellenőrizni kell a fájlt tartalma szerint. Az első ellenőrzés, hogy a fájl hossza megegyezik-e az optimalizálásba bevont szobák száma plusz 3 sorral. Ha ugyanis a megoldó nem képes a feladatot megoldani, akkor azt egy ennél rövidebb üzenetben közli a felhasználóval. Az eredményfájl sorainak számát megadó parancsot a visszaaposztróf operátorok közé írva a *lines* változó megkapja a parancs eredményét. Ha a sorok száma megfelelő, akkor az optimalizálás eredményét jelző sorokat a fájl végéről levágja a program és egy új, *.raw\_solution* kiterjesztésű fájlba menti. A művelet során nem csak a szobák foglaltságát jelző *Occupation* változók értéke kerül levágásra, hanem az előtte lévő két sor is, ahol ugyanis további információ olvasható ki az eredményről. Miután az eredmény elsődleges feldolgozása megtörtént, a folyamatot az *OptSolutionHelper* segédosztály veszi át a *read\_solution\_for* metódushívás útján.

Az *OptSolutionHelper* segédosztály feladata, hogy az optimalizálás eredményéből összeállítsa az eredményt képező szobák halmazát. A műveletet a *read\_solution\_for* metódus végzi, ezt mutatja be az alább látható kódrészlet. A metódus második paramétere azon szobák halmaza, amelyekből az optimalizálási adathalmazt korábban a program összeállította. A *rooms* halmaz kulcs-érték párokat tartalmaz, ahol az egyes szobákra mutató kulcs a korábban a szobákhoz rendelt, az adatbázisbeli entitás *id* mezője és egy szekvencia alapján képzett egyedi azonosító.

def self.read\_solution\_for(problem, rooms)

File.open("smartfilter/tasks/#{problem}.raw\_solution", 'r') **do** |f|

l\_idx = **1**

f.each\_line **do** |line|

**if** l\_idx == **1**

**return** compact\_solution(rooms) **if** feasibility(line) == 'Full'

**return** Array.new unless feasibility(line) == 'Optimal'

elsif l\_idx > **2**

unless is\_part\_of\_solution?(line)

room\_key = extract\_room\_key\_from(line)

rooms.delete(room\_key)

end

end

l\_idx += **1**

end

end

**return** compact\_solution(rooms)

end

A metódus megnyitja a korábban létrehozott .raw\_solution kiterjesztésű fájlt és sorról sorra beolvassa a tartalmát. A fájl első sorában az optimalizálás eredményessége olvasható ki. Ha a program eljut idáig, akkor az optimalizálás sikeres volt. Két lehetőséget kell megvizsgálni. Az első, hogy az optimalizálás során lett-e kizárva szoba. Ha ugyanis minden szoba az optimális megoldás része, akkor nincs további teendő, a metódus paramétereként kapott szobák halmazát minden további nélkül továbbítani lehet. Ellenkező esetben a kizárt szobákat el kell távolítani a szobák halmazából.

A megoldás fájl harmadik sorától kezdődik az *Occupation* változók felsorolása. A felsorolásban a szobákhoz rendelt azonosító és egy 0 vagy 1 érték szerepel. Azt, hogy az adott sorban szereplő szoba a megoldás rész-e, a *part\_of\_solution?* metódus ellenőrzi. A sorokat *regexp* (Regular Expression) kifejezésekkel, mintaillesztés útján ellenőrzi a program. Ha egy szoba nem része a megoldásnak, akkor szintén *regexp* kifejezéssel kiemeli az azonosítót a sorból, és annak segítségével a *rooms* halmazból eltávolítja a szobát.

Visszatérésként a *rooms* halmaz átalakításra kerül a *compact\_soltuion* nevű metódus által. Az átalakítás során a program eltávolítja a kulcsokat és a szobákat egy listába rendezi.

## Szobafoglalás

# Felületek és használat

## Menüsáv

## Szobák

## Szálláshelyek

## Foglalások

## Intelligens keresés

## Kosár

## Adminisztrációs felületek

# Tesztelés

## Tesztelési környezet

## Teszt adatok

## Teszt eredmények

# Összefoglalás

Irodalomjegyzék

Folyóirat cikk:

NASH, L., SMIDTH, G. (1999). The Alpha-Clustering. *Journal of Computing*. 5(2): 17-29.

Könyv:

KIMT, G. (1998). *A Fuzzy Logic Method In Window Design*. Springer, Berlin Heidelberg New York.

Fejezet (könyvben vagy proceedings-ben):

HINTON, H. (1997). The Heavens are Falling. In: Rosenberg, K. (ed.): *Random Thoughts*. MIT Press, 40-100.

Internetes hivatkozás (CD-n beadnadó)

<http://mik.uni-pannon.hu/index.php?func=news&main=262> MÉSZÁROS P. (letöltés dátuma 2008. október 10.) *Záróvizsga információk 2009. január*

Aláhúzás nélkül kell megadni az internet címeket.

Ábrajegyzék

[3.1 ábra Lineáris optimalizálási probléma grafikus megoldása. Forrás: [] 19](#_Toc416712451)

[3.2 ábra Kiinduló táblázat képzése lineáris egyenletrendszerből Szimplex módszerhez 20](file:///C:\Users\Rozsenich\Documents\THESIS\Documentation\Szakdolgozat_RozsenichBalázs.docx#_Toc416712452)

[6.1 ábra Szobafoglalás folyamata 26](#_Toc416712453)

[6.2 ábra Foglalás visszaigazolás folyamata 27](#_Toc416712454)

[6.3 ábra Intelligens keresés háttérfolyamata 28](#_Toc416712455)

[6.4 ábra Árak kategorizálása (Ft) 29](#_Toc416712456)

[6.5 ábra Távolságok kategorizálása (km) 29](#_Toc416712457)

[6.6 ábra A modellben megjelenő szoba objektum és a hozzá kapcsolódó változó és paraméterek 30](#_Toc416712458)

[6.7 ábra Az olcsó modellhez szükséges paraméterek 31](#_Toc416712459)

[6.8 ábra A közeli modellhez szükséges paraméterek 31](#_Toc416712460)

[6.9 ábra Az olcsó és közeli modellhez szükséges paraméterek 32](#_Toc416712461)

[6.10 ábra Az adatbázis entitásai és kapcsolatuk 34](#_Toc416712462)

[6.1 képlet Speciális relatív szórás képlet 30](#_Toc416712463)

[6.2 képlet Korlátozás a vendégek száma alapján 30](#_Toc416712464)

[6.3 képlet Az olcsó modell célfüggvénye 31](#_Toc416712465)

[6.4 képlet A közeli modell célfüggvénye 32](#_Toc416712466)

[6.5 Az olcsó és közeli modell célfüggvénye 32](#_Toc416712467)

Mellékletek

# Ruby on Rails fejlesztői környezet telepítése és konfigurálása

CD Melléklet

dolgozat (pdf-ben és az eredeti szerkeszthető formában is), internetes hivatkozások letöltött anyagai, összes elkészített saját munka (pl programkód, fénykép stb.)