

SZAKDOLGOZAT

**Kvadratikus hozzárendelési feladat és
billentyűzetkiosztás**

Vincze Roland

Témavezető: Eisenberg-Nagy Marianna

egyetemi adjunktus

BME Matematika Intézet

Differenciálegyenletek Tanszék



**BME
2013**

Tartalomjegyzék

1. Kvadratikus hozzárendelési feladat	6
1.1. Leírás	6
1.2. Alkalmazásai	9
2. Linearizációs módszer	11
2.1. Bevezető	11
2.2. RLT alkalmazása a QAP feladatra	13
3. A korlátozás és szétválasztás módszere	17
3.1. Alapeljárás	18
4. Billentyűzetkiosztás	20
4.1. A modell	20
5. Mohó és linearizációs módszerek	22
5.1. RLT-módszer	23
5.2. Mohó módszerek	24
6. Egészértékű modellek	25
6.1. Leírás	25
6.2. Jelölés	26
6.3. Páros táblaméret	26
6.3.1. 2×2 -es	26
6.3.2. 4×4 -es	27
6.3.3. 6×6 -os	28
6.3.4. 8×8 -as	29
6.4. Páratlan táblaméret	31
6.4.1. 3×3 -as	31
6.4.2. 5×5 -ös	31
6.4.3. 7×7 -es	32

7. Konklúziók, továbblépési lehetőségek	33
7.1. Összegzés	33
7.2. További tervek	34

Bevezető

A billentyűzet egyike a legelterjedtebb és leghatékonyabb eszközöknek, amelyet szöveges adat kézzel való rögzítésére és szerkesztésére használunk. A ma már számítógépekhez is használatos QWERTY-billentyűzet elődje az írógép volt, a jelenlegi kiosztás alig különbözik a több, mint 100 évvel ezelőtti változattól. Ez alatt az idő alatt több más billentyűkiosztás is megjelent a piacon, elsősorban a Dvorak, azonban szignifikánsan hatékonyabbnak ez sem bizonyult [7]. A fent említett billentyűzeteket akár 10 ujjal is használhatjuk, így ezeket többujjas billentyűzeteknek nevezzük. A századforduló óta elterjedtek a különféle hordozható eszközök, okostelefonok és PDA-k, amelyek szintén alkalmasak szöveg rögzítésére. Ezeken az eszközökön is QWERTY-billentyűzet található, azonban a gépelés általában legfeljebb egy ujjal lehetséges.

Egy kiosztás matematikai értelemben akkor optimális, ha összességében a lehető legkevesebb utat kell ujjainknak megtennie gépelés közben. Könnyen látható, hogy ez a fogalom a többujjas billentyűzetekre nehezen megfogható, értelmezhető, és attól is függ, hogy melyik gombot melyik ujjunkkal vagy ujjainkkal használjuk. Egyujjas billentyűzet esetén egyszerűbb a helyzet, itt a cél minimalizálni az írás során bejárt utat egyetlen ujj esetén. A dolgozat témája ez utóbbi billentyűzetfajta optimalizálása a magyar nyelv esetén.

A probléma során a billentyűk négyyszög rácson helyezkednek el, egy rácspontot nevezünk *helynek*. Minden billentyű egyforma méretű, valamint minden helyhez pontosan egy betű, és minden betűhöz pontosan egy hely tartozik. A cél úgy elhelyezni a betűket a lehetséges helyeken a fenti feltételeket figyelembe véve, hogy egy átlagos szöveg begépelésére fordított idő minimális legyen. Hogy megkapjuk ezt az időtartamot, két értéket veszünk figyelembe: hogy a betűk milyen gyakran fordulnak elő egymás után, illetve hogy a betűk által elfoglalt helyek milyen messze vannak egymástól. Ezeket az értékeket a betűk közti gyakoriság- és a helyek közti távolságmátrixok adják meg. A célfüggvényértékhez összeszorozzuk ezt a két értéket, majd összegezzük minden betűpárra.

A probléma hozzárendelési feladat (mert betűkhöz rendelünk helyeket), melyben a célfüggvény kvadratikus. A fenti feltételek miatt a probléma modellezhető az ún. *kvadratikus hozzárendelési feladattal* (*quadratic assignment problem - QAP*). Ez a probléma NP-nehéz, megoldása már kis dimenziókban is sok időt vesz igénybe, $n > 30$ esetén pedig nincs olyan algoritmus, ami elfogadható időn belül megoldást adna.

Az egyujjas billentyűzetkiosztás problémájával a *The single-finger keyboard layout problem* [8] című cikk foglalkozott először. Munkájuk során Dell’Amico és munkatársai főleg heurisztikus módszereket alkalmaztak az optimális kiosztás közélítésére, mint például a *lokális keresés* (*local search*), a *tabu keresés* (*tabu search*), a „*változó szomszédsági keresés*” (*variable neighborhood search*) vagy a fizikában is használt energiaminimalizálási algoritmuson az ún. *cooling scheme*-n alapuló eljárások. Az algoritmusokkal kapcsolatban azonban csak útmutatást tartalmazott a cikk, így azok megírása a vártnál több munkát vett volna igénybe. Részben ez is hozzájárult ahhoz, hogy elsősorban lineáris és egészértékű megoldók segítségével próbáljunk minél jobb kiosztást találni.

Első lépésben a *NEOS (Network-Enabled Optimization System) Server* különböző megoldóival próbáltuk megoldatni a problémát. Az említett weboldal egy ingyenes, internet alapú szolgáltatás, optimalizáló problémák megoldására készült. A billentyűzetkiosztás problémájának modelljét megírtuk **AMPL** és **GAMS** programnyelven is, és teszteltük a különböző *egészértékű* (*integer programming - IP*), *lineáris* (*linear programming - LP*), illetve *globális optimalizálási* (*global optimization - GO*) megoldókat, pl. **BARON**, **MINTO**, **MOSEK** és **XpressMP**. Nagy memóri igényre illetve időtúllépésre hivatkozva egyik megoldó sem adott eredményt. Ekkor döntöttünk a probléma kisebb problémákra történő felbontása mellett. A könnyebb kezelhetőség miatt a Matematika Intézet *Omnibus* szerverén folytattuk a munkát.

A továbbiakban szintén linearizációs, illetve egészértékű programozási módszerekkel dolgoztunk, ezekhez elméleti alapot a 2. és a 3. fejezetekben szolgáltatunk; előbb azonban a 1. fejezetben bemutatjuk a kvadratikus hozzárendelési feladatot, és belátjuk annak NP-teljességét. A 5. fejezetben a linearizációs módszer alkalmazásával foglalkozunk, ebben a **CPLEX** megoldó lesz segítségünkre, majd két mohó módszert alkalmazunk a *Mathematica* szoftver segítségével. Munkánk legnagyobb részét a 6. fejezetben leírtak képezik, ekkor egészértékű programozási feladatként írjuk fel a problémát, itt több kapott billentyűzetkiosztást is bemutatunk. Végül összegezzük az eredményeket, és megemlítünk néhány továbblépési lehetőséget.

1. fejezet

Kvadratikus hozzárendelési feladat

A kvadratikus hozzárendelési feladat (QAP) egyike a legbonyolultabb NP-nehéz problémáknak. Ebben a fejezetben kimondjuk a probléma többféle definícióját, belátjuk nehézségét illetve megemlítjük néhány alkalmazását, ezek közül egyet részletesen is ismertetünk.

1.1. Leírás

A probléma során helyekhez rendelünk hozzá létesítményeket, a létesítmények közt ún. *súlyokat*, a helyek közt ún. *távolságokat* definiálunk. A cél lehet minimalizálás illetve maximalizálás, a már említett súlyok és az egymáshoz rendelt elemek figyelembe vételével. Habár a feltételek egyszerűek, a probléma megoldása nehéz a benne szereplő kvadratikus tagok, valamint a célfüggvény konkavitása miatt.

1.1. Probléma (Formális definíció). *Adott két halmaz, A és B , amelyre $|A| = |B|$, valamint adott az $f : A \times A \rightarrow \mathbb{R}$ súlyfüggvény és a $d : B \times B \rightarrow \mathbb{R}$ távolságfüggvény. Keressük azt a $\varphi : A \rightarrow B$ bijektív függvényt, amelyre a következő összeg minimális:*

$$\sum_{i,k \in A} f(i,k)d(\varphi(i), \varphi(k)).$$

Mivel $|A| = |B|$, és a φ függvény bijektív, vehetjük úgy is, hogy a B halmaz elemeinek sorrendjét rögzítve az A halmaz elemeinek egy sorrendjét keressük, azaz vesszük az A halmaz elemeinek egy permutációját, és az így kapott sorrendben rendeljük hozzá a B halmaz elemeihez. Az irodalomban számos helyen szerepel a probléma felírása a permutációcsoport segítségével, elsőként Hillier és Michael munkája során [12]:

1.2. Probléma (Permutációs megfogalmazás). Legyen S_n az n -elemű permutációk halmaza, és legyen $\pi \in S_n$. Legyen f_{ik} a súly i és k egység közt, és legyen d_{jl} a j és l helyek közti távolság. Ekkor a probléma a következő:

$$\min_{\pi \in S_n} \sum_{i,k=1}^n f_{ik} d_{\pi(i)\pi(k)}.$$

Ekkor minden π permutáció az egységek egyfajta elhelyezését jelenti. A változókat definiálhatjuk logikai (bool típusú) változóknak is, amely értéke 1, ha az $i - j$ párt egymáshoz rendeljük, és 0 egyébként:

$$x_{ij} = \begin{cases} 1, & \text{ha } \pi(i) = j, \\ 0, & \text{ha } \pi(i) \neq j. \end{cases}$$

Ezek a változók egy $X = [x_{ij}]$ permutációmátrixot alkotnak. A modellt a következő formában először Koopmans és Beckmann írták fel [15]:

1.3. Probléma (Egészértékű feladat). Legyen f_{ik} a súly i és k egység közt, és legyen d_{jl} a j és l helyek közti távolság. Ekkor célunk:

$$\min \sum_{i,k=1}^n \sum_{j,l=1}^n f_{ik} d_{jl} x_{ij} x_{kl} \quad (1.1)$$

$$\sum_{i=1}^n x_{ij} = 1 \quad 1 \leq j \leq n \quad (1.2)$$

$$\sum_{j=1}^n x_{ij} = 1 \quad 1 \leq i \leq n \quad (1.3)$$

$$x_{ij} \in \{0, 1\} \quad (1.4)$$

A 1.2. és 1.3. feltétel gondoskodik arról, hogy minden helyhez pontosan egy létesítmény legyen hozzárendelve, illetve minden létesítmény pontosan egyszer legyen elhelyezve.

A következőkben kimondjuk a QAP feladatból származó döntési probléma definícióját, majd megmutatjuk annak NP-teljességét Sahni és Gonzales cikkje alapján [24].

1.1. Definíció (QAP döntési probléma). A döntési probléma bemenete az f_{ik} és d_{jl} nemnegatív egészek, valamint egy B nemnegatív egész. A feladat pedig eldönteni, hogy létezik-e π permutáció, melyre:

$$\sum_{i,k=1}^n f_{ik} d_{\pi(i)\pi(k)} \leq B.$$

1.1. Tétel. *A kvadratikus hozzárendelési feladatból származó eldöntési probléma NP-teljes.*

Bizonyítás:

Ahhoz, hogy belássuk a tétel helyességét, két dolgot kell igazolnunk: azt, hogy a probléma NP-beli, valamint azt, hogy egy NP-teljes probléma visszavezethető rá.

Ha adott egy π permutáció, annak mérete $\mathcal{O}(n)$; valamint $\mathcal{O}(n^2)$ lépésben leellenőrizhető, hogy teljesül-e a (1.1) egyenlőtlenség. Ez azt jelenti, hogy π jó tanú, és polinom idejű algoritmussal ellenőrizhető, hogy π teljesíti-e a feladat feltételeit, tehát a probléma NP-beli.

A bizonyítás második felében visszavezetjük a Hamilton-kör problémát a QAP döntési problémára. Adott $G(V, E)$ gráf esetén legyen $n = m = |V|$, $\omega > 1$, illetve legyenek adottak a következő súlyok:

$$c_{i,j} = \begin{cases} 1, & \text{ha } j = i + 1 \text{ és } i < m \text{ vagy } i = m \text{ és } j = 1 \\ 0, & \text{egyébként} \end{cases}$$

$$d_{k,l} = \begin{cases} 1, & \text{ha } (k, l) \in E \\ \omega, & \text{egyébként} \end{cases}$$

Legyen π a helyeknek létesítményekhez való hozzárendelése, azaz az i indexű létesítményt a $\pi(i)$ indexű helyhez rendeltük. Ekkor a π hozzárendelés $\kappa(\pi)$ költsége $\sum_{i=1}^n c_{i,j} d_{\pi(i), \pi(j)}$, ahol $j = (i \bmod m) + 1$. Ha G -ben van Hamilton kör, akkor π költsége $\kappa(\pi) = m$. Abban az esetben, ha G -ben nincs Hamilton-kör, legalább az egyik $d_{\pi(i), \pi((i \bmod m) + 1)}$ értéke ω , és így a költség nagyobb, vagy egyenlő, mint $m + \omega - 1$. Ekkor:

$$\begin{aligned} \kappa(\pi) &= m, \text{ ha } G\text{-nek van Hamilton-köre} \\ \kappa(\pi) &> m, \text{ ha } G\text{-nek nincs Hamilton-köre} \end{aligned}$$

Tehát az optimális megoldás alapján egyértelműen el tudjuk dönteni, hogy G -ben van-e Hamilton kör. \square

Bonyolultsága miatt a GRIBB projekt (great international branch-and-bound search) a kvadratikus hozzárendelési feladatot első számú feladatává választotta. A projekt egy JAVA-alapú szoftvercsomag elkészítését tűzte ki célul, és az internetre kapcsolódott számítógépek segítségével oldottak meg bizonyos problémákat [22].

Megjegyzés: Ha a 1.1 Problémában megadott A és B halmazokra $|A| < |B|$ teljesül, akkor ún. *SK-QAP* feladatról beszélünk, ezzel a jelöléssel Dell’Amico és munkatársai cikkjében találkoztunk [8].

1.2. Alkalmazásai

Bonyolultsága mellett a QAP számos alkalmazásával is kivívta a kutatók figyelmét világszerte: legelőször Koopmans és Beckmann gazdasági tevékenységek matematikai modellezésére használták [15], de előkerült ütemezési problémáknál (Geoffrion és Graves [10]), a régészetben (Krarup és Pruzen [17]), a statisztikai analízisben (Hubert [13]) és persze az optimális billentyűzetkiosztás problémájánál (Pollatschek, Gershoni és Radday [23]). Leggyakrabban valamilyen új létesítmények elrendezésénél alkalmazták.

Egy esetet részletesebben is leírunk, ez pedig a németországi Klinikum Regensburg építése 1972-ben. Ekkor megbízták Jakob Krarup cégét, a Spadille Inc.-t, hogy tervezzék meg az épület alaprajzát és keressenek egy elrendezést úgy, hogy a részlegek közt a betegeknek összességében minimális utat kelljen megtenniük. A cél az ún. *CDIST* összeg minimalizálása volt, amely során figyelembe vették a részlegek közti transzport (*c*ommunication) és távolság (*d*istance) értékeket. Ez egy QAP feladat, a 1.1 problémabeli jelöléssel az A , illetve B halmazoknak a részlegek, illetve a helyek, az f és d függvényeknek pedig rendre a C és $DIST$ függvények felelnek meg. A probléma két változata *Krarup 30a/b* néven került be az 1991-ben Burkard és munkatársai által megalapított *QAPLIB*-be [5] (quadratic assignment problem library). A *QAPLIB* kvadratikus hozzárendelési feladatokat tartalmaz, és a megjelenő új megoldási módszerek hatékonyságát ezeken a feladatokon vizsgálják.

A Klinikum Regensburg problémájában 30 részleg optimális elhelyezése volt a cél, az épület alakjával kapcsolatban egy feltétel volt adott: a szintek száma legfeljebb 2 lehet. Az 1970-es években a feladat egzakt megoldására nem volt hatékony algoritmus, így véletlen heurisztikákkal próbáltak minél jobb elrendezést kapni (Krarup, 1972 [16]; Krarup és Pruzan, 1978 [18]). Az eljárás során sokkal nagyobb méretű épületet feltételeztek, és véletlenszerűen helyezték el a részlegeket. Ekkor minden részleget sorban áthelyeztek úgy, hogy az elrendezés optimális legyen a „*súlyozott 1-medián*”-elv alapján, azaz a maradék üres helyek közül az új részleg helyét aszerint választották meg, hogy melyik elrendezés adta a legkisebb célfüggvényértéket. Ezt az eljárást 10-szer megismételték (minden futás körülbelül 1 percet vett igénybe), majd kiválasztották a legkisebb célfüggvényértékű elrendezést. A legígéretesebb elrendezés egy két szintből álló 4×4 -es rács volt, ez szolgált a Krarup 30a probléma alapjául; a szintén két szintből álló, 5×3 -as méretű alternatíva pedig a Krarup 30b probléma lett.

A fentihez képest jobb elrendezést adott 1976-ban Burkard és Stratmann [6], majd később Krarup kézi számolások útján javított a megoldáson. Érdekes meg-

jegyeznünk, hogy az utolsó változatból optimális megoldás kapható mindössze két részleg cseréjével, ez azonban akkor még nem volt ismert. A Krarup 30a/b problémákkal azóta is rengetegen foglalkoztak, számos heurisztikus és algoritmikus eljárás született, míg végül 1999-ben a feladatot az ún. *korlátozás és szétválasztás* módszerek (lásd később) segítségével egzakt módon is megoldották [11].

A dolgozatban szereplő billentyűzetkiosztási problémát modellező QAP feladatot még most sem tudjuk egzakt módon megoldani a feladat mérete ($n = 36$) miatt. Munkánk során ezért lineáris és egészértékű programozási módszerekkel próbáljuk közelíteni az optimális értéket. A következő fejezet ehhez a két módszerhez kíván elméleti alapot adni.

2. fejezet

Linearizációs módszer

2.1. Bevezető

Ebben a részben bemutatjuk az *RLT* (*reformulation-linearization technique*, azaz „átalakítás-linearizálás”) módszert, amely segítségével lineáris programozás útján kaphatunk becslést NP-nehéz kombinatorikus optimalizálási feladatok optimális célfüggvényértékére. A módszer segítségével várhatóan nagyobb méretű QAP feladatokat is meg tudunk oldani elfogadható időn belül, vagy legalábbis alsó becslést adhatunk az optimum értékére.

A módszert a 80-as években alkotta meg Adams és Sherali [2]. Az ötlet a következő: adott két lineáris egyenlőtlenség egy $S \subset \mathbb{R}^n$ halmazon, például $v_1^\top x - l_1 \geq 0$ és $v_2^\top x - l_2 \geq 0$ teljesül minden $x \in S$ esetén. Ekkor a bal oldalon álló kifejezések szorzata szintén nemnegatív, azaz átrendezve:

$$(v_1^\top x)(v_2^\top x) - l_2 v_1^\top x - l_1 v_2^\top x \geq -l_1 l_2 \quad \forall x \in S.$$

Az $x_i x_j$ kvadratikus tagok helyett bevezetjük az új X_{ij} változókat, ezzel linearizáljuk a feltételt:

$$\sum_{i,j} v_{1i} v_{2j} X_{ij} - \sum_i (l_2 v_{1i} + l_1 v_{2i}) x_i \geq -l_1 l_2.$$

Ezt a fajta egyenlőtlenséget az x és X változók szerinti *első szintű RLT-vágásnak* nevezzük, több egyenlőtlenség összeszorzásával magasabb szintű vágásokat kapunk. Használjuk az Adams és Sherali [3] által bevezetett jelölést:

$$\left[\prod_{i \in \mathcal{I}} x_i \right]_L = X_{\mathcal{I}},$$

ahol \mathcal{I} egy indexhalmaz, melynek elemei az $\{1, \dots, n\}$ halmazból kerülnek ki, és ahol megengedjük az elemek ismétlődését (pl. $[x_1^2 x_2]_L = X_{\{1,1,2\}}$ vagy X_{112}). Más szavakkal ez az ún. *linearizálási operátor* $[\cdot]_L$ egy monomhoz rendel új változót. Az operátor

kiterjeszthető polinomokra is, ekkor minden egyes monomra külön alkalmazva az új változók összegét kapjuk.

Tekintsük az alábbi kvadratikus optimalizálási problémát lineáris feltételekkel:

2.1. Probléma.

$$\begin{aligned} \min & x^\top Q x \\ & Ax = b \\ & x \geq 0, \end{aligned}$$

ahol $A = (a_{ij}) \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $Q = Q^\top \in \mathbb{R}^{n \times n}$.

A következőkben a 2.1. Probléma első szintű vágásait írjuk fel. Az alábbi két lemma Adams és Sherali [3] munkáján alapul:

2.1. Lemma. *Legyen $\mathcal{F}_0 = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$, $[x_i x_j]_L = X_{ij}$ és a_i^\top az A mátrix i . sora, továbbá*

$$\mathcal{F}_1 = \left\{ (x, X) : x \in \mathcal{F}_0, [(a_i^\top x - b_i) x_j]_L = 0 \ (\forall i, j), X = X^\top \in \mathbb{R}^{n \times n}, X \geq 0 \right\}.$$

Ekkor:

$$[(a_i^\top x - b_i)(a_j^\top x - b_j)]_L = 0 \quad \forall (x, X) \in \mathcal{F}_1,$$

azaz az első szintű RLT relaxáció maga után vonja a fenti feltételeket.

Bizonyítás:

$$\begin{aligned} [(a_i^\top x - b_i)(a_j^\top x - b_j)]_L &= \sum_s [(a_i^\top x - b_i)x_s a_{js}]_L - [b_j(a_i^\top x - b_i)]_L \\ &= \sum_s a_{js} [(a_i^\top x - b_i)x_s]_L - b_j(a_i^\top x - b_i) = 0, \end{aligned}$$

felhasználva, hogy $[(a_i^\top x - b_i)x_s]_L = 0$ minden s , illetve $a_i^\top x - b_i = 0$ minden i esetén. \square

A következő részben azt látjuk be, hogy az eredeti feladathoz *redundáns* lineáris feltételeket hozzávéve az \mathcal{F}_1 halmaz nem változik, tehát ez már az \mathcal{F}_0 halmaz, azaz a 2.1. Probléma megengedett halmazának teljes, első szintű relaxációja.

2.2. Lemma. *Ha $v^\top x - \gamma \geq 0$ minden $x \in \mathcal{F}_0$ esetén, akkor:*

$$\begin{aligned} [(v^\top x - \gamma)x_i]_L &\geq 0 \quad \forall (x, X) \in \mathcal{F}_1, \\ [(v^\top x - \gamma)(a_i^\top x - b_i)]_L &= 0 \quad \forall (x, X) \in \mathcal{F}_1. \end{aligned}$$

Bizonyítás:

Mivel $v^\top x - \gamma \geq 0$ minden $x \in \mathcal{F}_0$ esetén, létezik egy $y \in \mathbb{R}^m$, hogy $A^\top y \leq v$ és $b^\top y \geq \gamma$. Tehát, minden $x \in \mathcal{F}_0$ esetén:

$$\begin{aligned} [(v^\top x - \gamma)x_j] &= \sum_i v_i X_{ij} - \gamma x_j \\ &\geq \sum_i (A^\top y)_i X_{ij} - b^\top y x_j \\ &= [(x^\top A^\top y - b^\top y)x_j]_L \\ &= \sum_k y_k [(a_k^\top x - b_k)x_j]_L = 0. \end{aligned}$$

Ezenkívül

$$[(v^\top x - \gamma)(a_i^\top x - b_i)]_L = \sum_j v_j [x_j (a_i^\top x - b_i)]_L - \gamma (a_i^\top x - b_i) = 0.$$

Az állítást ezzel beláttuk. □

2.2. RLT alkalmazása a QAP feladatra

Az előző fejezetben bemutatott módszert most alkalmazzuk a kvadratikus hozzárendelési feladatra. Ehhez tekintsük a 1.3. Problémában felírt egészértékű megfogalmazást:

$$\begin{aligned} \min \quad & \sum_{i,j,k,l} f_{ik} d_{jl} x_{ij} x_{kl}, \\ \sum_{i=1}^n x_{ij} &= 1, \quad j = 1 \dots n, \\ \sum_{j=1}^n x_{ij} &= 1, \quad i = 1 \dots n, \\ x_{ij} &\in \{0, 1\} \quad i, j = 1 \dots n. \end{aligned}$$

Három lépésben írjuk fel az RLT közelítést:

1. A változókat folytonossá tesszük: $x_{ij}^2 = x_{ij}$ (ez ekvivalens a 1.4 feltétellel.)
2. Beszorozzuk a feltételeket x_{kl} -el, mely megfelel az $x_{kl} \geq 0$ feltétellel való szorzásnak, és linearizálunk az új változók bevezetésével:

$$\begin{aligned} x_{ij} x_{kl} &\rightsquigarrow y_{ijkl} \\ x_{ij}^2 = x_{ij} &\rightsquigarrow y_{ijij} = x_{ij} \\ \sum_{i=1}^n x_{ij} = 1, \quad j = 1 \dots n &\rightsquigarrow \sum_{i=1}^n y_{ijkl} = x_{kl}, \quad j, k, l = 1 \dots n \\ \sum_{j=1}^n x_{ij} = 1, \quad i = 1 \dots n &\rightsquigarrow \sum_{j=1}^n y_{ijkl} = x_{kl}, \quad i, k, l = 1 \dots n \end{aligned}$$

3. Elimináljuk a régi változókat, ezáltal csökkentjük a változók számát:

$$\begin{aligned}
\sum_{i=1}^n x_{ij} = 1, \quad j = 1 \dots n &\rightsquigarrow \sum_{i=1}^n y_{ijij} = 1, \quad j = 1 \dots n \\
\sum_{j=1}^n x_{ij} = 1, \quad i = 1 \dots n &\rightsquigarrow \sum_{j=1}^n y_{ijij} = 1, \quad i = 1 \dots n \\
\sum_{i=1}^n y_{ijkl} = x_{kl}, \quad j, k, l = 1 \dots n &\rightsquigarrow \sum_{i=1}^n y_{ijkl} = y_{klkl}, \quad j, k, l = 1 \dots n \\
\sum_{j=1}^n y_{ijkl} = x_{kl}, \quad i, k, l = 1 \dots n &\rightsquigarrow \sum_{j=1}^n y_{ijkl} = y_{klkl}, \quad i, k, l = 1 \dots n
\end{aligned}$$

Ezek után az egyik lehetőség, hogy az y változóról csak annyit kötünk ki, hogy 0 és 1 közötti valós szám legyen. Ekkor lineáris programozási feladatot (LP) kapunk. A folytonos változók miatt az új feladat megoldáshalmaza nagyobb, mint az eredeti feladaté. A célfüggvényérték tehát legfeljebb akkora lesz, mint a tényleges optimum, ezzel a módszerrel így csak alsó becslést kaphatunk. A folytonos változók miatt pedig nem fogunk egyértelmű választ kapni, hogy melyik helyhez melyik létesítmény tartozik: a megoldó a kapott optimumot általában valós, 0 és 1 közötti y értékek esetén fogja elérni.

Az RLT-1 feladat tehát a következő:

2.2. Probléma (RLT-1 feladat).

$$\min \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{l=1}^n f_{ik} d_{jl} y_{ijkl}$$

$$\sum_{i=1}^n y_{ijkl} = y_{klkl} \quad \forall j, k, l$$

$$\sum_{j=1}^n y_{ijkl} = y_{klkl} \quad \forall i, k, l$$

$$y_{ijkl} \geq 0 \quad \forall i, j, k, l$$

$$y_{ijkl} = y_{klij} \quad \forall i, j, k, l$$

$$\sum_{i=1}^n y_{ijij} = 1 \quad \forall j$$

$$\sum_{i=j}^n y_{ijij} = 1 \quad \forall i$$

Mivel a QAP feladat egy megengedett megoldásában egy A -beli elemhez pontosan egy B -beli elem tartozik, az $x_{ij}x_{il}$ ill. $x_{ij}x_{kj}$ szorzatok nullával egyenlők, így az alábbi feltétel hozzáadható az RLT-1 feladathoz:

$$y_{ijkl} = 0 \quad \forall i = k, j \neq l; \quad \forall i \neq k, j = l$$

A fenti modellből egy újabb szorzással RLT-2 konstruálható, és így tovább, egészen $RLT-n$ -ig, ahol n az eredeti feladat mérete. Az így kapott optimumok monoton növekvő sorozatot alkotnak, és alulról közelítik az eredeti feladat optimumértékét.

Az RLT-2 modell:

2.3. Probléma (RLT-2 feladat).

$$\min \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{l=1}^n f_{ik} d_{jl} z_{ijklkl}$$

$$\begin{aligned} \sum_{i=1}^n z_{ijklkl} &= z_{klklkl} && \forall j, k, l \\ \sum_{j=1}^n z_{ijklkl} &= z_{klklkl} && \forall i, k, l \\ \sum_{i=1}^n z_{ijklpq} &= z_{klklpq} && \forall j, k, l, p, q \\ \sum_{j=1}^n z_{ijklpq} &= z_{klklpq} && \forall i, k, l, p, q \\ z_{ijklpq} &\geq 0 && \forall i, j, k, l, p, q \\ z_{ijklpq} &= z_{klijpq} = z_{klpqij} = z_{pqijkl} = z_{pqklij} = z_{ijpqkl} && \forall i, j, k, l, p, q \\ \sum_{i=1}^n z_{ijijij} &= 1 && \forall j \\ \sum_{j=1}^n z_{ijijij} &= 1 && \forall i \end{aligned}$$

Az előző feladathoz hasonlóan a feltételek a jelen esetben is bővíthetők:

$$\begin{aligned} z_{ijklpq} &= 0 \quad \forall i = k, j \neq l, p, q; \quad \forall j = l, i \neq k, p, q; \quad \forall i = p, j \neq q, p, q; \\ &\quad \forall j = q, i \neq p, k, l; \quad \forall k = p, l \neq q, i, j; \quad \forall l = q, k \neq p, i, j; \end{aligned}$$

Az RLT-1 és RLT-2 feladatokat az Adams és munkatársai által publikált *A level-2 reformulation-linearization technique bound for the quadratic assignment problem* című cikkben leírtak alapján fogalmaztuk meg [1]. A cikkben bemutatták az RLT módszer alkalmazását a QAP feladatra, összehasonlították a módszer és más módszerek által kapott alsó korlátokat, valamint az RLT módszerek számítás- és memóriaigényét. Ezen felül foglalkoztak az RLT-2 ún. *Lagrange-relaxációjával*. Az RLT-2 eljárás során sokkal élesebb becslést kaptak az eredeti feladat célfüggvény-értékére, valamint különböző gyorsító eljárások miatt a futásidő is elfogadható volt. A fentiek miatt szeretnénk foglalkozni a módszerrel munkánk során.

3. fejezet

A korlátozás és szétválasztás módszere

A kvadratikus hozzárendelési feladat felírható egészértékű programozási feladatként is. Ehhez a (2.2), illetve (2.3) feladatokban szereplő y , illetve z változókat binárisnak kell választanunk. Az így kapott feladat megoldása nehéz, de az ún. *korlátozás és szétválasztás (branch-and-bound)* módszerek hatékonynak bizonyultak. Az alábbi leírás Imreh Balázs Kombinatorikus optimalizálás című könyvéből [14] származik.

A korlátozás és szétválasztás módszerének alapgondolata Land és Doig munkájában (1960) fordult elő első ízben [19]. A szerzők a vegyes egészértékű lineáris programozási feladat (mixed integer linear programming - MILP) megoldására adtak meg erre a technikára épülő eljárást. Maga a „branch-and-bound” elnevezés Little és munkatársai (1963) dolgozatában szerepelt először, amelyben egy, az utazóügynök-probléma megoldására szolgáló eljárás került publikálásra [20]. A korábbi speciális eseteket felhasználva, Bertier és Roy (1964) dolgozta ki a módszer első általános változatát [4]. Azóta számos formában és alkalmazásban nyert publikálást a szétválasztás és korlátozás módszere, ezek segítségével sok speciális problémához hatékony megoldási eljárások születtek, köztük a dolgozatban is előkerülő problémafajtára is.

3.1. Alapeljárás

A módszer tárgyalásához tekintsük az alább optimalizálási problémát:

3.1. Probléma.

$$\min\{z(\mathbf{x}) : \mathbf{x} \in L\}$$

ahol L azonos dimenziójú, egész koordinátájú, nemnegatív vektorok véges és nemüres halmaza.

Az adott feltételek mellett a fenti feladatnak nyilván létezik optimális megoldása, hiszen véges sok célfüggvényérték minimumát keressük. Az alapeljáráshoz szükséges két függvény. Az egyik az ún. *szétválasztási függvény*, amely az L halmaz tetszőleges $|L'| > 1$ részhalmazához hozzárendeli L' egy partícionálását (azaz L' két, vagy több részhalmazát, melyeknek diszjunkt uniója visszaadja az L' halmazt). A másik függvény, az ún. *korlátozó függvény*, amely L tetszőleges $L' \neq \emptyset$ részhalmazához hozzárendeli a $z(\bar{\mathbf{x}}) : (\bar{\mathbf{x}} \in L')$ függvényértékek egy alsó korlátját. Speciálisan, $L' = \{\tilde{\mathbf{x}}\}$ esetén a $z(\tilde{\mathbf{x}})$ függvényértéket. Tegyük fel, hogy a rendelkezésünkre állnak az említett függvények, jelölje a szétválasztási függvényt ϕ , a korlátozó függvényt pedig g .

Az eljárás során egy ún. *leszámlálási fát* vagy *branch-and-bound fát* (B & B fa) építünk fel a következők szerint.

Inicializálás: A fa gyökere legyen L . Határozzuk meg $g(L)$ -t és rendeljük címkéként az L csúcshoz. Legyen $r = 1$.

Iterációs rész:

1. Az aktuális fa levelein határozzuk meg a címkék minimumát és válasszunk ki egy minimális címkéjű L' levelet.
2. Ha $L' = \{\bar{\mathbf{x}}\}$, akkor vége az eljárásnak; $\bar{\mathbf{x}}$ optimális megoldás. Ellenkező esetben a 3. lépés következik.
3. Bővítsük az aktuális fát $\phi(L')$ elemeivel, mint L' leszármazottaival, majd az új csúcsokhoz rendre számítsuk ki a korlátokat és rendeljük az illető csúcsokhoz címkéként. Ezek után növeljük r értékét 1-gyel, majd térjünk rá a következő iterációs lépésre.

A dolgozat során a 6. fejezetben előkerülő egészértékű programozási feladatban szereplő változók binárisak, így a megoldó valamilyen branch-and-bound módszer segítségével oldja meg a feladatot. A módszer minden lépés során két partícióra bontja az aktuális halmazt: kiválaszt egy változót, majd a részfa egyik ágában 0-nak, míg a másikban 1-nek rögzíti azt. A megoldónak az ezt követő lépésben két kisebb

IP feladatot kell megoldania. A korlátozó függvény a fa minden egyes csúcsához hozzárendeli a részfeladat LP-relaxációjában (amikor az IP feladatot LP feladatként oldjuk meg, folytonos változókkal) kapott célfüggvényértéket. Ez az érték (ahogy az előző fejezetekben kifejtettük) alsó becslést ad az adott ágon szereplő IP-feladatok célfüggvényértékére. Mindeközben az algoritmus számon tartja az aktuálisan legjobb (egész értékű) megoldást. Amennyiben az egyik ág alsó korlátja magasabb, mint az aktuálisan legjobb megoldás értéke, az említett ágon nincs értelme továbblépnünk, hiszen nem tudnánk az aktuálisan legjobb megoldáson javítani. Tehát ezt az ágot elhagyhatjuk.

A továbbiakban bemutatjuk a billentyűzetkiosztás problémájának számítógépes modelljét, illetve a részproblémák modelljeit. Ezek után az utóbbi fejezetekben tárgyalt módszerek segítségével kíséreljük meg a problémák megoldását.

4. fejezet

Billentyűzetkiosztás

A magyar ábécé a szóközzel együtt 36 betűből (karakterből) áll. Célunk a betűket elhelyezni egy négyzetrácson (táblán). Munkánk során az **AMPL** szoftvercsomagot, azon belül pedig a **CPLEX** megoldót használjuk. Segítségünkre áll a Matematika Intézet *Omnibus2* névre keresztelt szervere, illetve a Differenciálegyenletek Tanszék Operációkutatási csoportjának számítógépe, amelyre *OpKut* néven hivatkozunk majd.

4.1. A modell

Ahogy azt már említettük, a billentyűzetkiosztás problémája modellezhető a QAP feladattal. A számítógépes modellt 1.3. Probléma alapján készítettük, az f , illetve d függvényeknek az új modellben rendre a $freq$, illetve $dist$ mátrixok felelnek meg. Az előzőekben létesítményekként definiált halmazt most a betűk halmaza jelenti.

Halmazok: A modellben definiált halmazok: *Letters* és *Places*. A *Letters* tartalma: a 0 karakter (ami a szóközt jelöli), az angol ábécé betűi, illetve számokkal kiegészített magánhangzók, melyek az ékezetes magánhangzókra felelnek meg, tehát az $a2, e2, i2, o2, o3, o4, u2, u3$ és $u4$ rendre az á, é, í, ó, ö, ő, ú, ü és ű betűket jelölik. A *Places* halmaz pozitív egészeket tartalmaz, m hely esetén 1-től m -ig, amely a lehetséges helyek sorfolytonos számozásának felel meg.

Változók: $x_{ij} \in [0, 1]$, illetve $x_{ij} \in \{0, 1\}$ (rendre a 5.1., illetve a 6. fejezetekben). Jelentése:

$$x_{ij} = \begin{cases} 1, & \text{ha az } i. \text{ betű a } j. \text{ helyen található,} \\ 0, & \text{egyébként.} \end{cases}$$

Paraméterek: Modellünk két paramétermátrixot tartalmaz:

A $freq_{ik}$ megadja, hogy az i . és a k . betű hányszor fordul elő egymás után. E mátrix kiszámolásának legegyszerűbb módja egy ún. webkorpusz feldolgozása: ez arra hivatott, hogy tükrözze egy adott nyelv szóhasználatát, azaz minden benne szereplő szó relatív gyakorisága megközelítőleg ugyanakkora, mint az egész, adott nyelven íródott irodalomban. Használható, magyar nyelvű webkorpuszt azonban nem találtam, így feldolgoztam néhány kortárs könyvet a *Mathematica* segítségével, a könyvek listáját a függelékben mellékelem.

A $dist_{jl}$ megadja, hogy a j . és az l . hely közti mozgáshoz mennyi idő szükséges. Ezt Fitts törvénye [9] adja meg: $\alpha + \beta \cdot \log_2(\frac{D}{A} + 1)$, ahol D a két hely közti távolság, A pedig a billentyűk mérete. Munkánk során a MacKenzie és munkatársai [21] által meghatározott $\alpha = 0$, illetve $\beta = \frac{10}{49}$ értékekkel számolunk. Ezen kívül az egyszerűbb számolás végett feltesszük, hogy a billentyűk egység méretűek, illetve hogy szorosan egymás mellett találhatók. Modellünkben az egyes billentyűket koordinátákkal azonosítjuk, ahol a $([j_1, j_2])$ koordinátákkal rendelkező j . hely sorát j_1 , míg oszlopát j_2 jelöli (pl. 3. sor 5. mezője [3, 5]), távolságként pedig az euklideszi távolságot használjuk:

$$dist_{jl} = \frac{10}{49} \cdot \log_2 \left(\sqrt{(l_1 - j_1)^2 + (l_2 - j_2)^2} + 1 \right)$$

Célfüggvény:

$$\sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^n \sum_{l=1}^m freq_{ik} dist_{jl} x_{ij} x_{kl} \rightarrow \min$$

Feltételek:

QAP esetén a betűk, és a helyek száma is n :

$$\begin{aligned} \sum_{i=1}^n x_{ij} &= 1 & 1 \leq j \leq n \\ \sum_{j=1}^n x_{ij} &= 1 & 1 \leq i \leq n \end{aligned}$$

SK-QAP esetén n db betűt rendelünk hozzá m darab helyhez és $n < m$:

$$\begin{aligned} \sum_{i=1}^n x_{ij} &\leq 1 & 1 \leq j \leq m \\ \sum_{j=1}^m x_{ij} &= 1 & 1 \leq i \leq n \end{aligned}$$

A következő fejezetekben többféle módszerrel kísérjük meg a billentyűzetkiosztás problémájának megoldását. A linearizációs és egészértékű módszerek során a fent ismertetett modellt használjuk majd, előbb azonban a betűk kétféle sorrendjét, illetve két mohó algoritmust mutatunk be.

5. fejezet

Mohó és linearizációs módszerek

Mivel az összköltség minimalizálása a cél, ezért magától értetődik, hogy a leggyakrabban előforduló betűket igyekszünk közel elhelyezni egymáshoz, azonban nem szabad megfeledkeznünk az egymás utáni előfordulásukról sem. Így a betűk kétféle sorrendjével dolgozunk:

#	Gyakoriság szerint		Iteratív sorrend szerint	#	Gyakoriság szerint		Iteratív sorrend szerint
	Betű	Gyakoriság			Betű	Gyakoriság	
1.	∅	4005181	∅	19.	v	425324	v
2.	e	2308084	a	20.	h	398679	h
3.	a	2081639	t	21.	b	396212	b
4.	t	1705982	e	22.	ö	254974	ö
5.	n	1374003	n	23.	j	252650	j
6.	l	1311425	l	24.	u	238135	u
7.	s	1211496	k	25.	f	216212	f
8.	k	1057392	m	26.	p	203674	p
9.	o	977353	i	27.	ó	192431	ó
10.	i	945420	s	28.	c	185838	c
11.	m	915636	z	29.	ő	181077	ő
12.	r	881734	o	30.	ü	127649	ü
13.	z	850116	r	31.	í	105580	í
14.	g	811944	é	32.	ú	78169	ú
15.	á	723243	g	33.	ű	34734	ű
16.	é	700051	á	34.	w	18765	w
17.	y	527086	y	35.	x	2899	x
18.	d	477578	d	36.	q	325	q

A fenti táblázat első részében csökkenő sorrendben haladunk az abszolút gyakoriság szerint, azaz az első a magyar szövegekben leggyakrabban előforduló karakter (ez a szóköz, melyet a \emptyset jel jelöl), majd a második leggyakoribb, és így tovább. A

második sorrendet iteratív módon határozzuk meg: a sorrend szerinti k . betű az, amelyik az első $k - 1$ betű mellett a leggyakrabban fordul elő a még megmaradt betűk közül. A későbbiekben a betűk mindkét sorrendjével foglalkozunk majd.

5.1. RLT-módszer

A feladat AMPL szoftverrel történő megoldását a 2. fejezetben ismertetett linearizációs módszerekkel kezdtük. Az RLT-1, illetve RLT-2 közelítések azonban nem adtak eredményt az eredeti feladatra elfogadható időn belül, így a módszereket kipróbáltuk kisebb méretű feladatokon.

Elsőként 2×2 -es táblát vettünk és 4 betűt, erre mindkét módszer 0-1-mátrixszal tért vissza, így egyértelmű, és egymással (forgatás és tükrözés erejéig) megegyező megoldást adott. Ezután megpróbáltunk egy 3×3 -as táblát, illetve 9 betűt tartalmazó modellt megoldani, az RLT-1 módszer által kapott eredményből csak az volt egyértelmű, hogy a szóköz a tábla közepére kerül, a többi betű 0.25 vagy 0.5 súllyal szerepelt több helyen is. Míg az eddig említett problémák futásideje kevesebb, mint 1 másodperc volt mindkét gépen, a megoldó a 3×3 -as problémára az RLT-2 módszer esetén 2 nap alatt sem adott megoldást. Vegyük észre, hogy míg az eredeti feladatban 9^2 változónk volt, hiszen 9 betű-hely párról kellett eldönteni, hogy igaz-e, vagy sem; míg az RLT-2 relaxáció esetén a modellben $9^2 \cdot 9^2 \cdot 9^2$ db, azaz több, mint félmillió változó szerepelt. Ezen felül a feltételek nagy számával is magyarázható a nagy futásidő.

Végül az RLT-1-ben kapott eredmény alapján a szóközt lerögzítve már 18 (Omnibus2), illetve 2 (OpKut) óra múlva kaptunk egy 0-1-mátrixot, itt megjegyeznénk, hogy a mátrix a 6. fejezetben kapott mátrixszal megegyezett. Érdekes megjegyeznünk azt is, hogy az RLT-1 közelítéssel is megoldottunk egy modellt a szóközt lerögzítve, de a megoldásként kapott mátrix továbbra sem szolgáltatott hasznos információval a többi betű helyét tekintve. Utolsóként egy 4×4 -es táblát vettünk az RLT-1 módszerrel, ez 1 percen belül lefutott, a megoldásmátrix azonban szinte semmilyen információt nem adott. Az RLT-2 módszerrel való kísérletezést előző tapasztalatok alapján a túl magas futásidő miatt elvetettük.

5.2. Mohó módszerek

A mohó módszer abból áll, hogy elhelyezzük az első betűt a tábla közepére, majd a betűket egyesével lerakjuk úgy, hogy minden egyes lerakásnál a lehető legkisebb költséggel járó kiosztás mellett döntünk.

A következő 2 kiosztást a *Mathematica* segítségével kaptuk, a programkódot a függelékben mellékeljük.

				q				
			ő	u	ű			
		j	d	m	b	ó		
	i	á	t	e	l	y	ú	
	p	r	a	Ø	n	g	ü	
		é	k	s	o	h		
		ö	z	i	v	f		
			x	c	w			

Költség: 4 070 919

Gyakoriság szerinti sorrend

		q	c	p	w			
		u	é	s	z	ú		
	í	g	m	a	i	j		
	h	o	t	Ø	n	d	x	
	ű	y	l	e	k	ö		
		b	á	r	v	f		
			ő	ó	ü			

Költség: 4 082 935

Iteratív sorrend

5.1. ábra. A mohó módszerek által kapott kiosztások

Az első kiosztás „jobb”, mert kisebb a költsége, mint a másodiknak. Ez azzal magyarázható, hogy míg az első kiosztás készítésekor inkább globális szempontokat (abszolút gyakoriság), a második kiosztás készítése során inkább lokális szempontokat (az aktuális betűk melletti előfordulás) vettünk figyelembe.

6. fejezet

Egészértékű modellek

Az előző fejezetben a billentyűzetkiosztási problémát felírtuk lineáris programozási feladatként, ez azonban nem hozott érdemi eredményt, a mohó módszerek pedig szintén nem biztosítanak optimális megoldást. A következő lépés az egészértékű programozási feladatként való felírás, és a feladat megoldása a CPLEX megoldóval. Egzakt megoldása természetesen az IP feladatnak sem lehetséges a probléma nehézsége miatt, azonban kisebb méretű feladatok esetén ($n \leq 20$) még elfogadható időn belül kapunk megoldást. Ebben a fejezetben felírjuk az IP modellt általánosan, illetve hogy milyen lépéseken át jutunk el a végső megoldásokig. Végül részletesen bemutatjuk az IP feladatok megoldásával nyert billentyűzetkiosztásokat.

6.1. Leírás

Az IP modellek megegyeznek a (2.2) RLT-1 modellekkel, azzal a különbséggel, hogy a folytonos y változók helyett bináris y változókat feltételezünk.

A különböző méretű feladatok közti különbség most is csak a két halmaz elemeiben nyilvánul meg (a *Letters* halmaz más betűket tartalmaz, illetve a *Places* halmaz több vagy kevesebb pozitív egészet). A halmazok elemeit az egyes alfejezetekben feltüntetjük, valamint az AMPL modellt a Függelékben mellékeljük.

Ami a tábla méretét illeti, elindulhatunk 2×2 -es, illetve 3×3 -as rácsból, majd mindig egy újabb „keretet” hozzávéve helyezünk el egyre több betűt. Munkánk ezen részében tehát a betűk kétféle sorrendjét használva fogjuk elhelyezni a betűket páros (2×2 , 4×4 , 6×6 és 8×8 méretű) és páratlan (3×3 , 5×5 és 7×7 méretű) táblákon.

6.1. Probléma (Egészértékű feladat).

$$\min \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^n \sum_{l=1}^m freq_{ik} dist_{jl} y_{ijkl}$$

$$\begin{aligned}
\sum_{i=1}^n y_{ijkl} &= y_{klkl} & \forall j, k, l \\
\sum_{j=1}^m y_{ijkl} &= y_{klkl} & \forall i, k, l \\
y_{ijkl} &= y_{klij} & \forall i, j, k, l \\
y_{ijkl} &= 0 & \forall i = k, j \neq l; \\
& & \forall i \neq k, j = l \\
y_{ijkl} &\in \{0, 1\} & \forall i, j, k, l \\
\sum_{i=1}^n y_{ijij} &= 1 & \forall j \\
\sum_{i=j}^m y_{ijij} &= 1 & \forall i
\end{aligned}$$

6.2. Jelölés

A feladatokat és megoldásaikat a következő módon azonosítjuk: $[l, p, f]$, ahol l az elhelyezendő betűk száma, p a helyek száma és f a rögzített betűk száma. Mivel munkánk során kizárólag négyzet alakú táblákkal foglalkozunk, ez a jelölés egyértelmű (pl. $p = 25$ esetén tudjuk, hogy 5×5 -ös tábláról van szó). Amennyiben egy feladatra több megoldást is kapunk, azt a következő módon jelöljük: $[l, p, f]_i$ (itt $i = 1 \dots$).

6.3. Páros táblaméret

6.3.1. 2×2 -es

Először az első 4 betűt helyezzük el egy 2×2 -es táblán, ezek mindkét sorrend szerint a következők: $\{\emptyset, a, e, t\}$. A helyek: 1..4. Az alábbi 3 kiosztás lehetséges, a többi belőlük forgatással és tükrözéssel megkapható:

a	\emptyset
t	e

a	\emptyset
e	t

t	\emptyset
a	e

Költség: 379 101

$[4, 4, 0]_1$

Költség: 380 635

$[4, 4, 0]_2$

Költség: 405 798

$[4, 4, 0]_3$

6.1. ábra. 2×2 -es tábla 4 betűvel

6.3.2. 4×4-es

Az, hogy az első tábla esetében lokális optimumot kaptunk, nem feltétlenül jelenti azt, hogy a globális optimumot is az első táblát folytatva fogjuk elérni. Ezért mindhárom esettel foglalkozunk számításaink során. A második lépésben sorra kerülő betűk szintén megegyeznek a két sorrendben: tehát az $\{\emptyset, a, e, t, \acute{a}, \acute{e}, g, i, k, l, m, n, o, r, s, z\}$ betűket helyezzük el egy 4×4-es rácson. A rács közepébe már lerögzítettük az első 4 betűt, a rögzített betűket most és a továbbiakban is vastag kerettel jelöljük. Ezt a bináris IP feladatot az AMPL segítségével oldottuk meg, és a következő kiosztásokat kaptuk:

z	s	k	i
é	a	∅	m
r	t	e	n
á	l	o	g

Költség: 2 521 532

$[16, 16, 4]_1$

é	n	k	i
g	a	∅	m
l	e	t	s
á	r	o	z

Költség: 2 538 765

$[16, 16, 4]_2$

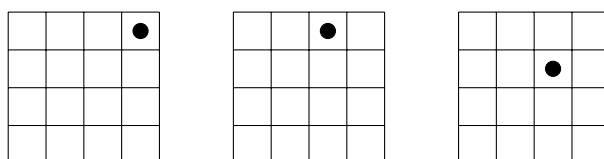
é	s	k	i
z	t	∅	m
o	a	e	n
g	r	l	á

Költség: 2 556 751

$[16, 16, 4]_3$

6.2. ábra. 4×4-es tábla 16 betűvel, ebből 4 rögzítve

Felmerül a kérdés, hogy kaphatunk-e jobb elhelyezést, ha nem rögzítjük a belső négy betűt. Kísérleteink azonban azt mutatták, hogy amennyiben nem rögzítünk le egy betűt sem, a megoldó nem ad eredményt elfogadható időn belül; egy betű rögzítése esetén azonban még igen. Ha kizárólag egy betű helyzetét tekintjük, a szimmetria miatt csak 3 lehetőség van: a „sarokban”, a tábla „oldalán” és a „közepén”:



6.3. ábra. A szóköz helye a 4×4-es táblán

Tehát, ha lerögzítjük a szóközt ezekre a helyekre, és megoldjuk mindhárom feladatot, a három megoldás közül a legkisebb célfüggvényértékű lesz az optimális megoldás a $[16, 16, 0]$ problémára (6.4. ábra).

A 6.2, illetve 6.4 ábrákat tekintve látszik, hogy a $[16, 16, 0]$ probléma optimális megoldása a $[16, 16, 4]_1 = [16, 16, 1]_3$ kiosztás.

i	k	a	∅
s	n	t	m
z	o	e	l
é	g	r	á

Költség: 2 685 321
 $[16, 16, 1]_1$

z	a	∅	k
s	t	n	i
é	l	e	m
á	r	o	g

Költség: 2 590 815
 $[16, 16, 1]_2$

z	s	k	i
é	a	∅	m
r	t	e	n
á	l	o	g

Költség: 2 521 532
 $[16, 16, 1]_3$

6.4. ábra. 4×4-es tábla 16 betűvel, 1 rögzítve

6.3.3. 6×6-os

Most a maradék betűket is elhelyezzük egy 6×6-os táblán, azaz a betűk halmaza tartalmazza az összes betűt, a helyek halmaza pedig a helyeket 1-től 36-ig. Az alábbi 5 kiosztást rendre a $[16, 16, 4]_1$, $[16, 16, 4]_2$, $[16, 16, 4]_3$, $[16, 16, 1]_1$ és $[16, 16, 1]_2$ kiosztások felhasználásával kaptuk:

w	í	c	v	ö	x
ú	z	s	k	i	f
p	é	a	∅	m	u
j	r	t	e	n	d
ó	á	l	o	g	y
q	ő	b	h	ü	ű

Költség: 4 005 205
 $[36, 36, 16]_1$

x	u	d	v	ü	ű
p	é	n	k	i	f
y	g	a	∅	m	ö
j	l	e	t	s	c
ó	á	r	o	z	í
q	ő	b	h	ú	w

Költség: 4 030 767
 $[36, 36, 16]_2$

q	w	c	ö	f	x
í	é	s	k	i	u
p	z	t	∅	m	v
h	o	a	e	n	d
ú	g	r	l	á	j
ű	y	ó	b	ő	ü

Költség: 4 051 945.
 $[36, 36, 16]_3$

w	u	d	v	h	ő
c	i	k	a	∅	f
ö	s	n	t	m	b
p	z	o	e	l	ó
í	é	g	r	á	j
q	ű	y	ü	ú	x

Költség: 4 161 936
 $[36, 36, 16]_4$

w	ő	v	h	f	ű
í	z	a	∅	k	ö
c	s	t	n	i	b
p	é	l	e	m	y
ü	á	r	o	g	ú
q	ó	j	d	u	x

Költség: 4 081 836
 $[36, 36, 16]_5$

6.5. ábra. 6×6-os tábla 36 betűvel, 16 rögzítve

6.3.4. 8×8-as

Az ötlet a következő: az előzőtől eltérően nem egy, hanem két mező vastagságú keretet adunk meg, így a megoldónak van lehetősége körhöz hasonlító formában elhelyezni a maradék betűket. A probléma jelölése tehát [36, 64, 16]. Mivel a helyek száma több, mint a betűk száma, ezt a modellt az SK-QAP modellek közé soroljuk. A célfüggvényérték ebben az esetben nyilván legfeljebb akkora lehet, mint a [36, 36, 16] megoldásainál, de abban bízunk, hogy szigorúan kisebb lesz.

A megoldó azonban a feladatot az eredeti formában nem tudta megoldani a túl nagy memóriaigényre hivatkozva, ennek oka a változók nagy száma volt. Bár a legtöbb változó értékét rögzítettük, amikor a középső 16 betűt a helyekhez kötöttük, de a memóriaigény ettől nem csökkent. Ezért ahelyett, hogy a modellben plusz feltételekkel rögzítjük a már elhelyezett 16 betűt a középső 4×4-es táblán, elimináljuk az ezen betűkhöz és helyekhez tartozó változókat a feladatból. Ez azt jelenti, hogy 64 helyett 48 hely marad, illetve 36 helyett 20 betű, valamint a célfüggvény együtthatóját is megfelelően módosítanunk kell. A maradék betűk és helyek közt a következő súlymátrixot definiáljuk:

$$wght_{ij} = \sum_{k \in Fixed} freq_{ik} \cdot dist_{j\varphi(k)}$$

A fenti képletben a *Fixed* halmaz a már rögzített betűk indexét tartalmazza, a φ függvény pedig azok helyét adja meg. Az i , illetve j indexek végigfutnak a maradék betűk, illetve a szabad helyek halmazán. Ezt a mátrixot a *Mathematica* segítségével számoltuk ki. Az AMPL modell célfüggvénye a következőre változik:

$$\sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^n \sum_{l=1}^m freq_{ik} dist_{jl} y_{ijkl} + 2 \cdot \sum_{i=1}^n \sum_{j=1}^m wght_{ij} y_{ijij} + C \rightarrow \min$$

A fenti kifejezésben az első tag az új betűk egymás közti kapcsolatát, a második tag az új-régi betűk kapcsolatát jelenti. A C érték az előző lépésben megkapott kiosztás költsége, ez a régi betűk kapcsolatát jelenti. A programkódot és az átírt modellt a függelékben mellékeljük.

Az előzőekben leírtak segítségével elért méretcsökkenés már elegendő volt ahhoz, hogy a feladatot a megoldó kezelni tudja. A kapott kiosztások:

			w				
		í	c	v	ö	x	
	ú	z	s	k	i	f	
	p	é	a	∅	m	u	q
	j	r	t	e	n	d	ű
	ó	á	l	o	g	y	
		ő	b	h	ü		

Költség: 4 004 786

$[36, 64, 16]_1$

				x			
	q	u	d	v	ü	ű	
	p	é	n	k	i	f	
	y	g	a	∅	m	ö	
	j	l	e	t	s	c	w
	ó	á	r	o	z	í	
		ő	b	h	ú		

Költség: 4 030 625.

$[36, 64, 16]_2$

		w	c	ö	f	x	
	í	é	s	k	i	u	q
	p	z	t	∅	m	v	ű
	h	o	a	e	n	d	
	ú	g	r	l	á	j	
		y	ó	b	ő	ü	

Költség: 4 051 435

$[36, 64, 16]_3$

6.6. ábra. 8×8 -os tábla 36 betűvel, 16 rögzítve ($[16, 16, 4]$ táblák alapján)

			x	ű			
		p	ö	v	h	f	
	w	i	k	a	∅	b	
	c	s	n	t	m	d	ő
	í	z	o	e	l	u	ó
		é	g	r	á	j	q
			y	ü	ú		

Költség: 4 159 519

$[36, 64, 16]_4$

				ű			
	w	ő	v	h	f	x	
	í	z	a	∅	k	ö	
	c	s	t	n	i	b	
	p	é	l	e	m	y	
	ü	á	r	o	g	ú	
		ó	j	d	u	q	

Költség: 4 081 600

$[36, 64, 16]_5$

6.7. ábra. 8×8 -os tábla 36 betűvel, 16 rögzítve ($[16, 16, 1]$ táblák alapján)

Sejtésünk beigazolódott, valóban javítottunk valamennyit a költségeken. Itt jegyeznénk meg, hogy ezt a módszert kipróbáltuk a $[16, 36, 4]$ és a $[25, 49, 9]$ problémák megoldásánál is, azonban rendre azokat a megoldásokat adták, mint a $[16, 16, 4]$ és a $[25, 25, 9]$ problémák megoldásai, ezért a továbbiakban ezeket az eseteket nem tárgyaljuk.

6.4. Páratlan táblaméret

6.4.1. 3×3 -as

A 3×3 -as tábla kitöltéséhez szükség van az első 9 betűre. A betűk halmaza abszolút gyakoriság szerint $\{\emptyset, a, e, k, l, n, o, s, t\}$, míg az iteratív sorrend szerint $\{\emptyset, a, e, i, k, l, m, n, t\}$. A helyek halmaza a pozitív egészek 1-től 9-ig, a kapott megoldások pedig rendre:

n	e	l
a	\emptyset	t
k	s	o

k	t	l
a	\emptyset	e
i	n	m

Költség: 1 175 843

$[9, 9, 0]_1$

Költség: 1 236 402

$[9, 9, 0]_2$

6.8. ábra. 3×3 -as tábla 9 betűvel

Megjegyzés: Mivel a két táblán nem ugyanazok a betűk szerepelnek, a költségek összehasonlítása lényegében jelenleg értelmetlen.

6.4.2. 5×5 -ös

A következő lépésben a listák első 25 helyén szereplő betűket helyezzük el az előző pontban kapott táblák középre rögzítésével. A betűk halmaza mindkét esetben a $\{\emptyset, a, á, b, d, e, é, f, g, h, i, j, k, l, m, n, o, ö, r, s, t, u, v, y, z\}$, a helyek listája pedig a pozitív egészek 1-től 25-ig. A következő két megoldás adódik:

f	i	m	b	j
y	n	e	l	á
g	a	∅	t	r
v	k	s	o	d
ö	é	z	h	u

ö	é	r	á	j
z	k	t	l	b
s	a	∅	e	g
v	i	n	m	y
f	h	o	d	u

Költség: 3 594 090

$[25, 25, 9]_1$

Költség: 3 585 879

$[25, 25, 9]_2$

6.9. ábra. 5×5-as tábla 25 betűvel, ebből 9 rögzítve

Érdekes megfigyelni, hogy ebben az esetben az iteratív sorrend szerint kaptunk olcsóbb kiosztást.

6.4.3. 7×7-es

Az utolsó lépésben a maradék 11 betűt is elhelyezzük a már meglévő 25 „köré”, ezek a következők: {c, í, ó, ő, p, q, ú, ü, ű, w, x}. A helyek halmaza most az egészek 1-től 49-ig, ez SK-QAP modell. A magas memóriaigény miatt itt is a 6.3.4 fejezetben bemutatott átírt feladatot oldottuk meg, és kaptuk az alábbi megoldásokat:

	x	ű	ü	ú		
	f	i	m	b	j	
	y	n	e	l	á	ó
w	g	a	∅	t	r	ő
	v	k	s	o	d	
	ö	é	z	h	u	
		p	c	í	q	

		p	ő	ó		
í	ö	é	r	á	j	ü
c	z	k	t	l	b	ú
w	s	a	∅	e	g	ű
	v	i	n	m	y	
	f	h	o	d	u	
			x		q	

Költség: 4 039 959

$[36, 49, 25]_1$

Költség: 4 019 669

$[36, 49, 25]_2$

6.10. ábra. 7×7-es tábla 36 betűvel, ebből 25 rögzítve

A végeredmény: az iteratív sorrend szerinti kiosztás lett a kisebb költségű, azonban a páros méretű táblákkal szemben még ez is gyengének bizonyult.

7. fejezet

Konklúziók, továbblépési lehetőségek

7.1. Összegzés

Munkánk során igyekeztünk minél jobb billentyűzetkiosztást találni az egy ujjal használatos billentyűzetre a magyar nyelv esetén. Munkánk elején a Dell’Amico és társai által írt cikkben [8] is leírt heurisztikus eljárásokkal foglalkoztunk, azonban a kezdeti sikertelenségek után úgy döntöttünk, hogy elsősorban lineáris és egészértékű megoldókat használunk. A probléma, mint minden hozzárendelési feladat, NP-nehéz, így közelítő módszerekkel, a problémát alproblémákra bontva haladtunk.

Elsőként két mohó eljárást alkalmaztunk, majd a NEOS szerver segítségével több megoldót is kipróbáltunk. Az ezt követő munka a Matematika Intézet Omnibus számítógépén, az **AMPL** szoftverrel zajlott. Az első módszer egy linearizációs eljárás volt, az ún. reformulation-linearization módszer, mely során az optimális kiosztás költségére kívántunk alsó becslést kapni. Az első szintű relaxáció nem szolgáltatott elég információt, a második szintű relaxáció futásideje pedig már elég kicsi méret esetén „elszállt”, így ezt az utat elvetettük.

Ezután vettük az RLT-1 relaxációt, azonban megtartottuk a változók bináris tulajdonságát. Az így kapott feladatot a **CPLEX** megoldó a korlátozás és szétválasztás eljárásának segítségével hatékonyan tudta kezelni még $n = 20$ esetén is. Egyre nagyobb méretű, páros, illetve páratlan táblára találtunk billentyűzetkiosztásokat, a lokális minimumot keresve. Ezek közül bemutatnánk a legkisebb költségű négyzet alakú, illetve a legkisebb költségű szabálytalan alakú elrendezést. (Érdekesség, hogy összesítésben is ez a két elrendezés a „legjobb”, azaz az alábbi négyzet alakú táblánál nincs „jobb” sem négyzet alakú, sem szabálytalan alakú tábla.)

w	í	c	v	ö	x
ú	z	s	k	i	f
p	é	a	∅	m	u
j	r	t	e	n	d
ó	á	l	o	g	y
q	ő	b	h	ü	ű

Költség: 4 005 205

QAP: $[36, 36, 16]_1$

			w				
		í	c	v	ö	x	
	ú	z	s	k	i	f	
	p	é	a	∅	m	u	q
	j	r	t	e	n	d	ű
	ó	á	l	o	g	y	
		ő	b	h	ü		

Költség: 4 004 786

SK-QAP: $[36, 64, 16]_1$

7.1. ábra. A két legjobb megoldás

Az ábráról leolvasható, hogy tulajdonképpen azzal, hogy nagyobb helyet hagytunk a betűknek, a szóköztől távolabb eső „sarkakból” a q , w és $ű$ betűk közelebb kerültek a szóközhöz, ettől eltekintve a két kiosztás megegyezik. Gyakorlati szempontból az első kiosztás a vonzóbb, mert sokkal könnyebb a képernyőn elhelyezni egy szabályos 6×6-os négyzetrácsot, mint egy szabálytalan alakút.

7.2. További tervek

A dolgozat témájával kapcsolatban még számos ötlet és technika merült fel, melyek az idő és a dolgozat rövidege miatt nem kaptak említést. A heurisztikus eljárások közül elsősorban a Dell’Amico és munkatársai által publikált cikkben [8] előforduló algoritmusokkal szeretnénk foglalkozni, és azok működését jobban megismerni. Az RLT-2 relaxáció Lagrange-módszer segítségével történő redukálása is egy lehetőség, erről bővebben Adams és társai írnak [1].

Vizsgáltuk a problémát négyzetrács helyett hatszögrácsot feltételezve, és bár a kezdeti kutatások nem hoztak kedvező eredményt, a téma további vizsgálatot igényel. Ezen kívül a *szóköz* billentyű méretét is változtathatjuk, annak helyét lerögzíthetjük, és funkcióbillentyűt is bevonhatunk. Különböző téglalap alakú táblákkal is dolgozhatunk, valamint a problémát kiterjeszthetjük minden nyelvre, és külön elhelyezhetjük a minden nyelvben előforduló betűket valamint a speciális karaktereket. Ez utóbbi esetén összehasonlíthatjuk eredményeinket a heurisztikus eljárásokról szóló cikk eredményeivel.

Köszönetnyilvánítás

Köszönöm témavezetőmnek, Eisenberg-Nagy Mariannának a segítségét, türelmét és a sok közös munkát. Nélküle ez a dolgozat nem jött volna létre ebben a formában. Ezen kívül szeretném megköszönni, hogy a számítások során használhattam a Matematika Intézet, illetve a Differenciálegyenletek Tanszék Operációkutatási csoportjának számítógépeit.

Függelék

Irodalom és AMPL-modellek

A gyakoriság-táblázat készítése során felhasznált könyvek és az AMPL modellek a könnyebb olvashatóság végett a honlapomon tekinthetők meg:

<http://www.math.bme.hu/~vroland/szakdolgozat.xhtml>

Mathematica programkódok

A gyakoriságmátrixot konstruáló algoritmus

Az alábbi programkódban a ToCode parancs a betűket azok ASCII-kódjaira cseréli, majd eléri, hogy a szóköz és a {a, b, ..., y, z, á, é, ..., ü, ő} karakterek rendre az 1-36 számokat kapják. Ezen kívül az írásjeleket is szóköznek tekinti.

```
ToCode[list_] :=  
  Replace[Flatten[ToCharacterCode[ToLowerCase[list]] - 96], {-64 -> 0,  
    129 -> 27, 137 -> 28, 141 -> 29, 147 -> 30, 150 -> 31, 241 -> 32,  
    154 -> 33, 156 -> 34,  
    273 -> 35, -50 -> 0, -52 -> 0, -63 -> 0, -33 -> 0, -51 ->  
    0, -38 -> 0, -37 -> 0, -57 -> 0, -62 -> 0, -49 -> 0, -4 -> 0},  
    2] + 1  
Txt[x_] := StringJoin[ToString[x], ".txt"]  
Freq = Table[0, {36}, {36}];  
y = 1;  
While[y <= 30,  
  Print[y];  
  file = StringSplit[Import[Txt[y]], ""];  
  list = ToCode[file];  
  i = 2;  
  While[i <= Length[Lista],  
    If[list[[i]] >= 1 && list[[i]] <= 36 && list[[i - 1]] >= 1 &&  
      list[[i - 1]] <= 36,  
      Freq[[Min[list[[i]], list[[i - 1]]],  
        Max[list[[i]], list[[i - 1]]]] += 1  
      i++];  
  y++]
```

A mohó módszerek során használt algoritmusok

Az alábbi sorokban a P az egyes betűk helyét, az R az egyes helyeken található betűk indexét adja meg. A `list` tömb a betűk indexét tartalmazza az abszolút gyakoriság, illetve az iteratív algoritmus szerinti sorrendben. A `letter` index végigfut a betűkön, a `place` a maradék helyeken, a `costs` tömb pedig minden egyes lépésben az összes lehetséges elrendezés költségeit tartalmazza az éppen soron levő betűtől függően.

```
P = Table[0, {36}];
R = Table[0, {81}];
P[[1]] = 41;
R[[41]] = 1;
For[letter = 2, letter <= 36, letter++,
  zeros = Flatten[Position[R, 0]];
  costs = {};
  For[place = 1, place <= Length[zeros], place++,
    RNew = R;
    PNew = P;
    RNew[[zeros[[place]]]] = list[[letter]];
    PNew[[list[[letter]]]] = zeros[[place]];
    AppendTo[costs,
      N[Sum[A[[list[[i]]], list[[k]]]]*
        B[[ PNew[[list[[i]]]], PNew[[list[[k]]]] ]],
        {i, 1, letter}, {k, 1, letter}]]];
  minimum = Min[costs];
  minpos = Flatten[Position[costs, minimum]][[1]];
  R[[zeros[[minpos]]]] = list[[letter]];
  P[[list[[letter]]]] = zeros[[minpos]];
]
```

A *wght* mátrixot konstruáló algoritmus

Az alábbi kódban a `LettersR` tömböt a még elhelyezendő betűk indexei alkotják, a `Matrix` tömb pedig sorfolytonosan tartalmazza a már elhelyezett elemeket. A `Matrix` tömb tartalmazza az új, üres helyeket is, ezeket 0-k jelölik.

```
Sulyok = Table[0, {Length[LettersR]}, {m}];
For[i = 1, i <= Length[LettersR], i++,
  For[j = 1, j <= m, j++,
    If[Matrix[[j]] == 0,
      For[k = 1, k <= m, k++,
        If[Matrix[[k]] != 0,
          Sulyok[[i, j]] =
            Sulyok[[i, j]] + freq[[LettersR[[i]], Matrix[[k]]]]*dist[[j, k]]
        ]
      ]
    ]
  ]
]
```

Irodalomjegyzék

- [1] W. P. Adams, M. Guignard, P. M. Hahn, and W. L. Hightower. A level-2 reformulation–linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research*, 180:983–996, 2007.
- [2] W. P. Adams and H. D. Sherali. A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Science*, 32(10):1274–1290, 1986.
- [3] W. P. Adams and H. D. Sherali. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer, Dordrecht, 1998.
- [4] P. Bertier and B. Roy. Une procédure de résolution pour une classe de problèmes pouvant avoir un caractère combinatoire. *ICC Bulletin*, 4:19–28, 1965.
- [5] R. E. Burkard, S. E. Karisch, and F. Rendl. Qaplib - a quadratic assignment problem library. *European Journal of Operational Research*, 55:115–119, 1991.
- [6] R. E. Burkard and K. H. Stratmann. Numerical investigations on quadratic assignment problems. *Naval Research Logistical Quarterly*, 25:129–148, 1978.
- [7] P. Buzing. Comparing different keyboard layouts: Aspects of QWERTY, DVORAK and alphabetical keyboards, July 2003.
- [8] M. Dell’Amico, J. C. D. Díaz, M. Iori, and R. Montanari. The single-finger keyboard layout problem. *Computers & Operations Research*, 36:3002–3012, 2009.
- [9] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology: General*, 47:381–391, 1992.

- [10] A. M. Geoffrion and G. W. Graves. Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment/lp approach. *Operations Research*, 24:595–610, 1976.
- [11] P. M. Hahn and J. Krarup. A hospital facility layout problem finally solved, 2000.
- [12] F. S. Hillier and M. C. Michael. Quadratic assignment problem algorithms and the location of indivisible facilities. *Management Science*, 13:44–57, 1966.
- [13] L. J. Hubert. Assignment methods in combinatorial data analysis. *Textbooks and Monographs Series*, 73, 1987.
- [14] B. Imreh. *Kombinatorikus optimalizálás*. Novadat, 1999.
- [15] T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957.
- [16] J. Krarup. Quadratic assignment. *DATA*, 3(72):12–15, 1972.
- [17] J. Krarup and P. M. Pruzan. Computer-aided layout design. *Mathematical Programming Study*, 9:75–94, 1978.
- [18] J. Krarup and P. M. Pruzan. Computer-aided layout design. *Mathematical Programming Study*, 9:75–94, 1978.
- [19] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.
- [20] J. D. C. Little, K. G. Murty, D. W. Sweeny, and C. Karel. An algorithm for the traveling salesman problem. *Operation Research*, 11:979–989, 1963.
- [21] I. S. MacKenzie, A. Sellen, and W. Buxton. A comparison of input devices in elemental pointing and dragging tasks. *Proceedings of the CHI ‘91 Conference on Human Factors in Computing Systems*, pages 161–166, 1991.
- [22] R. Moe. GRIBB – branch-and-bound methods on the internet. *Lecture Notes in Computer Science*, 3019:1020–1027, 2003.
- [23] M. A. Pollatschek, G. N., and Y. Radday. Optimization of the typewriter keyboard by simulation. *Angewandte Informatik*, 17:438–439, 1976.
- [24] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the Association of Computing Machinery*, 23:555–565, 1976.