Eötvös Loránd Tudományegyetem

Természettudományi Kar

Dencs Veronika

Autonóm járművek útvonaltervezésének modellezése

BSc szakdolgozat

Témavezető:

Király Tamás

Operációkutatási Tanszék

Budapest, 2018

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Király Tamásnak a szakdolgozatom megírása közben nyújtott segítségéért és útmutatásaiért, illetve hogy kérdéseimmel bármikor fordulhattam hozzá.

Továbbá szeretném megköszönni szüleimnek a tanulmányaim során nyújtott támogatásukat, hogy lehetővé tették számomra mindezt.

Tartalomjegyzék

Bevezetés

A témakörömnek választott útvonaltervezés optimalizálás manapság nagyon sokrétűen alkalmazható módszer a gyakorlatban. Én kifejezetten az űrjárművek trajektóriáinak lineáris programmal történő modellezésével fogok foglalkozni a szakdolgozatomban. Emellett számos további felhasználási területe van az optimalizálásnak az operációkutatás témakörében, csak úgy mint a robotika, szállítás optimalizálás, hatékony gyártási folyamatok, mezőgazdasági termelések optimalizálása.

A szakdolgozatom első fejezetében a lineáris programozásról írok röviden egy összefoglalót mielőtt rátérnék a következő fejezetben a dolgozatom fő témájára, az űrhajók röppályatervezésének modellezésére. A modellezést bemutató fejezetben részletesen tárgyalom a gyakorlatban előforduló körülményeket, melyeket a modell elkészítése közben figyelembe kell venni, abba beépíteni. Majd végezetül az utolsó fejezetben a gyakorlatba átültetve a korábban tárgyalt modellt, saját implementációimon mutatok be néhány konkrét példát.

A címben is már említett autonóm jármű olyan járművet jelent, amely képes a környezetét észlelni és emberi közbeavatkozás nélkül irányítani saját magát. Az autonóm járművek működése nagyon komplex folyamat, az útvonaltervezésen kívül több komponens is szükséges hozzá (például: környezet észlelése, feldolgozása, lokalizáció meghatározása), amelyek kivitelezéséről nem esik szó a szakdolgozatomban.

1. fejezet

Bevezetés a lineáris programozás témakörébe

Bevezetésként áttekintenék néhány, a későbbiek során felhasznált fogalom jelentését és algoritmus működését. Ehhez az [1, 2, 3, 4] forrásokat használom fel.

1.1. Emlékeztető: a lineáris program fogalma

1.1.1. Definíció (Poliéder)**.** Véges sok féltér metszete, $R := \{x : Ax \leq b\}$, ahol A egy $m \times n$-es mátrix, $b \in \mathbb{R}^m$ vektor. Tehát a poliéder egy lineáris egyenlőtlenségrendszer megoldáshalmaza.

1.1.2. Definíció (Lineáris programozási feladat)**.** Keressük meg adott lineáris, $\mathbb{R}^n$ értelmezési tartományú függvény, az ún. célfüggvény szélsőértékét (minimumát vagy maximumát) adott poliéder felett. Azaz $f(x) : \mathbb{R}^n \to \mathbb{R}$, $f(x) = cx$, $(c, x \in \mathbb{R}^n)$ lineáris célfüggvény optimumát keressük, az alábbi feltételek mellett:

$$Ax \leq b \tag{1.1}$$

$$x \geq 0 \tag{1.2}$$

ahol $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$.

1.1.3. Állítás. *Amennyiben az $R = \{x : Ax \leq b\}$ poliéder nem üres, és az $f(x) = = cx$, $x \in R$ célfüggvény felülről korlátos, úgy létezik maximuma.*

1.1.4. Definíció (Egészértékű lineáris programozási feladat)**.** Keressük az 1.1.2 definícióban felírt feltételek mellett az $f(x)$ célfüggvény optimumát, kiegészítve $x \in \mathbb{Z}^n$ feltétellel. Bizonyos esetekben csak egyes változókról tesszük fel az egészértékűséget, ekkor kevert egészértékű lineáris programozásról beszélünk.

Belátható az egészértékű lineáris programozásról, hogy NP-teljes probléma. A bonyolultsági osztályok gyors áttekintése érdekében röviden írok a három, témához kapcsolódó bonyolultsági osztályról.

Ezek a bonyolultsági osztályok a P (polinomiális), NP (nemdeterminisztikusan polinomiális) és NP-teljes osztályok. Azt mondjuk, hogy egy probléma polinomiális időben megoldható, azaz P-ben van, ha létezik k konstans, hogy n hosszú input esetén $O(n^k)$ idő alatt megoldható. Egy problémára azt mondjuk, hogy NP-ben van, ha polinom időben ellenőrizhető. Azaz figyelmen kívül hagyjuk azt, hogyan tudunk bizonyítékot találni, a fontos, hogy polinom időben ellenőrizhető legyen ez a bizonyíték. Bármely P-beli probléma az NP-beli osztályba is beletartozik, $P \subseteq NP$. Még nem ismert, hogy P valódi részhalmaza-e NP-nek. Az NP-teljes problémák halmazába olyan problémák tartoznak, amelyek legalább olyan nehezek, mint bármely NP-beli probléma. Vagyis egy probléma NP-teljes, ha

1. a probléma NP-beli, és
2. minden NP-beli probléma visszavezethető rá polinomiális időben.

Ha az NP-teljes problémák közül akár csak egy is megoldható lenne polinomiális időben, akkor valamennyi NP-beli probléma megoldható lenne P-ben, és ez P=NP érvényességét jelentené.

1.1.5. Állítás. *Az egészértékű lineáris programozás NP-teljes probléma.*

Bizonyítás. Az állítás bizonyításához az előzőleg említett két pontot kell belátni. Az, hogy a probléma NP-ben van, könnyen igazolható. A második pont igazolásához a 3SAT NP-teljes problémát vezetem vissza az egészértékű lineáris programozásra polinom időben. A 3SAT nyelv az olyan kielégíthető konjunktív nomálformák által alkotott nyelvet jelenti, ahol minden klózban legfeljebb 3 literál fordul elő.

Jelölje $a_1, ..., a_n$ a literálokat a 3SAT nyelvben, $z_1, ..., z_n$ pedig a megfelelő változókat az egészértékű LP feladatban. Tegyük fel minden z_i változóról, a következőt: $0 \leq z_i \leq 1$. Amennyiben $z_i = 1$ az LP feladatban, legyen $a_i =$ igaz a formulában, $z_i = 0$ esetben pedig legyen $a_i =$ hamis.

Ekkor minden klóz megfeleltethető egy lineáris egyenlőtlenségnek. Például a $(a_1 \vee a_2 \vee \neg a_3)$ megfeleltethető a $z_1 + z_2 + (1 - z_3) \geq 1$ egyenlőtlenséggel. Ekkor a kettő pontosan ugyanakkor elégíthető ki. A klózokat a $\bigvee$ jellel összefűzve olyan 3-normálformát kapunk, mely akkor és csak akkor elégíthető ki, ha az egyenlőtlenségekből kapott LP feladat kielégíthető. □

A fejezetben szó lesz egy algoritmusról, amely a gyakorlatban használatos egészértékű megkötéseket tartalmazó LP feladatok megoldására. Előtte viszont tekintsük át a relaxált LP feladatok megoldásánál használt szimplex algoritmust.

1.2. Szimplex algoritmus

A szimplex algoritmusnak, sok más algoritmushoz hasonlóan, többféle változata létezik optimalizálási feladatok megoldására. Ilyen a primál, a duál és a kétfázisú

szimplex algoritmus, illetve változataik. Az algoritmus lépéseinek áttekintéséhez a primál változat lépéseit írom le vázlatosan. Ehhez először is szükséges pár fogalom tisztázása.

Tekintsük a következő primál feladatot:

$$\begin{aligned} Ax &= b \\ x &\geq 0 \\ \max(cx) & \end{aligned}$$

és a hozzá tartozó duál feladatot:

$$\begin{aligned} yA &\geq c \\ \min(yb) & \end{aligned}$$

Itt $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $x \in \mathbb{Q}^n$, $y \in \mathbb{Q}^m$, illetve tegyük fel, hogy az A mátrix rangja m.

1.2.1. Állítás. *Legyen x megoldása a primál feladatnak, y megoldása a duál feladatnak. Ekkor $cx \leq yb$.*

1.2.2. Tétel (Erős dualitás tétel)**.** *Ha a primál feladat megoldható, és az optimuma korlátos (azaz cx nem tetszőlegesen nagy), akkor* $\max(cx) = \min(yb)$.

1.2.3. Definíció (Bázis, bázismegoldás)**.** A primál feladat bázismegoldása egy olyan x megoldás, amire A-nak tekintve azon oszlopait, amelyekre $x_j \geq 0$, lineárisan függetlenek. Bázisnak nevezzük A egy $m \times m$-es invertálható B részmátrixát. Rögzített B bázis mellett egy $x \in \mathbb{R}^n$ vektor $x = (x_B, x_N)$ alakú felírásában x_B-vel jelöljük a bázishoz tartozó, és x_N-nel a bázishoz nem tartozó koordinátákat.

1.2.4. Definíció (Primál megoldás, duál megoldás, megengedett megoldás)**.** Legyen B bázis. A hozzá tartozó primál megoldás $x \in \mathbb{R}^n : x_B = B^{-1}b, x_N = 0$. Ez az x megengedett, ha $x \geq 0$. A B-hez tartozó duál megoldás $y \in \mathbb{R}^n : y = c_B B^{-1}$. Ez megengedett, ha $yA \geq c$.

1.2.5. Definíció (Primál, duál megengedett bázis)**.** B primál megengedett, ha a hozzá tartozó primál megoldás megengedett. B duál megengedett, ha a hozzá tartozó duál megoldás megengedett.

1.2.6. Állítás. *Ha B egyszerre primál és duál megengedett, akkor B optimális.*

Bizonyítás. Mivel x primál megengedett, ezért $x \geq 0$, és mivel y duál megengedett, így $yA \geq c$ is teljesül. Ezért használhatjuk az 1.2.1 állítást a következő felírásában: $c_B x_B = cx \leq yb = yBx_B = c_B x_B$. Azaz $cx = yb$, ami azt jelenti, hogy x és y optimális megoldások. □

Most rátérhetünk a primál szimplex algoritmus lépéseinek áttekintésére. Ez a szimplex algoritmus egy olyan egyszerűbb változata, ahol kezdetben rendelkezésünkre áll egy primál megengedett bázis. A célunk egy optimális bázis megtalálása a

primál megengedett bázisokon való lépkedések során. Amikor találunk egy optimális bázist, akkor találtunk egy optimális megoldást is.

A primál szimplex módszer lépései (Feltettük, hogy kiindulásként adott egy B primál megengedett bázis.):

1. Legyen $y = c_B B^{-1}$.
2. Ha $yA \geq c$, akkor B optimális. KÉSZ.
3. Ellenkező esetben létezik j, hogy $ya_j \leq c_j$. A Bland-szabályt alkalmazva, válasszuk a legkisebb ilyen j-t, és az ennek megfelelő oszlop kerül majd az új bázisba.
4. Ekkor definiáljuk x'-t a következőképpen: $x'_B = -B^{-1}a_j$, $x_j = 1$, többi koordinátában 0. Ekkor egyrészt $Ax' = Bx_B + a_j = 0$, másrészt $0 = y(Ax') = = (yA)x' < cx'$. A továbbikban két lehetőség áll fenn:
   a) $x' \geq 0$. Ebben az esetben a primál feladatunk nem korlátos felülről, mivel minden $\lambda > 0$ esetén $x + \lambda x' \geq 0$ és $A(x + \lambda x') = b$ $Ax' = 0$ miatt. Az algoritmus megáll.
   b) $x' \ngeq 0$. Ebben az esetben megkeressük azt az oszlopot, amit kiveszünk a bázisból. Legyen $\lambda^* = \min\left\{-\frac{x_i}{x'_i} : i : x'_i < 0\right\}$, $i = \arg\min\left\{-\frac{x_i}{x'_i} : i : x'_i < 0\right\}$. Ekkor λ^* pozitív, és i csak bázisbeli oszlop lehet. Továbbá minden k-ra $x_k + \lambda^* x'_k \geq 0$, ugyanis ha $x'_k \geq 0$, akkor a pozitív x_k-hoz adunk nemnegatív értéket, ha pedig $x'_k < 0$, akkor $0 = x_k + \left(-\frac{x_k}{x'_k}\right) x'_k \leq x_k + \lambda^* x'_k$.
5. A bázisban az i-ik oszlopot helyettesítjük a j-ik oszloppal. Az így kapott új B' bázishoz tartozó primál megoldás $x + \lambda^* x'$: $B'(x_B + \lambda^* x'_{B+j}) = B(x_B + + \lambda^* x'_B) + \lambda^* a_j = Bx_B + \lambda^* Bx'_B + \lambda^* a_j = Bx_B + \lambda^*(-a_j) + \lambda^* a_j = Bx_B = b$. Ez megengedett megoldás lesz, mivel előbb láttuk, hogy $x_k + \lambda^* x'_k \geq 0$ minden k-ra.
6. Folytassuk az 1. ponttal.

1.2.7. Tétel. *A szimplex algoritmus véges sok lépésben véget ér a Bland-szabályt használva.*

Mivel a gyakorlatban nem áll rendelkezésre kezdeti primál megengedett bázis, a kétfázisú szimplex módszert szokták használni. Ennek első fázisában primál megengedett bázist keres az algoritmus, míg a második fázisban ezt a bázist felhasználva lehet alkalmazni a primál szimplex módszert.

1.3. Branch-and-bound algoritmus

A későbbiekben szó lesz erről az algoritmusról, így érdemes megemlíteni röviden, hogy milyen elv alapján működik, és mire tudjuk használni. A szimplex algoritmushoz hasonlóan ennek is létezik több verziója, ezek közül egyet írok le.

A lineáris programozási feladatoknál sokkal nehezebben, és hosszabb idő alatt megoldhatóak az egészértékű lineáris programok. Egészértékű lineáris programok (későbbiekben ILP - integer linear program) megoldásánál az egyik leggyakrabban alkalmazott módszer a branch-and-bound algoritmus, lineáris programok megoldására használt solverek ennek az algoritmusnak az elve alapján működnek. A branch-and-bound algoritmus során a szimplex algoritmus többször is lefut. Vannak esetek, amikor ez elég sokszor következik be, mire megkapjuk az optimális megoldást, ebből is látszik, hogy ILP feladatok optimumának megtalálása jelentősen hosszabb futási időt igényel, mint azokban az esetekben, amikor nincsenek egészértékű megkötések.

Az algoritmus a következőképpen fut: vegyünk egy megoldandó LP feladatot bizonyos változók egészértékű megkötéseivel. A módszer lényege, hogy minden lépésben szétválasztjuk a megoldások halmazát (innen ered a branch elnevezés), megkeressük az ágakon a relaxált LP feladat optimumát, majd összehasonlítjuk ennek értékét az eddigi megtalált legjobb egészértékű megoldással (erre utal a bound elnevezés), mivel az algoritmus a futás során azt számon tartja (amíg nincsen ilyen egészértékű megoldás, addig azt maximum keresése esetén $-\infty$-nek, minimum keresése esetén ∞-nek tekinti az algoritmus). A megoldáshalmazokon további lépésekben folytatjuk az optimum keresését ugyanezzel a módszerrel, miközben próbálunk egyes halmazokat elvetni a halmazoknak megfelelő relaxált LP feladatok optimumának és az addig megtalált legjobb egészértékű megoldás felhasználásával (jelöljük ezt L-lel). Ha szemléletesen szeretnénk ábrázolni, akkor egy bináris fát képzejünk el, amely egyre mélyebbre ágazik, ugyanakkor az optimális megoldást nem tartalmazó halmazok elvetésével egyre fogy is. A következőkben pontokba szedve olvashatók az algoritmus lépései egy maximum keresés esetén.

1. Oldjuk meg egy kiválasztott H halmazon a relaxált LP feladatot, azaz töröljük az egészértékű megkötéseket, és keressük az így kapott feladat $u(H)$ optimumát, melyet a x^* pontban vesz fel.

2. Ha az optimum az egészértékű megkötéseket teljesíti, akkor $L = \max(L, u(H))$.

3. Ha $u(H) < L$, akkor H halmaz már biztosan nem tartalmazza az optimális megoldást, ezt a halmazt elvethetjük. Folytassuk az 1. ponttal.

4. Ha a H halmazon nincs megoldása az LP feladatnak, akkor ezt a halmazt elvethetjük. Folytassuk az 1. ponttal.

5. Különben: létezik i index, amire $x_i^* \notin \mathbb{Z}$. Most bontsuk szét a a H halmazt úgy, hogy az egyik ághoz vegyük hozzá az $x_i \leq \lfloor x_i^* \rfloor$, a másik ághoz az $x_i \geq \lceil x_i^* \rceil$ megkötéseket. Ezzel nem hagyunk figyelmen kívül egy lehetséges megoldását sem az IPL feladatnak. Folytassuk az 1. ponttal.

Az algoritmus addig folytatódik, amíg minden halmazt megvizsgáltunk.

2. fejezet

Űrhajók röppályatervezése kevert egészértékű lineáris programozás segítségével

2.1. Bevezetés

Röppályák, útvonalak optimális tervezése a gyakorlatban gyakran előforduló probléma. Az űreszközök - ezek a világűrbe indított mesterséges égitestek, melyek lehetnek automata műholdak, űrszondák, teher vagy ember szállítására alkalmas űrhajók - röppályatervezésének modellezését NASA vagy U.S. Air Force küldetések inspirálják. Például egy olyan küldetés, ahol műhold rajt használnak egy egyedülálló műhold helyett olyan előnyökkel jár, mint például egy-egy műhold meghibásodása esetén a küldetés könnyű rekonfigurálhatósága. Az ilyen küldetéseknek egy összehangolt formációs repülésre van szükségük, ráadásul törekedve a költségek minimalizálására.

Ebben a fejezetben egy üzemanyag optimalizációs módszert ismertetek űrjárművek röppályatervezésére lineáris programozás használatával. A röppályatervezés során különböző akadályokra kell figyelni, mint például ütközések elkerülése vagy egy űrjármű csóvája kárt tehet más járművekben, ha elhanyagolunk bizonyos megkötéseket. Az optimalizáció során különböző bináris változókat tartalmazó megkötéseket fogunk a kezdeti üzemanyag optimalizációs lineáris programunkhoz hozzávenni, így végeredményként egy kevert egészértékű lineáris programot (mixed-integer linear program) fogunk kapni. A kezdeti LP problémát három kibővítés fogja követni, egyrészt fontos a járművek akadállyal és egymással való ütközésének elkerülése, másodrészt üzemanyag használat esetén más jármű vagy akadály nem kerülhet a jármű csóvájának útjába, harmadrészt pedig egy végső konfiguráció kialakítását írom fel. Mindegyik bővítés során megjelennek majd a bináris változók, melyek fontosságáról mindjárt írok.

Az olyan jellegű útvonaltervező feladatokról, ahol akadályok is megjelennek, belátható, hogy NP-teljesek. Szemléletes érvelés erre az egyszerűség kedvéért kétdi-

menziós esetben az, hogy az útvonal során, mikor egy jármű akadállyal találkozik, el kell dönteni, hogy balról vagy jobbról kerüli-e ki azt. Az akadályok számának növelésével a döntések száma exponenciálisan fog növekedni. A választási lehetőségek ilyenkor olyan kifejezések felírását adják, ahol az egyes egyenletek, egyenlőtlenségek közti logikai kapcsolat *vagy* kapcsolat. A bináris változók segítségével tudunk felírni olyan egyenlőtlenségeket, ahol ez kiküszöbölhető, és a gyakorlatban alkalmazható *és* logikai kapcsolatokat kapunk. Az így kapott kevert egészértékű lineáris programok (későbbiekben MILP) bár nem túl gyorsan, de megoldhatóak a branch-and-bound algoritmus használatával. A legnagyobb előnye a MILP használatának trajektória optimalizálások során, hogy mindenki számára hozzáférhető softwarek segítségével megoldható. Ezek közül többen a branch-and-bound algoritmust hajtják végre, és elég sok adatot magába foglaló problémák megoldhatóak a gyakorlatban használható időn belül.

A trajektória optimalizációs probléma MILP formája mindenhol lineáris, így a globális minimum megtalálása garantált, a módszer lokális minimumoktól mentes. Mivel minden kikötés lineáris a MILP problémában, így szintén arra törekedünk, hogy a rendszer dinamikáját leíró formula, melyről a következő részben bővebben szót ejtek, lineáris legyen. Bár én egy másik egyenlőséget fogok használni a későbbiekben a dinamika leírására és az implementálás során, szintén érdemes megemlíteni a gyakorlatban sokszor alkalmazott linearizált Hill egyenleteket, mely jó modellt nyújt olyan problémákra, ahol járművek egymáshoz közel operálnak az űrben, és relatív rövid időn belüli manővereket hajtanak végre.

A fejezet végén továbbá szó lesz arról, milyen módszerek állnak rendelkezésünkre, hogy felgyorsítsuk a feladat futási idejét. Ezen esetekben felesleges, inaktív kikötések elhagyása a cél az adott feladat és információk ismerete alapján. Sok esetben például előre megjósolható, hogy az optimális megoldás úgy fog kinézni, hogy csak a röppálya elején, illetve végén használ üzemanyagot a jármű, a köztes időben viszont nem. Ilyen, és ehhez hasonló megfontolásokkal elhagyhatunk számos kikötést a feladatból, ráadásul a kapott megoldást egyszerűen ellenőrizhetjük, hogy valóban minden időpillanatban minden megkötésünket teljesíti-e.

A fejezetet a [5] forrás alapján dolgoztam fel, míg a rendszer dinamikáját leíró egyenletet a [6, 7] forrásból merítettem.

2.2. A probléma formalizálása

Ebben az alfejezetben adjuk meg a röppálya optimalizációs feladatra a modellt kevert, egészértékű lineáris programozással.[5] A későbbiekben a következő jelöléseket, illetve indexeket fogom használni:

$G =$ a járművek végső konfigurációinak halmaza, illetve annak számossága
$M =$ tetszőleges pozitív szám, mely nagyobb a feladat során előforduló bármely távolságnál
$N =$ dimenzió
$P =$ csóva hossz
$T =$ időegységek száma
$\mathbf{u} =$ a járművek gyorsulása/üzemanyag fogyasztása, illetve ezzel tudjuk számolni a költségünket is
$V =$ járművek száma
$W =$ csóva szélesség
$\mathbf{x} =$ jármű helyzete

Indexek:
$i =$ időpont
$l =$ akadály
$n, m =$ koordináták
$p, q =$ járművek
$r =$ pozíció

2.2.1. Az alap LP-probléma

Az optimalizálásról először is megemlítendő, hogy nem folytonosan írja le a folyamatot, azaz az időpontok és az $\mathbf{u}$ vektor értékei is mind diszkrétek. Annak ellenére, hogy a folyamat diszkretizálva van, fontos, hogy a rendszer dinamikájával összhangban legyen. A járművek dinamikájáról a következő pontban írok részletesen, ahol bővebb magyarázattal szolgálok az ebben a pontban használt, a járművek viselkedését a diszkretizált rendszerben leíró lineáris egyenletrendszer hátteréről.

A kezdő pillanatban minden jármű helyzetét inputként meg kell adnunk:

$$\mathbf{x}_{0p} = \mathbf{x}_{Sp} \tag{2.1}$$

ahol $\mathbf{x}_{Sp}$ a p járműhöz tartozó kezdeti állapotvektor. Minden egyes p járműhöz megadható egy $\mathbf{x}_{Fp}$ végső állapotvektor is hasonló módon:

$$\mathbf{x}_{Tp} = \mathbf{x}_{Fp} \tag{2.2}$$

de akár lehetséges az is, hogy egy kevésbé kötött eljárást használjunk, mely során csupán a lehetséges végső konfigurációk halmazát adjuk meg. Ekkor több kikötést

is meg kell adnunk a lineáris programunkban, ennek leírásával egy későbbi pontban foglalkozom részletesebben.

Az egyszerűség kedvéért az $n = 2$ esetre írjuk fel a diszkrét rendszer dinamikáját leíró egyenletrendszert.[6] A síkban a jármű állapota 4 koordinátával írható le, ezek a koordináták a vízszintes és függőleges pozíció, illetve a vízszintes és függőleges sebesség: $\mathbf{x} = (x, y, \dot{x}, \dot{y})$. A vízszintes és függőleges gyorsulást pedig az $\mathbf{u} = (u_x, u_y)$ vektor adja meg. Ekkor minden i időpillanatra a következő összefüggést írjuk fel, ahol α jelöli két diszkrét időpont között eltelt időt:

$$\mathbf{x}_{(i+1)p} = \mathbf{A}\mathbf{x}_{ip} + \mathbf{B}\mathbf{u}_{ip} \tag{2.3}$$

ahol

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1-e^{-\alpha} & 0 \\ 0 & 1 & 0 & 1-e^{-\alpha} \\ 0 & 0 & e^{-\alpha} & 0 \\ 0 & 0 & 0 & e^{-\alpha} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \alpha - 1 + e^{-\alpha} & 0 \\ 0 & \alpha - 1 + e^{-\alpha} \\ 1 - e^{-\alpha} & 0 \\ 0 & 1 - e^{-\alpha} \end{pmatrix}$$

Feltesszük még a továbbiakban, hogy a folyamat során bizonyos értékeken belül mozoghatnak csak a vektorok értékei:

$$-x_{min} \leq x_{ipn} \leq x_{max} \tag{2.4}$$
$$-u_{min} \leq u_{ipn} \leq u_{max} \tag{2.5}$$

ahol a változók az i időponthoz, n koordinátához és p járműhöz tartoznak.

A feladatunk minimalizálni az összes jármű üzemanyagfogyasztásának összegét, azaz a költségünket szeretnénk optimalizálni. Ezt úgy tesszük meg, hogy minden járműre minden időpillanatban vesszük az $\mathbf{u}_{ipn}$ vektor értékeinek összegét. Mivel a jármű gyorsulása összhangban van az üzemanyagfogyasztással, illetve a költségekkel, így ez lesz a mérőszámunk a költségek számolásánál. Vagyis a költségfüggvény minimumát

$$\min \sum_{i=0}^{T-1} \sum_{p=1}^{V} \sum_{n=1}^{N} |u_{ipn}| \tag{2.6}$$

alakban írjuk fel.

Ezzel megkaptuk az alap probléma esetén a lineáris programunkat. Ezt a következő pontokban bővítjük különböző megkötésekkel, úgy hogy minden megkötés hozzávételével egy MILP feladatot kapjunk.

2.2.2. Járművek dinamikája

Ebben a fejezetben két példát is mutatok járművek dinamikájának leírására. Az egyik ezek közül a leggyakrabban használt forma űrhajók dinamikájának leírására, ugyanakkor nem szükséges ehhez ragaszkodni, így a későbbiekre tekintettel bemutatok egy másik rendszert is.

A Hill egyenletek az amerikai csillagász, matematikus George William Hill nevéhez köthetők, aki már 1886-ban ismertette ezeket az egyenleteket periodikus trajektóriák vizsgálata kapcsán. Linearizált formája a Hill egyenleteknek a következőképpen néz ki $n = 3$ esetén, ahol m a jármű tömegét, ω a pálya szögsebességét jelöli, továbbá $\mathbf{x} = (x, y, z, \dot{x}, \dot{y}, \dot{z})$:

$$\mathbf{x}_{(i+1)p} = \mathbf{A}\mathbf{x}_{ip} + \mathbf{B}\mathbf{u}_{ip} \tag{2.7}$$

ahol

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3\omega^2 & 0 & 0 & 0 & 2\omega & 0 \\ 0 & 0 & 0 & 2\omega & 0 & 0 \\ 0 & 0 & -\omega^2 & 0 & 0 & 0 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/m \end{pmatrix}$$

A másik dinamikai rendszert, bár nem kifejezetten űrhajók, hanem például robotok mozgására használják [6, 7], a mi rendszerünkre is alkalmazhatjuk. Az egyszerűség kedvéért most is, és később az implementáció során is $n = 2$ esetre írjuk fel. A linearizált alakhoz a következő differenciálegyenletet használjuk fel a járművek mozgásának leírásához, ahol $x(t)$ a t időpontban a jármű helyzetét, $\dot{x}(t)$ a jármű sebességét és $\ddot{x}(t)$ a jármű gyorsulását jelöli a x koordináta mentén, y koordináta mentén hasonlóan:

$$\ddot{x}(t) + \dot{x}(t) = u_x(t)$$
$$\ddot{y}(t) + \dot{y}(t) = u_y(t)$$

azzal a megkötéssel, hogy bármilyen irányban a gyorsulásvektor értéke nem haladhatja meg az 1-et:

$$u_x^2(t) + u_y^2(t) \leq 1.$$

A differenciálegyenletek megoldása

$$x(t) = c_1 e^{-t} + u_x(t)t + c_2$$
$$y(t) = d_1 e^{-t} + u_y(t)t + d_2$$

tetszőleges c és d konstansokkal. A differenciálegyenletekben $u_x(t)$ és $u_y(t)$ szintén konstansokként kezelendőek, viszont korábban kikötöttük, hogy ezek értékei bizonyos határok között mozoghatnak csak.

A diszkrét linearizált egyenleteket így a következő alakban adják meg [7], ahol α jelöli a diszkrét időpontok között eltelt időt, illetve $\mathbf{x} = (x, y, \dot{x}, \dot{y})$:

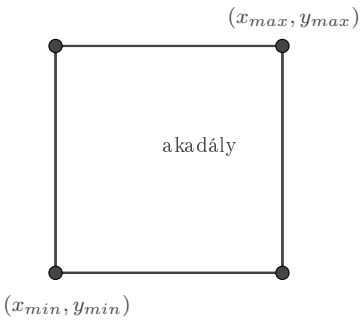
$$\mathbf{x}_{(i+1)p} = \mathbf{A}\mathbf{x}_{ip} + \mathbf{B}\mathbf{u}_{ip}$$

ahol

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1-e^{-\alpha} & 0 \\ 0 & 1 & 0 & 1-e^{-\alpha} \\ 0 & 0 & e^{-\alpha} & 0 \\ 0 & 0 & 0 & e^{-\alpha} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \alpha - 1 + e^{-\alpha} & 0 \\ 0 & \alpha - 1 + e^{-\alpha} \\ 1 - e^{-\alpha} & 0 \\ 0 & 1 - e^{-\alpha} \end{pmatrix}$$

2.2.3. Akadályok elkerülése

Az egyszerűség kedvéért először az $n = 2$ esetet tárgyaljuk. A négyszögletes akadályok helyzetét a bal alsó sarkot $(x_{\min}, y_{\min})$, és a jobb felső sarkot $(x_{\max}, y_{\max})$ jelölő vektorokkal határozzuk meg.



2.1. ábra. Akadály 2 dimenzióban

Az i-ik időpillanatban a jármű helyzetét a (x_i, y_i) vektor jelöli, és minden időpillanatban ennek a pontnak az akadályon kívüli területen kell elhelyezkednie. Ezt az alábbi rendszer írja le:

$$\begin{aligned} & x_i \leq x_{\min} \\ \text{vagy} \quad & x_i \geq x_{\max} \\ \text{vagy} \quad & y_i \leq y_{\min} \\ \text{vagy} \quad & y_i \geq y_{\max} \end{aligned} \tag{2.8}$$

Mivel az akadállyal nem ütközhet a jármű, így a négy egyenlőtlenség közül legalább egynek teljesülnie kell, különben $x_{\min} < x_i < x_{\max}$ és $y_{\min} < y_i < y_{\max}$ egyenlőtlenségek állnak fent. Ez azt jelentené, hogy az (x_i, y_i) vektor, ami a jármű helyzetét jelöli, megegyezik az akadály egy belső pontjának koordinátáival, azaz ütközés történne.

A (2.8) rendszerrel olyan ekvivalens alakot fogunk megadni, amelyben minden időpillanatban bináris változók egy halmaza van hozzárendelve a problémához, és amelyek száma a dimenziószám függvénye. Legyen M egy tetszőlegesen választott szám (például a legnagyobb távolság háromszorosa), amely nagyobb bármely, a problémában előforduló távolságnál. Ekkor a következőképpen írjuk fel a problémát:

$$\begin{aligned} & x_i \leq x_{\min} + Ma_1 \\ \text{és} \quad & -x_i \leq -x_{\max} + Ma_2 \\ \text{és} \quad & y_i \leq y_{\min} + Ma_3 \\ \text{és} \quad & -y_i \leq -y_{\max} + Ma_4 \\ \text{és} \quad & \sum_{k=1}^{4} a_k \leq 3 \end{aligned} \tag{2.9}$$

Ez a felírás azért helyes, mert ha bármelyik bináris változó 1 értéket vesz fel, akkor nem befolyásolja az adott koordináta értékét, mivel M-et olyan nagy számnak választjuk, mely nagyobb, mint a feladatban szereplő bármely távolság. Ugyanakkor vegyük észre, hogy ha $a_k = 0$, akkor a k-ik egyenlőtlenség teljesül (2.8)-ban. A (2.9) egyenlőtlenségrendszerben az utolsó sor biztosítja, hogy háromnál több az a_i-k közül nem veheti fel az 1 értéket, azaz legalább egy 0 értéket kell felvegyen. Ez pedig biztosítja, hogy a (2.8) egyenlőtlenségei közül legalább egy teljesül.

Ezt általánosíthatjuk több dimenziós esetre, illetve a lépések és járművek számát szintén tetszőleges módon választhatjuk meg. A p jármű i-ik pillanatban lévő helyzetét leíró vektor $\mathbf{x_{ip}} = [x_{ip1}, \ldots, x_{ipn}]^T$, ahol n a koordinátát jelöli. Az akadályokat továbbra is $[a_1, b_1] \otimes \ldots \otimes [a_n, b_n]$ tégláknak tekintjük. Az l akadály azon csúcsa, amelyben minden koordináta a minimum értéket veszi fel (kétdimenziós esetben a bal alsó sarok), legyen $\mathbf{L_l}$ csúcs. Hasonlóan $\mathbf{U_l}$ jelölést használunk annak a csúcsnak a jelölésére, ahol minden koordináta a maximális értéket veszi fel. $\mathbf{L_l}$ és $\mathbf{U_l}$ a dimenziószámmal megegyező méretű vektorok. A bináris változók indexeit kibővitjük, a_{iplk} esetén i az adott időpillanatot, p a járművet, l az akadályt jelöli, míg $k \in [1, \ldots, 2N]$,

ahol N a dimenziószám. Ekkor a teljes formulát a következőképpen írhatjuk fel (az előzőhöz képest minimálisan átrendezve):

$$\begin{aligned} &\forall p, \forall l, \forall i, \forall n \in [1, \ldots, N]: \\ &\qquad x_{ipn} \leq L_{ln} + M a_{ipl(n+N)} \\ &\text{és} \qquad x_{ipn} \geq U_{ln} - M a_{ipln} \\ &\text{és} \qquad \sum_{k=1}^{2N} a_{iplk} \leq 2N - 1 \end{aligned} \tag{2.10}$$

ahol L_{ln} és U_{ln} a két vektor megfelelő elemeit jelöli. Hasonlóan a kétdimenziós esethez, az utolsó sor biztosítja, hogy legalább egy bináris változó 0 értéket vegyen fel, azaz legalább egy j koordináta esetén teljesüljön, hogy $x_{ipj} \geq U_{lj}$ vagy $x_{ipj} \leq L_{lj}$.

Az előző fejezetben felvázolt alapproblémát ezekkel a kikötésekkel bővíthetjük.

2.2.4. Ütközések elkerülése

Az akadályok elkerülése mellett fontos figyelembe venni a különböző járművek ütközésének elkerülését is. Bármely tetszőlegesen választott két jármű esetén rögzítenünk kell, hogy bármely irányban és időpillanatban nem közelíthetik meg egymást tetszőlegesen közel. Ez a megkötés minden járműhöz egy négyszögletes területet jelöl ki, melybe a folyamat során nem léphet más jármű. Az előző ponthoz hasonlóan, először $N = 2$ esetet vizsgáljuk. Jelölje r_x és r_y az X és Y koordinátáknak megfelelő irányú távolságokat, amelynél közelebb nem közelítheti meg egymást két különböző jármű, p és q. A p és q jármű i időpontbeli helyzetét jelöli (x_{ip}, y_{ip}) és (x_{iq}, y_{iq}). A megkötéseket az alábbi módon írhatjuk fel:

$$\begin{aligned} & x_{ip} - x_{iq} \geq r_x \\ \text{vagy} \quad & x_{iq} - x_{ip} \geq r_x \\ \text{vagy} \quad & y_{ip} - y_{iq} \geq r_y \\ \text{vagy} \quad & y_{iq} - y_{ip} \geq r_y \end{aligned} \tag{2.11}$$

Az implementálás során a következő ekvivalens átírást használjuk, melyben szintén alkalmazunk bináris változókat, és M az előzőekhez hasonlóan egy olyan nagy, természetes számot jelöl, mely nagyobb minden problémában előforduló távolságnál. Korábbiakban látottakhoz hasonlóan az utolsó sor ez esetben is biztosítja, hogy legalább egy bináris változó értéke 0 legyen, ezáltal a (2.11) biztosan teljesül, míg abban az esetben, ha a bináris változó értéke 1, az nem befolyásolja a jármű

azon koordinátájának értékét M megválasztásának köszönhetően:

$$\begin{aligned} & x_{ip} - x_{iq} \geq r_x - Mb_1 \\ \text{és} \quad & x_{iq} - x_{ip} \geq r_x - Mb_2 \\ \text{és} \quad & y_{ip} - y_{iq} \geq r_y - Mb_3 \\ \text{és} \quad & y_{iq} - y_{ip} \geq r_y - Mb_4 \\ \text{és} \quad & \sum_{k=1}^{4} b_k \leq 3 \end{aligned} \tag{2.12}$$

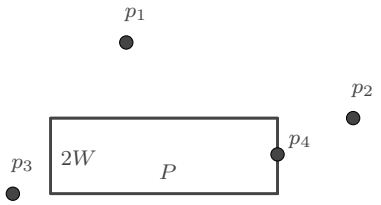
A (2.12) általánosított felírásában r_n jelöli az $n \in [1, \ldots, N]$ irányú biztonsági távolságot, melynél közelebb nem kerülhet egymáshoz két jármű, b_{ipqk} pedig az i időpillanatban p és q járművekhez tartozó bináris változó, ahol k az irányt jelöli. Ez egy p és q jármű-párra 2N+1 egyenlőtlenség felírását jelenti adott időpontban:

$$\begin{aligned} & \forall p, q \mid q > p, \forall i, \forall n \in [1, \ldots, N]: \\ & \qquad x_{ipn} - x_{iqn} \geq r_n - Mb_{ipqn} \\ & \text{és} \quad x_{iqn} - x_{ipn} \geq r_n - Mb_{ipq(n+N)} \\ & \text{és} \quad \sum_{k=1}^{2N} b_{ipqk} \leq 2N - 1 \end{aligned} \tag{2.13}$$

2.2.5. Csóvák elkerülése

Ebben a pontban folytatjuk a formalizálás bővítését, olyan egyenletekkel, melyekkel biztosíthatjuk annak megelőzését, hogy egy p jármű egy másik q járműhöz közelebb kerüljön a megengedettnél, mikor a q jármű éppen működteti a meghajtóit. Ekkor az egyik jármű csóvája veszélyeztetheti a többi járművet, ha a megadott távolságon belül találhatóak. Az egyszerűség kedvéért azt a területet, amelyet nem biztonságos megközelíteni ilyen esetben más járműveknek, téglalap alakú területnek fogjuk fel, R_q-val jelölöm, szélessége $2W$, és hossza a gyorsulás irányával ellentétesen P hosszú, az egyszerűség kedvéért a csóva méretét ennek a terület méretének tekintem. A korábbi pontokban látottakhoz hasonlóan először $N = 2$, majd az általános esetre írok fel egyenlőtlenségeket.

Az ilyen jellegű ütközések elkerülése kétféleképpen lehetséges: egyrészt, ha a járművek nem tartózkodnak egymáshoz elég közel ahhoz, hogy veszélyeztetve lehessenek más járművek csóváitól, másrészt pedig előfordulhat időpillanat, amikor egy jármű túlságosan közel kerül egy másik járműhöz, viszont azokban az időpillana-



2.2. ábra. p_1, p_2 és p_3 jármű biztonságos távolságban a p_4 jármű $2W$ szélességű és P hosszúságú csóvájától

tokban, amikor ez előfordul, nem történik a másik jármű részéről gyorsulás. Így a következő rendszer írható fel, ahol u_{xip} input a p jármű, i időpontban, X koordinátatengely mentén történő pozitív irányú gyorsulását jelöli:

$$\begin{aligned} & u_{xip} \leq 0 \\ \text{vagy} \quad & x_{ip} - x_{iq} \geq P \\ \text{vagy} \quad & x_{iq} - x_{ip} \geq 0 \\ \text{vagy} \quad & y_{ip} - y_{iq} \geq W \\ \text{vagy} \quad & y_{iq} - y_{ip} \geq W \end{aligned} \tag{2.14}$$

Azaz, ha az első sorban lévő egyenlőtlenség fennáll, azaz a p jármű gyorsulása az i időpillanatban az x irányban nem pozitív, azaz ebben a pillanatban nincsen csóvája negatív irányban, akkor más jármű áthaladhat R_p területén. Ha viszont az első sor nem teljesül, akkor a következő 4 sorban álló egyenlőtlenségek közül egynek legalább teljesülnie kell, azaz valamelyik irányban a q járműnek mindenképpen ki kell kerülnie az R_p területet. A (2.14) formával ekvivalens a következő bináris változókat tartalmozó felírás:

$$\begin{aligned} & u_{xip} \leq Mc_0 \\ \text{és} \quad & x_{ip} - x_{iq} \geq P - Mc_1 \\ \text{és} \quad & x_{iq} - x_{ip} \geq 0 - Mc_2 \\ \text{és} \quad & y_{ip} - y_{iq} \geq W - Mc_3 \\ \text{és} \quad & y_{iq} - y_{ip} \geq W - Mc_4 \\ \text{és} \quad & \sum_{k=0}^{4} c_k \leq 4 \end{aligned} \tag{2.15}$$

Ha $c_0 = 0$, akkor nem fenyegeti veszély a q járművet. Ha $c_0 = 1$, akkor az utolsó sor biztosítja, hogy a maradék négy bináris változó legalább egyike 0 értéket vegyen fel, ezáltal a q jármű kikerüli a veszélyes területet. (Természetesen c_1 a megoldás során összhangban lesz c_2-vel, például nem állhat fenn olyan eset, amikor mindkét változó 0 értéket vesz fel.)

A (2.15) általánosítása során jelölje $\mathbf{u_{ip}} = [u_{ip1}, \ldots, u_{ipN}]^T$ a p járműhöz az i időpontban tartozó gyorsulásvektort, ahol N a dimenziószám. Mivel egy koordináta mentén mindkét irányban történhet gyorsulás, így a vektorban lévő elemek felvehetnek egyaránt pozitív, negatív, illetve 0 értéket is, attól függően, hogy milyen irányban történik a jármű mozgása. Ezért az általánosítás során minden koordináta mentén kétszer kell felírni a megkötéseket, amit alább $+$, illetve $-$ felsőindexszel jelölök. Először a pozitív irányú gyorsulások megkötéseit írom fel. A felírást minden $p \neq q$ párra, minden i időpillanatra megcsináljuk, illetve minden koordináta mentén vizsgáljuk a gyorsulást. Ha a p jármű az n irányban végez pozitív irányú gyorsuló mozgást, akkor a negatív irányban P távolságra nem szabad más járműnek tartózkodnia tőle az adott pillanatokban, ettől eltérő koordináták mentén pedig pozitív és negatív irányokban egyaránt W ez a kritikus távolság. A bevezetett c^+_{ipqnm}, c^-_{ipqnm} bináris változók indexei rendre az alábbiakat jelölik: i időpillanat, p jármű, amelyhez az u_{ipn} érték tartozik, $q \neq p$ jármű, n az adott koordináta, melyhez u_{ipn} érték tartozik, $m \neq n$ koordináta. Ez alapján minden n-hez $2N+1$ darab c bináris változó tartozik.

$$
\begin{aligned}
&\forall p, q \mid q \neq p, \forall i, \forall n \in [1, \ldots, N]: \\
&\qquad u_{ipn} \leq M c^+_{ipqn0} \\
&\text{és} \quad x_{ipn} - x_{iqn} \geq P - M c^+_{ipqnn} \\
&\text{és} \quad x_{iqn} - x_{ipn} \geq -M c^+_{ipqn(n+N)} \\
&\text{és} \quad x_{ipm} - x_{iqm} \geq W - M c^+_{ipqnm} \quad \forall m \mid m \neq n \\
&\text{és} \quad x_{iqm} - x_{ipm} \geq W - M c^+_{ipqn(m+N)} \quad \forall m \mid m \neq n \\
&\text{és} \quad \sum_{k=0}^{2N} c^+_{ipqnk} \leq 2N
\end{aligned}
\tag{2.16}
$$

Hasonlóan írható fel a negatív irányba való gyorsulás esetére a kikötés:

$$
\begin{aligned}
&\forall p, q \mid q \neq p, \forall i, \forall n \in [1, \ldots, N]: \\
&\qquad -u_{ipn} \leq M c^{-}_{ipqn0} \\
&\text{és} \quad x_{ipn} - x_{iqn} \geq -M c^{-}_{ipqnn} \\
&\text{és} \quad x_{iqn} - x_{ipn} \geq P - M c^{-}_{ipqn(n+N)} \\
&\text{és} \quad x_{ipm} - x_{iqm} \geq W - M c^{-}_{ipqnm} \quad \forall m \mid m \neq n \\
&\text{és} \quad x_{iqm} - x_{ipm} \geq W - M c^{-}_{ipqn(m+N)} \quad \forall m \mid m \neq n \\
&\text{és} \quad \sum_{k=0}^{2N} c^{-}_{ipqnk} \leq 2N
\end{aligned}
\tag{2.17}
$$

Ebben a pontban még kitérünk arra is, hogy a különböző akadályokra is figyelnünk kell az egyes járművek gyorsulása esetén. Ennek nem szánok külön fejezetet, mivel ugyanazon megfontolás alapján írjuk fel a megkötéseket, mint abban az esetben, ahol két járművet kellett vizsgálni. Annyi különbséggel, hogy q jármű helyzetét leíró x_{iqm} helyett most az l téglalap alapú akadály U_{ln} és L_{ln} koordinátáit helyettesítjük a (2.17)-ben a megfelelő módon, ahol U_{ln} és L_{ln} jelöli az n koordináta szerinti maximális és minimális koordinátáit az akadálynak. Így minden l akadályra a következő két rendszert kell hozzávennünk a korábbiakhoz:

$$
\begin{aligned}
&\forall p, \forall l, \forall i, \forall n \in [1, \ldots, N]: \\
&\qquad u_{ipn} \leq M d^{+}_{ipln0} \\
&\text{és} \quad x_{ipn} - U_{ln} \geq P - M d^{+}_{iplnn} \\
&\text{és} \quad L_{ln} - x_{ipn} \geq -M d^{+}_{ipln(n+N)} \\
&\text{és} \quad x_{ipm} - U_{lm} \geq W - M d^{+}_{iplnm} \quad \forall m \mid m \neq n \\
&\text{és} \quad L_{lm} - x_{ipm} \geq W - M d^{+}_{ipln(m+N)} \quad \forall m \mid m \neq n \\
&\text{és} \quad \sum_{k=0}^{2N} d^{+}_{iplnk} \leq 2N
\end{aligned}
\tag{2.18}
$$

Illetve:

$$
\begin{aligned}
&\forall p, \forall l, \forall i, \forall n \in [1, \ldots, N]: \\
&\qquad -u_{ipn} \leq M d^{-}_{ipln0} \\
&\text{és} \quad x_{ipn} - U_{ln} \geq -M d^{-}_{iplnn} \\
&\text{és} \quad L_{ln} - x_{ipn} \geq P - M d^{-}_{ipln(n+N)} \\
&\text{és} \quad x_{ipm} - U_{lm} \geq W - M d^{-}_{iplnm} \quad \forall m \mid m \neq n \\
&\text{és} \quad L_{lm} - x_{ipm} \geq W - M d^{-}_{ipln(m+N)} \quad \forall m \mid m \neq n \\
&\text{és} \quad \sum_{k=0}^{2N} d^{-}_{iplnk} \leq 2N
\end{aligned}
\tag{2.19}
$$

2.2.6. Végső állapot elérése

Korábbi pontban említettem, hogy az egyszerűbb, de kötöttebb végső állapotkritériumnál megadhatunk egy rugalmasabb, valamivel bonyolultabb feltételt. Ehhez minden jármű esetén megadjuk a lehetséges állapotok halmazát a T időpontra, azaz a végső konfiguráció során több lehetséges pozíciója is lehet egy járműnek. A különböző járművek felállásainak lehetséges permutációival megkapjuk a lehetséges végső felállások G halmazát, melyből jelöljön g egy tetszőlegesen választott végső konfigurációt. Tehát g egy olyan elem, mely minden jármű egy adott végső helyzetét tartalmazza, azaz egy végső felállást pontosan egy g elem ír le.

Jelölje továbbá f_{pgr} azt a p járműhöz tartozó bináris változót, mely 1 értéket vesz fel akkor és csak akkor, ha a jármű az r pozíciót veszi fel a g végső konfiguráción belül, különben pedig 0 értéket.

A következőkben három kikötést teszünk fel. Egyrészt minden p jármű esetén

$$\sum_{g}\sum_{r} f_{pgr} = 1 \tag{2.20}$$

teljesül, hiszen egy jármű egyszerre csak egyetlen helyen tartózkodhat, illetve a konfigurációk közül is pontosan egy állhat fent. Másrészt ki kell kötnünk, hogy azok az f_{pgr} változók, melyek értéke 1, azonos g konfigurációhoz tartoznak, azaz minden g konfigurációra teljesül, hogy

$$\sum_{p}\sum_{r} f_{pgr} = V\sum_{r} f_{1gr} \tag{2.21}$$

ahol V a járművek száma. Ez az egyenlőség nyilván igaz, hiszen ha az első indexű járműre valamely g esetén $f_{1gr} = 1$, az azt jelenti, hogy a g konfiguráció következik be, és ekkor az ehhez a konfigurációhoz tartozó $\sum_{p}\sum_{r} f_{pgr} = V$, mivel minden jármű esetén pontosan egy r esetén lesz $f_{pgr} = 1$. Ellenkező konfigurációt tekintve ez az összeg 0 lesz, amit szintén biztosít az egyenlet, hiszen ekkor az összegben minden tag 0 értéket vesz fel abból adódan, hogy nem az adott konfiguráció áll fent. Harmadrészt pedig biztosítani szeretnénk, hogy egy r pozíciót egyszerre csak egy jármű vehessen fel, azaz minden r pozícióra fel kell tennünk, hogy

$$\sum_{p}\sum_{g} f_{pgr} = 1 \tag{2.22}$$

azaz pontosan egy konfigurációban egyetlen jármű esetén lesz $f_{pgr} = 1$.

Ezzel a három megkötéssel elértük, hogy minden jármű csakis egy helyen tartózkodhat, ugyanazon konfiguráción belül, illetve egy pozícióban csak egy jármű landolhat.

Minden p jármű koordinátáira nézve végezetül a következőt írjuk még fel:

$$\mathbf{x}_{Tp} = \sum_{g=1}^{G} \sum_{r=1}^{V} \mathbf{x}_r^g f_{pgr} \tag{2.23}$$

azaz a p jármű koordinátái a T időpillanatban megegyeznek a g végső konfigurációban felvett r pozíció koordinátáival, amit a fönti összegalakban írunk fel, ahol egyetlen g és r esetén lesz $f_{pgr} = 1$.

2.3. Próbálkozások a formalizált probléma egyszerűsítésére

Ebben a pontban röviden megemlítenék pár módszert arra, hogyan lehetséges a gyakorlatban a probléma megoldásának felgyorsításával próbálkozni. Ezek úgy működnek, hogy felhasználják a megadott adatokat, és ezek ismeretében szűkítenek az egyenletek, egyenlőtlenségek számán, amivel a megoldási időt lényegesen fel lehet gyorsítani. Két példát fogok röviden bemutatni ebben a témában.

A megkötések szűkítésének egyik módja, hogy megpróbáljuk megszűrni a csóvák kikerülésére felírt egyenlőtlenségeket. Legtöbb esetben a problémában a távolságok sokkal nagyobbak, mint az egyes járművek körül lévő terület, amit veszélyes lenne megközelíteni adott pillanatokban más járműveknek, akadályoknak. Ezért, ha adott pillanatokban biztosan túl nagy a távolság tőlük, feleslegessé válik bizonyos kikötések felírása. Másrészt megmutatható, hogy a költségminimalizálós problémákban az optimális megoldás úgy néz ki, hogy a röppálya elején, illetve végén történik a jármű üzemanyagfogyasztása, leszámítva az út során akadályok, egyéb járművek kikerülése esetén lévő fogyasztást, vagyis számos időpillanatban nem jelent veszélyforrást másokra. Így egy konkrét probléma megoldása előtt, felmérve az adott körülményeket, lehetőségünk van jelentősen redukálni a csóvák, akadályok kikerülése vagy ütközések céljából felírt megkötések számát, mellyel általában jóval gyorsabb megoldási időt tudunk elérni. Ekkor viszont szükséges utólag ellenőrizni az optimális megoldásról, hogy valóban nem sérti egyik kezdeti kikötésünket sem. Az ellenőrzés általában gyorsan elvégezhető, és ha nem találtunk ellentmondást, akkor az eredeti, bővebb LP feladat optimális megoldását is megkaptuk. Ez azért hasznos a gyakorlatban, mert egy megoldásról igazolni, hogy valóban optimális megoldás, polinomiális időben elvégezhető, míg az optimum megtalálása lényegesen hosszabb időt igényel. Ha a megoldásunk sérti valamelyik kikötést, akkor ebben az esetben a korábban eltávolított kikötések közül néhányat szükséges újra beleszámítani a felírásba, viszont ezzel a módszerrel általában még mindig egy gyorsabb eljárást kapunk, mintha az egész komplett feladatot kellene megoldanunk.

Azonban azokban az esetekben, mikor a járművek csóvájából adódó elkerülési terület nem kicsi a problémában előforduló többi távolsághoz képest, akkor az előző módszer nyilván nem alkalmazható. Viszont ebben az esetben is lehetséges bizonyos megkötések elhagyása, illetve pár új kikötés felállítása más megkötések elhagyásának érdekében. Általában egy jármű mikor elhalad egy akadály mellett, akkor ez hosszabb időintervallumot felölő manővert jelent, ebből kifolyólag ha egy i időpontban egy jármű egy kikerülendő akadály egyik oldalán tartózkodik, akkor elég valószínű, hogy a következő időpillanatban szintén azon az oldalon található. Ezáltal a járműhöz tartozó, egymást követő időpillanatokhoz tartozó bináris változókat csoportokba tudjuk osztani úgy, hogy az azonos csoportban lévő változók megegyezzenek. Azaz az azonos csoportokban lévő változók azt jelentik, hogy a megfelelő, eredeti *vagy* kikötések közül mindegyik a csoportból ugyanazt teljesíti. Az ezzel járó plusz kritériumokkal azért persze vigyáznunk kell, erre egy példa: vegyünk 3 egymást követő időpillanatot, tegyük fel hogy az ehhez tartozó bináris változók egy csoportban vannak, ugyanazt az értékeket veszik fel, illetve az eredeti, teljes problémában ezek közül az időpillanatok közül az elsőben még nem veszélyeztetné a jármű csóvája az akadályt, ugyanakkor a második és harmadik időpontban már igen. Ekkor a csoportosítás miatt mind a három időpillanatban tiltott a járműnek a tüzelőanyag égetés, helyette a megelőző vagy utána következő pillanatokban teheti azt meg. Ezzel viszont lehet, hogy kevésébé optimális megoldást kapunk. Mivel ezek a diszkrét időpillanatok elég közel vannak egymáshoz a problémába foglalt távolságokhoz képest, ezért pár pillanattal eltolva az üzemanyagfogyasztást, az még nem jelentősen befolyásolja a költséget. Ugyanakkor ezzel a módszerrel is jelentősen csökkenthetjük az eredeti formalizációban lévő megkötések számát.

3. fejezet

Implementáció példákkal

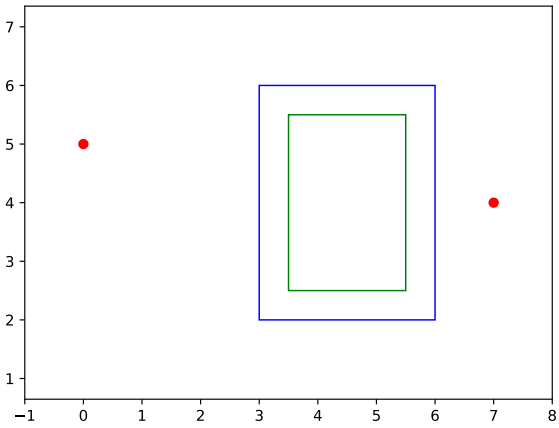
Ebben a fejezetben kétdimenziós példákon mutatom be az előző fejezetben tárgyalt modell pár változatát. Az optimalizációs feladatotokat Python nyelven implementálom és a PuLP könyvtárat telepítettem, amely egy nyílt forráskódú software, és olyan matematikai programok írására alkalmas, mint lineáris vagy kevert-egészrétékű lineáris programok modellezése. A Python nyelv ismerete miatt döntöttem a PuLP használata mellett.

3.1. Példa akadály elkerülésére

Első körben a következő példát tekintem: van egy járművünk, aminek el kell jutnia a $(0, 5)$ pontból a $(7, 4)$ pontba kikerülve az ütközést egy téglalap alakú akadállyal, és ennek az útnak az üzemanyagköltségét szeretnénk minimalizálni. Az akadály csúcsainak koordinátái legyenek: $(3.5, 2.5)$, $(3.5, 5.5)$, $(5.5, 2.5)$, $(5.5, 5.5)$ (A képen zöld téglalappal jelölt terület). A biztonság kedvéért ennél egy nagyobb területet tekintek a feladatmegoldás során elkerülendő területnek, legyenek ennek a nagyobb téglalapnak a csúcsai: $(3, 2)$, $(3, 6)$, $(6, 2)$, $(6, 6)$ (Ez az ábrán kék színnel látható téglalap). A MILP feladatunk megoldásához szükséges importálni a PuLP, és a kirajzoláshoz szükséges matplotlib csomagot.

```
from pulp import *
import matplotlib.pyplot as plt
import math
```

Szükségünk van megadni, hogy az optimalizálás során minimumot vagy maximumot keresünk-e. A következő kódrészlet első sorában olvasható, hogy a *model* elnevezésű MILP feladatunk során minimalizálni szeretnénk a majd későbbiekben definiált célfüggvényt. Az LP változóink deklarálásához szükséges még megadnunk az időpillanatok számát. Ez a kódban a *timestep* input, érdekes ennek változatásával megfigyelni, hogyan változik az optimum, illetve az optimális útvonal. Ezon kívül létrehozok három különböző vektort - *time*, *time*2, *time*3 - mivel lesznek egyes válto-



3.1. ábra. A feladat ábrája

zók, illetve megkötések, amik eltérő időintervallumokon vannak értelmezve. Továbbá a koordinátákra is szintén létrehozok két vektort, amiket a későbbiekben használni fogok.

```
model = pulp.LpProblem("Fuel cost minimising problem", pulp.
                                    LpMinimize)

time = []
time2 = [0]
time3 = [0]
timestep = 35
for i in range(1, timestep):
    time.append(i)
    time2.append(i)
    time3.append(i)
time2.append(timestep)
koord_4 = [1, 2, 3, 4]
koord_2 =[1, 2]
```

A következőkben fogom deklarálni az LP változókat. Minden változó esetén megadhatjuk, hogy az értékeket milyen korlátok között veheti fel, illetve hogy folytonos, egész, netán bináris változót szeretnénk deklarálni. A jármű helyzetét és sebességét az x változó tartalmazza, azaz $x[(i,j)] = \mathbf{x}_i[j]$ $(j = 1,2,3,4)$, ahol $\mathbf{x}_i = = (x_i, y_i, \dot{x}_i, \dot{y}_i)$. Deklarálnunk kell továbbá a gyorsulást mutató $u[(i,j)]$ változókat, melyek az i-ik időpillanatban a j-ik koordinátához tartozó értéket mutatják. Az akadály elkerüléséhez felírt bináris változókat tartalmazó kikötésekhez pedig szükségünk lesz az $a[(i,j)]$ bináris változókra, minden i időpillanathoz négy darabra. A változók értékei bizonyos korlátok között mozoghatnak csak, és ezeket a határokat már a változók deklarálása során megadhatjuk, illetve a program lehetőséget nyújt arra, hogy egy változóról feltegyük, hogy bináris változó.

```
x = pulp.LpVariable.dicts("x",
  ((i, j) for i in time2 for j in koord_4),
  lowBound=-100,
  upBound=100,
  cat='Continuous')

a = pulp.LpVariable.dicts("a",
  ((i, j) for i in time for j in koord_4),
  cat='Binary')

u = pulp.LpVariable.dicts("u",
  ((i, j) for i in time2 for j in koord_2),
  lowBound=-1,
  upBound=1,
  cat='Continuous')
```

Ezeken kívül még meg kell adnunk a kezdő és végső helyzetét a járműnek, és kikötjük, hogy az utolsó pillanatban az u változók értéke nulla.

```
model += x[(0, 1)] == 0
model += x[(0, 2)] == 5
model += x[(0, 3)] == 0
model += x[(0, 4)] == 0
model += x[(timestep, 1)] == 7
model += x[(timestep, 2)] == 4
model += x[(timestep, 3)] == 0
model += x[(timestep, 4)] == 0
model += u[(timestep, 1)] == 0
model += u[(timestep, 2)] == 0
```

A jármű dinamikáját felíró egyenletet már láttuk korábban. Most a (2.3)-nak megfelelő egyenleteket kell felírni minden $i = 0, 1, ..., timestep - 1$ időpontokra. Legyen $\alpha = 0{,}3$, azaz a két diszkrét időpont között eltelt idő.

```
c = math.exp(-0.3)
for i in time3:
    for j in koord_2:
        model += x[(i+1, j)] == x[(i, j)] + (1-c)*x[(i, j+2)] + (0.
                                            3-1+c)*u[(i, j)]
        model += x[(i+1, j+2)] == c*x[(i, j+2)] + (1-c)*u[(i, j)]
```

Ez az i szerinti ciklusonként négy plusz kikötést jelent.

A járműre vonatkozó feltételekkel ezzel végeztem, most jön az akadállyal való ütközés elkerülésére vonatkozó kritérium. $M = 2000$ esetén biztosan kellően nagy számot választottam minden feladatbeli távolsághoz viszonyítva. A (2.9) pontban lévő kikötés felírását mutatja a következő kódrészlet:

```
for i in time:
    model += x[(i, 1)] <= 3 + 2000*a[(i, 1)]
```

```
    model += -x[(i, 1)] <= -6 + 2000*a[(i, 2)]
    model += x[(i, 2)] <= 2 + 2000*a[(i, 3)]
    model += -x[(i, 2)] <= -6 + 2000*a[(i, 4)]
    model += pulp.lpSum([a[i, k] for k in koord_4]) <= 3
```

Ezt követően felírhatjuk a célfüggvényünket, aminek az optimumát keressük. Mivel a solver nem tudja kezelni LP válozók abszolútértékét, így nem használható egyszerűen a

$$\min \sum_{i=0}^{T-1} \sum_{p=1}^{V} \sum_{n=1}^{N} |u_{ipn}| \tag{3.1}$$

célfügvény. Ennek megoldására újabb változók bevezetése szükséges, ugyanis az u változó felvehet egyaránt pozitív és negatív értéket is. Így minden $u[(i,j)]$ változóhoz definiálunk egy $t[(i,j)]$ változót, melyre a következőt kötjük ki:

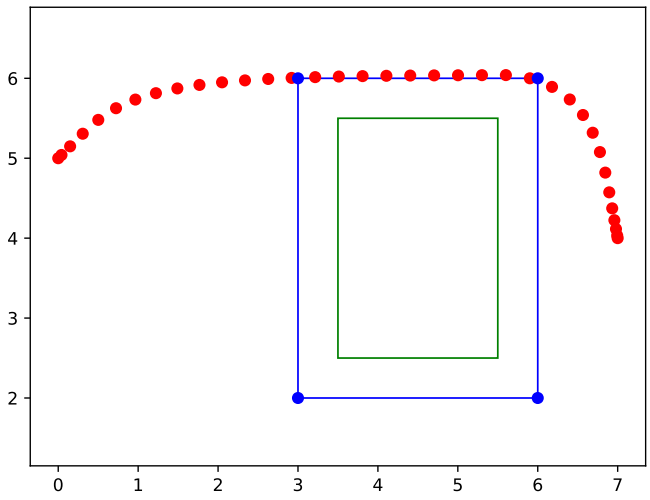
$$\begin{aligned} t[(i,j)] &\geq u[(i,j)] \\ t[(i,j)] &\geq -u[(i,j)] \end{aligned} \tag{3.2}$$

Azaz a t LP változók minden (i,j) párra nem kisebbek u abszolútértékénél. Az optimalizációs feladat célfüggvényét a t LP változók összegével határozom meg, ennek a minimumát keresi a solver. Mivel a t változók nem kisebbek $|u|$ értékénél, ugyanakkor minimalizálja a program ezek értékét, így az optimális eredmény pontosan meg fog egyezni a 3.1 optimális értékével.

```
t = pulp.LpVariable.dicts("t",
  ((i, j) for i in time3 for j in koord_2),
  lowBound=0,
  upBound=1,
  cat='Continuous')

for i in time3:
    for j in koord_2:
        model += t[(i, j)] >= u[(i, j)]
        model += t[(i, j)] >= -u[(i, j)]

model += (
    pulp.lpSum([
        pulp.lpSum([
            t[(i, j)]
            for j in koord_2])
        for i in time3])
)

model.solve()
```

Ezzel a MILP feladat felírása teljes, a solver a branch-and-bound algoritmus segítségével számolja ki az optimális megoldást, ebben a konkrét esetben az optimális

értékhez tartozó optimális útvonal a következő ábrán látható piros színnel. A bináris változók száma minden időpillanatban 4, így nagyságrendileg $4 \cdot timestep$ darab bináris változót használunk fel. A kód pár másodperces futási idő alatt megtalálja az optimális útvonalat.



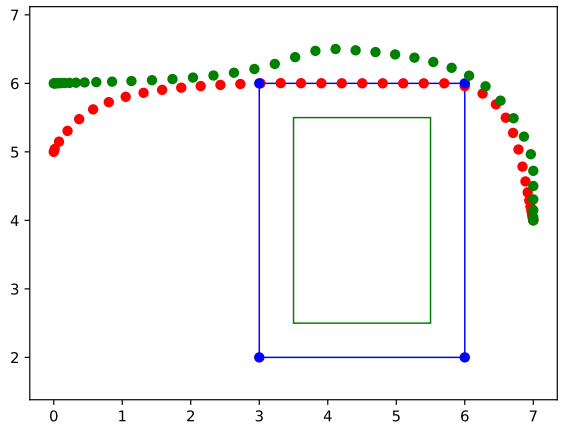
3.2. ábra. Az optimális útvonal $timestep = 35$ esetén

3.2. Akadály kikerülése két jármű esetén

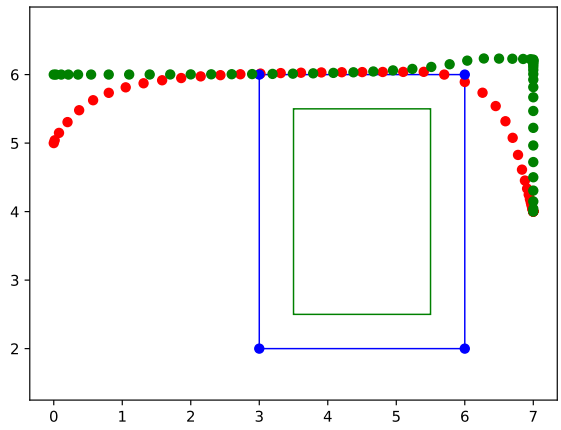
Az előző pontban felírt MILP feladatot bővítem ki még egy jármű hozzávételével, mely a $(7, 4)$ pontból indul, és a $(0, 6)$ pontba érkezik. Erre a járműre szintén fel kell írni ugyanazokat a kikötéseket, mint a másik járműre az előző pontban felírtakat. A célfüggvényünk most a két jármű összesített üzemanyagfogyasztása lesz. Továbbá figyelnünk kell a járművek ütközésének elkerülésére, ami az alábbi kódrészletben látható, ahol $x[i, j]$ az első, míg $y[(i, j)]$ a második jármű megfelelő koordinátáit jelöli $j = 1{,}2$ esetén, és a $d[(i, j)]$ változók bináris változók. Az egyszerűség kedvéért attól függetlenül, hogy éppen történik-e üzemanyagfogyasztás vagy sem, egy állandó biztonsági távolságot veszek fel mind a két koordináta mentén, melynél közelebb nem közelíthetik meg egymást a járművek. Ezt a távolságot 0,5-nek rögzítem.

```
for i in time:
    model += x[(i, 1)] - y[(i, 1)] >= 0.5 - 2000 * d[(i, 1)]
    model += y[(i, 1)] - x[(i, 1)] >= 0.5 - 2000 * d[(i, 2)]
    model += x[(i, 2)] - y[(i, 2)] >= 0.5 - 2000 * d[(i, 3)]
    model += y[(i, 2)] - x[(i, 2)] >= 0.5 - 2000 * d[(i, 4)]
    model += pulp.lpSum([d[i, k] for k in koord_4]) <= 3
```

A kódot futtatom $timestep = 40$ és $timestep = 50$ esetekre is. Az ábrákon látható, hogy mennyire különböző optimális útvonalakat kapunk. A második esetben kapott útvonal alakja annak köszönhető, hogy kellő időpillanat áll rendelkezésre a jármű fékezésére a kanyar előtt. A fékezés, a dinamikai rendszernek köszönhetően, nem kerül költségünkbe. Az előző esetben az optimális útvonalhoz tartozó üzemanyagköltség 67,8, míg az utóbbi esetben ez az érték 66,5, meglepő módon ugyanakkor a második esetben a futási idő sokszorosa az első kód futási idejének.



3.3. ábra. Az optimális útvonal két jármű és $timestep = 40$ esetén, a költségfüggvény optimuma 67,8



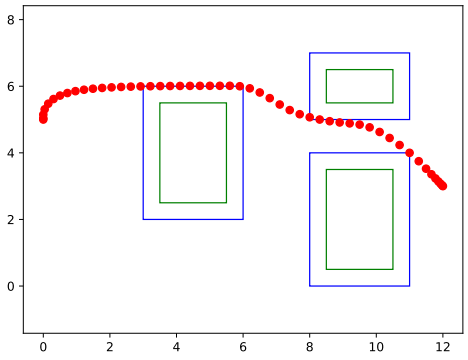
3.4. ábra. Az optimális útvonal két jármű és $timestep = 50$ esetén, a költségfüggvény optimuma 66,5

Annak ellenőrzésére, hogy valóban nem történt ütközés, a consolra kiíratom annak a vektornak az elemeit, amik az összes irányból kiválasztják a két jármű koordinátáinak különbségéből a legnagyobb távolságot. Az ütközés elkerülése érdekében ennek a vektornak minden elemének nagyobbnak vagy egyenlőnek kell lennie 0,5-nél.

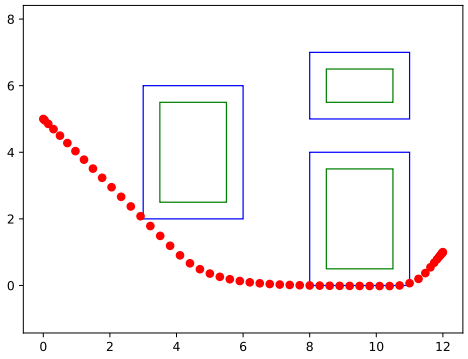
3.3. További példák

Ebben a fejezetben szeretnék még pár példát mutatni az optimális útvonalra különböző esetekben. A járművek dinamikáját továbbra is ugyanaz a rendszer írja le. Az akadályok zöld színű téglalapokkal vannak jelölve, a körülöttük lévő kék téglalapok jelölik ki azt a határt, ahol a jármű tartózkodása még biztonságos.

A következőkben három akadályt veszek fel, és különböző végállapotok mellett futtatom le a kódot. A 3.5 ábrán a jármű a $(0, 5)$ pontból a $(12, 3)$ pontba, a 3.6 ábrán pedig ugyanebből a pontból kiindulva a $(12, 1)$ pontba érkezik.

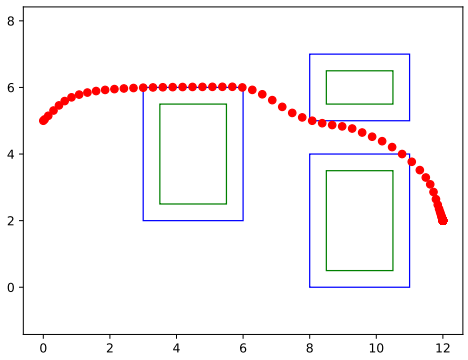


3.5. ábra. Az optimális útvonal három akadály, $timestep = 50$ és $(12, 3)$ cél esetén

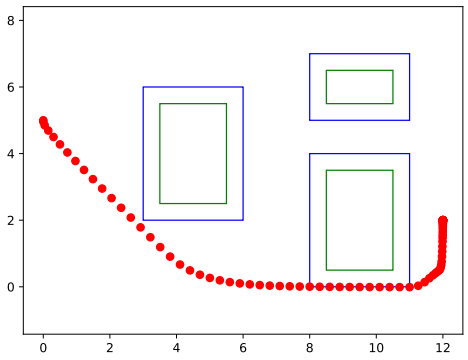


3.6. ábra. Az optimális útvonal három akadály, $timestep = 50$ és $(12, 1)$ cél esetén

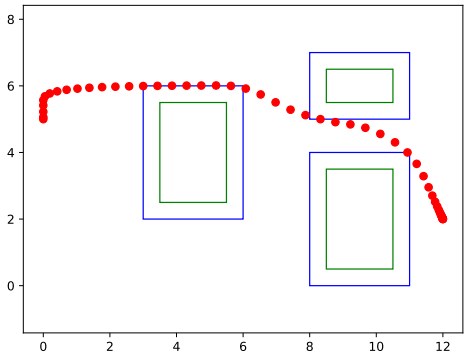
Ugyanakkor $(12, 2)$ végpont esetén a solver a kód lefutása után azt írja, hogy megoldhatatlan a MILP feladatunk. Ez lehetséges, ha például nem elég nagy a *timestep* változónk, mivel az u változónkra korlátot adtunk meg, és ezért a dinamikát leíró egyenleteknek nincsen megoldása. Tehát ekkor megadhatunk egy nagyobb értéket az időpontok számának. Ugyanakkor az u felső korlátjának kiterjesztésével is orvosolható a probléma. Különböző adatok változtatása esetén az optimális útvonalak igen eltérőek lehetnek, ezt láthatjuk a következő ábrákon.



3.7. ábra. Az optimális útvonal három akadály, $timestep = 100$ és $(12, 2)$ cél esetén



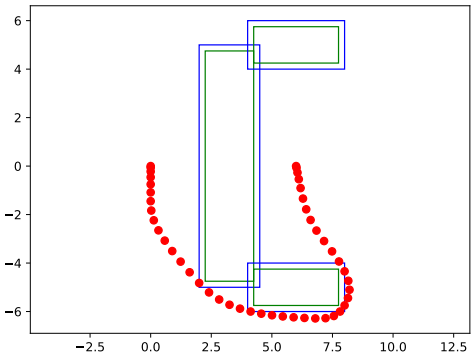
3.8. ábra. Az optimális útvonal három akadály, $timestep = 90$ és $(12, 2)$ cél esetén



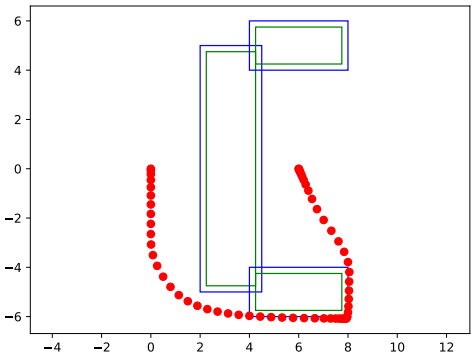
3.9. ábra. Az optimális útvonal három akadály, $timestep = 50$, $upBound(u) = 1.5$ és $(12, 2)$ cél esetén

Egy másik elrendezése a három akadálynak és a hozzájuk tartozó optimális útvonalak a 3.10 és a 3.11 ábrákon látható $timestep = 45$ és $timestep = 60$ esetén. A költségfüggvény optimális értéke az első esetben 83,25, míg a második esetben 75,04,

ugyanakkor mind a két kód pár percen belüli futási idővel megtalálja az optimális megoldást.



3.10. ábra. Az optimális útvonal három akadály, *timestep* = 45 esetén, a költségfüggvény optimuma 83,25



3.11. ábra. Az optimális útvonal három akadály, *timestep* = 60 esetén, a költségfüggvény optimuma 75,04

Irodalomjegyzék

[1] Frank András, Király Tamás, *Operációkutatás* (2013)

[2] Lovász László, *Algoritmusok Bonyolultsága* (2014)

[3] http://compalg.inf.elte.hu/∼tony/Oktatas/Osztott-algoritmusok/P-NP-NPC.pdf

[4] http://www.cs.cmu.edu/afs/cs/academic/class/15451-s10/www/recitations/rec0408.txt

[5] Arthur Richards, Tom Schouwenaars, Jonathan P. How, Eric Feron, *Spacecraft Trajectory Planning with Avoidance Constraints Using Mixed-Integer Linear Programming*, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, Journal of Guidance, Control, and Dynamics (2002)

[6] Matthew G. Earl and Raffaello D'Andrea, *Multi-vehicle Cooperative Control Using Mixed Integer Linear Programming*, In Cooperative Control of Distributed Multi-Agent Systems, J. S. Shamma ed., John Wiley & Sons (2007)

[7] Tamás Kalmár-Nagy, Raffaello D'Andrea, Pritam Ganguly, *Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle*, Robotics and Autonomous Systems 46 (2004)

Ábrák jegyzéke