Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

Limpek Márton Bertalan

BELTÉRI MOBIL RÁDIÓHÁLÓZATOK OPTIMALIZÁLÁSA

KONZULENS

Dr. Nagy Lajos

BUDAPEST, 2021

Tartalomjegyzék

Összefoglaló

A telekommunikációs szektorban napjaink legnagyobb technológiai újdonsága az ötödik generációs (5G) mobil hálózatok. Ezen hálózatok, mind maghálózati, mind a rádiós hozzáférési hálózatban különböznek a korábbi generációs mobilhálózatoktól. Tervezés során ezeket az eltéréseket szükséges figyelembe venni. A mobil rádióhálózatok fejlődésével a pikocellák kialakítása különösen fontos kutatási és rádióhálózat tervezési feladat. A TDK dolgozatban a mobilhálózatok – különösen ötödik generációs mobilhálózatok – rádiós hozzáférési hálózatának tervezésével foglalkozom, beltéri környezetben.

A dolgozatban bemutatásra és összehasonlításra kerülnek a második, harmadik, negyedik és ötödik generációs mobil rádiós hozzáférési hálózatok teljesítmény mérleg és dimenzionálás számításai. Egy ilyen hálózat tervezésekor célszerű első lépésként ezen számításokat elvégezni.

Ezek után a legfontosabb empirikus és félempirikus beltéri hullámterjedési modelleket (Motley-Keenan modell, egyszeres meredekségű modell [One Slope Model], kettős meredekségű modell [Dual Slope Model], lineáris csillapítású modell [Linear Attenuation Model], particionált modell [Partitioned Model], ITU [ITU-R P.1238 Indoor Model] modell) összevetem a determinisztikus sugárút keresés alapú terjedési modellel a 700MHz-4GHz frekvenciasávban. Az elméleti vizsgálatok mérésekkel kerülnek alátámasztásra. A vizsgálatok alapján a cél javaslatot tenni félempirikus beltéri hullámterjedési modell alkalmazására a 3.6GHz-es frekvenciasávban.

Tanulmányozásra kerül továbbá a Genetikus Algoritmuson alapuló optimalizálás alkalmazhatósága. Az alkalmazhatóság lefedettség és forgalom szempontokból vizsgálom meg, optimális beltéri mobil rádiós hozzáférési hálózat kialakítása céljából.

Abstract

In the telecommunication sector the greatest technological invention today is 5G mobile networks. These networks, differ from the previous mobile network generations both in Core Network (CN) and in Radio Access Network (RAN). These differences should be taken into consideration in the planning phase. Parallel with the development of mobile radio networks the creation of picocells is a particularly important research and radio network planning task. In this TDK paper I deal with the planning of Radio Access Networks - especially 5G mobile networks - in indoor environment.

In the paper the power balance and dimensioning calculations of second, third, fourth, fifth generation mobile radio access networks are shown and compared. While planning such a network it is advisable to do these calculations as the first step.

After that I compare the most important empirical and semi-empirical indoor wave propagation models (Motley-Keenan model, One Slope Model, Dual Slope Model, Linear Attenuation Model, Partitioned Model, ITU-R P.1238 Indoor Model) to the deterministic ray tracing-based propagation model in the 700MHz-4GHz frequency range. The theoretical examinations are supported by measurements. Based on the examinations the aim is to make a proposal for the application of indoor wave propagation model in the 3.6GHz frequency range.

In addition, the applicability of the optimization based on Genetic Algorithm (GA) is studied. The applicability will be examined in terms of coverage and network traffic in order to create optimal indoor mobile radio access network.

1 Bevezetés

Feladatként beltéri mobil rádióhálózatok optimalizálását választottam. Napjaink legnagyobb technológiai újdonsága az ötödik generációs (5G) mobil hálózatok. Ezek a mobilhálózatok sok szempontból különböznek a korábbi mobilhálózat generációktól. A legnagyobb különbség talán az, hogy a hálózat nagyon flexibilis a „network slicing”-nak köszönhetően, azaz a hálózat egyszerre több felhasználói igénynek is meg tud felelni.

Hogy ezt elérjék, a maghálózaton (Core Network, CN) és a rádiós hozzáférési hálózaton (Radio Access Network, RAN) is kellett változtatásokat végezni a korábbi generációkhoz képest.

A feladat elvégzésének első lépését irodalomkutatással kezdtem, ahol az optimalizálás/beltéri rádióhálózat tervezéséhez szükséges alapokat vizsgáltam meg. Ezek után modellező szoftver segítségével beltéri modelleket készítettem, majd különböző eseteket szimuláltam. A szimulációkból nyert adatokat matlab segítségével tovább elemeztem. A szimulációk mellett Java nyelven implementáltam egy szoftvert, amivel optimális megoldást keresek az adóantennák elhelyezésére egy adott beltéri területen.

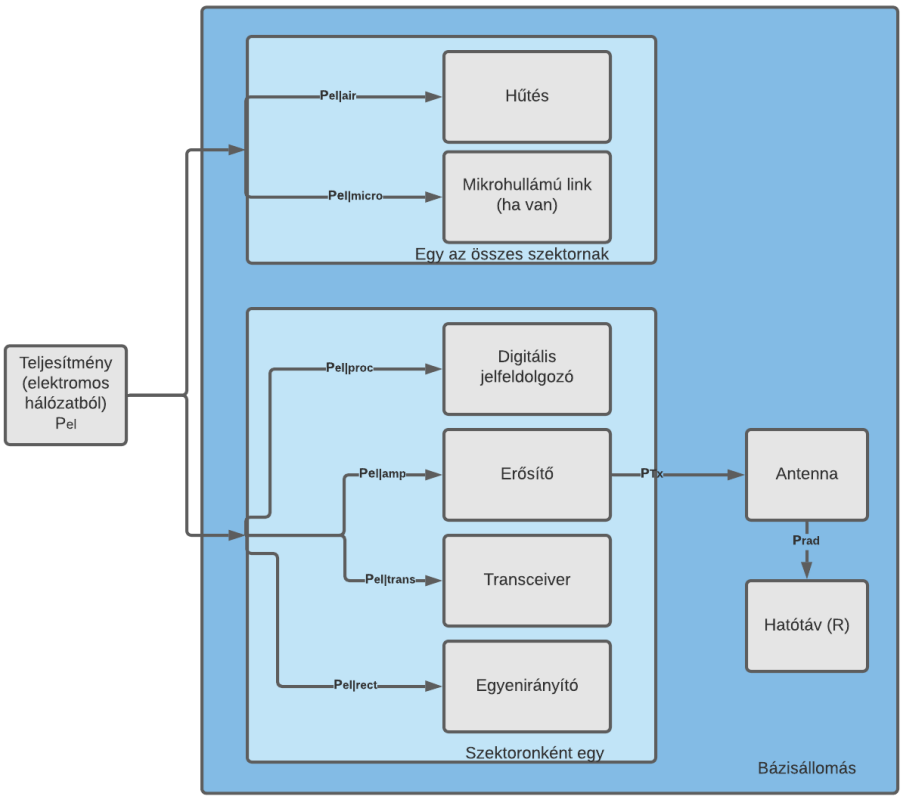
2 Dimenzionálás

A rádiós hozzáférési hálózat tervezése több lépésből áll. Amennyiben ilyen feladatunk van, úgy az előkészületek után az előzetes tervezést szükséges elvégezni. Az előzetes tervezéshez tartozik a rádiós hálózat dimenzionálása. A dimenzionálás egy módszer, melynek bemenetei között szerepelnek lefedettséggel, kapacitással és minőséggel összefüggésben álló modellek/követelmények. Ezeken felül a rádiós hozzáférési hálózatot minden szempontból költséghatékonyan szükséges létrehozni, amely költségeket már az előzetes tervezés során szükséges figyelembe venni.

Mivel manapság az összes megtermelt energia egy jelentős részét a világon az ICT rendszerek használják fel, így környezetvédelmi célokból is érdemes olyan implementációkra törekedni, amelyek minél optimálisabbak energiahatékonyság szempontjából. Ez az elkövetkező években még kritikusabb kérdés lesz, ugyanis a trendek azt mutatják, hogy egyre több helyen használják az ilyen fajta rendszereket. A bázisállomások optimális elhelyezésével tehát környezetvédelmi szempontból is megfelelően járunk el.

2.1 Energiafogyasztás és lefedettségi modell

A bázis állomást definiáljuk úgy, hogy egy eszköz, ami képes kommunikálni az egyes mobil eszközökkel és a gerinchálózattal. Egy ilyen bázisállomás több energiafogyasztó komponensből áll. A területet, amit egy bázisállomás lefed cellának nevezzük. A cellák szektorokra vannak osztva, amiket egy-egy szektorantenna fed le. Amennyiben egy szektorantenna energiafogyasztását meghatározzuk, ezt a szektorok számával megszorozva kaphatjuk meg a bázisállomás energiafogyasztását. A teljesítmény fogyasztó komponenseket ábrázoltam és jelöléssel láttam el.



ábra 1: Teljesítményfogyasztók a bázisállomáson

Ezek alapján P_{el}-t a következő képlettel számíthatjuk ki:

$$P_{el} = n_{sector} \times (n_{TX} \times (P_{el|amp} + P_{el|trans}) + P_{el|proc} + P_{el|rect}) + P_{el|micro} + P_{el|air}$$

, ahol n_{sector} a szektorok száma, n_{TX} az egy szektorban lévő adóantennák száma. Az eredmény Watt mértékegységben kapjuk meg. A képlet akkor igaz, ha egy szektorban egy frekvenciát használunk fel. UMTS, HSPA és LTE mobilhálózatok esetén, ha 3 szektoros a bázisállomás, szektoronként egy antennával, akkor P_{el} értéke 1700W és 1500W tartományba esik.

Ezek után számítsuk ki az R lefedési távolságot. Ezt link költségvetés (link budget) számítással tehetjük meg, amely számításnál minden erősítést illetve veszteséget figyelembe veszünk. Első lépésként a maximális megengedett szakaszcsillapítást szükséges kiszámolni (PL_{max}, [dB]). Ezen érték kiszámításához több paramétert szükséges figyelembe venni, köztük a frekvenciát, adóteljesítményt, valamint az antennák típusát is. Amennyiben PL_{max} -ot meg meghatároztuk, a maximális távolságot (méterben) amit a bázisállomással le tudunk fedni az alábbi képletből kapjuk meg:

$$R = g^{-1}((PL_{max} - SM)|f, h_{BS}, h_{MS})$$

, ahol PL_{max} a maximálisan megengedhető szakaszcsillapítás [dB], SM az árnyékolás [dB], f a frekvencia [Hz], h_{BS} a bázisállomás magasága [m], h_{MS} a mobil antennájának magassága a földtől [m]. A g(.) függvény a használt terjedési modelltől függ. (1)

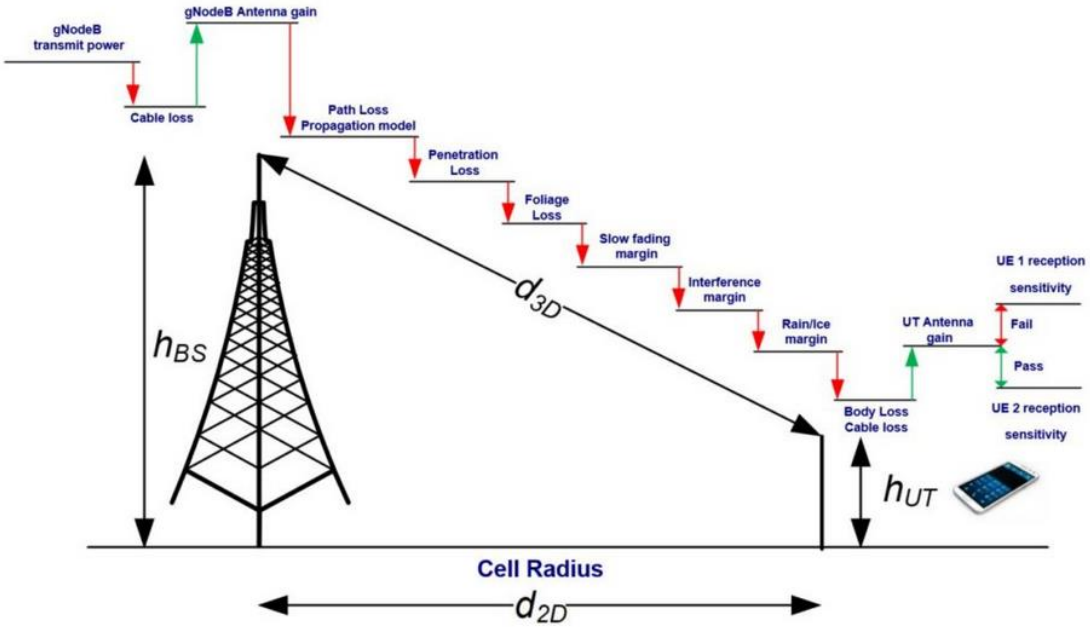
Esetünkben a PL_{max} értéket a mellékelt forrásból olvastam ki GSM, HSPA és LTE mobilhálózatokra vonatkozóan. Amennyiben az uplink esetet tekintjük, ezek értéke rendre 162, 161.6 és 163.4, a mértékegység minden esetben dB. Downlink irányban ezen értékek nagysága 161.5, 163.4, 163.5 [dB]. Esetünkben a g(.) függvényként az egyszeres meredekségű modellhez tartozó függvény választottam, mivel a későbbi számításoknál pontosnak bizonyult, és a számítást egyszerűen lehet vele elvégezni. A számításokat 3.6GHz-re végeztem el. Beltéri környezetben az SM értékét a sok mozgó személyek és objektumok miatt nagyra választottam (SM = 20dB).

RAN Technology	**GSM (2G)**	**HSPA (3G)**	**LTE (4G)**
Adatsebesség (kbps) - Uplink	**12.2**	**64**	**64**
Maximális szakaszcsillapítás	**162**	**161.6**	**163.4**
R [m]	**991.24**	**963.96**	**1093**
Adatsebesség (kbps) - Downlink	**12.2**	**1024**	**1024**
Maximális szakaszcsillapítás	**161.5**	**163.4**	**163.5**
R [m]	**957.26**	**1093**	**1100**

Az R értékekre igen nagy értékeket kaptam, ezért feltételezem, hogy a link budget értékek kültéri lefedettség esetén voltak érvényesek és beltéri esetben ezektől kisebb értékeket kapnánk.

5G hálózat esetére nem állt rendelkezésre a maximális szakaszcsillapítás, így azokat is nekem szükséges meghatározni. A maximális szakaszcsillapítást és ennek

segítségével a cellasugarat az 2. ábra szerinti modell segítségével, egy online elérhető ’5G NR Link budget calculator’-al kerül majd kiszámításra, azonban jelenleg még nincs információnk az adó konkrét paramétereiről, így ezen számításokat a dolgozat írásakor nem tudtam elvégezni.



ábra 2: Link költségvetés (link budget) és cellasugár számítása

A link költségvetés számításához felhasznált online szoftvernek a 3GPP 38.901 szabvány az alapja.

3 Hullámterjedési modellek

Ahhoz, hogy egy bizonyos területre tervezni tudjunk rádiós hozzáférést, szükségünk van egy hullámterjedési modellre. Ezen modellek elsődleges feladata rádióhálózatok tervezésénél, a forrás antenna, illetve interferáló egyéb források által generált teljesítményszintek vagy az adótól érkező jel csillapodásának becslése a vételi pontokban. Jelenleg nincs olyan, hogy 'legjobb modell', az egyes hullámterjedési modelleknek vannak előnyeik és hátrányaik, amiket figyelembevéve kell eldönteni, hogy egy adott helyzetben milyen modellt érdemes használni. (2) (3)

Általánosságban igaz, hogy egy modell minél pontosabb, annál bonyolultabb és számításigényesebb. A kevésbé számításigényes modellek pontatlanabbak, de sok esetben nincs szükségünk akkora pontosságra, mint amit a bonyolultabb modellel kaphatnánk.

A térerősség becslésére empirikus, félempirikus, valamint determinisztikus modelleket alkalmazhatunk.

3.1 Empirikus és Félempirikus modellek

Az empirikus terjedési modelleket nagy számú mérés alapján dolgozták ki: a mérési eredményekre illesztettek paraméteres egyenleteket. A kívánt eredmény – ami többnyire a szakaszcsillapítás – könnyen és gyorsan meghatározható, viszont nem kapunk nagyon pontos értéket. (2)

A félempirikus modellek empirikus és determinisztikus részekből tevődnek össze, a kettő keveréke.

Amennyiben mobil lefedettséget tervezünk, az úgynevezett cellás elvet alkalmazzuk. Egy cella alatt az egy adóval lefedett területet értjük. A cellák mérete és alakja különféle lehet. Beltéri tervezés esetén pikocellákról beszélhetünk. Az épületek belsejében számos paramétert figyelembe kell venni a hullámterjedés szemszögéből. A falak jelenléte, valamint az épületben tartózkodók mozgása miatt egy időben változó többutas terjedés alakul ki. (4)

A vett teljesítményt a vételi pontban általánosan az alábbi képletből olvashatjuk ki, ahol P_T az adó által kisugárzott teljesítmény, G_T illetve G_R az adó, illetve vevő

antennák nyereségei és L_P a szakaszcsillapítás, amit a különböző modellek esetén máshogy számítunk ki (5):

$$P_R|_{dBm} = P_T|_{dBm} + G_T|_{dBm} + G_R|_{dBm} - L_P|_{dB}$$

3.1.1 Motley-Keenan modell

Ez a modell talán a legegyszerűbb az itt felsorolt modellek közül, így a legpontatlanabb is, de könnyű implementálhatósága miatt az optimalizáló szoftverben is ezt a modellt alkalmaztam.

A modellt felhasználva $L_P|_{dB}$ -t a következőképpen számíthatjuk ki:

$$L_{P_K}|_{dB} = L_1|_{dB} + 20\log_{10}(d) + \sum n_f a_f|_{dB} + \sum n_w a_w|_{dB}$$

, ahol $L_1|_{dB}$ az adótól 1 méterre lévő szabadtéri csillapítás, amit a következő képlettel számolunk:

$$L_1|_{dB} = 20\log_{10}\left(\frac{4\pi d_0}{\lambda}\right), ahol\ d_0 = 1méter$$

A képletben d az adó és a vevőpont közötti távolság, a_f és a_w a födémek, illetve falak csillapításai, valamint n_f és n_w az adó és a vevőpontot összekötő egyenes szakaszt metsző födémek és falak száma. (6)

3.1.2 Egyszeres meredekségű modell (One slope)

Ez a modell a γ paraméter segítségével alkalmazkodik a környezet jellemzőihez, az emeletek és a falak csillapításait ezen a paraméterén keresztül veszi figyelembe.

$$L_{P\_OSM}|_{dB} = 20\log_{10}(f) + \gamma\log_{10}(d) - 28$$

, ahol f az antenna által kisugárzott jel frekvenciája MHz-ben, d az adó és a vevő távolsága. Beltéri irodai környezetben (mint az általam vizsgált terület) γ értéke 33. A frekvenciát, azaz f értékét MHz-ben szükséges megadni. (5)

3.1.3 Kettős meredekségű modell (Dual slope)

A hullámterjedést beltéri környezetben az alapján kategorizálja be, hogy az első Fresnel-zóna ’tiszta’-e. Amennyiben az első Fresnel-zónában nincs akadály és az útveszteség kitevője kisebb mint 2 akkor adó közeli terjedésről beszélünk. Töréspontos

terjedésről beszélünk, ha az első Fresnel-zónában van akadály és az útveszteség kitevője nagyobb mint 2. A vevő által vett teljesítményt a következő képlettel lehet számolni:

$$P_R = P_T - 10\begin{cases} n_1 \log_{10}(d) & , \quad d < d_{bp} \\ n_1 \log_{10}(d_{bp}) + n_2 \log_{10}\left(\frac{d}{d_{bp}}\right), & d > d_{bp} \end{cases}$$

, ahol n_1 és n_2 a szakaszcsillapítás kitevői és d_{bp} a töréspontí távolság. (7)

3.1.4 Lineáris csillapítású modell

Lineáris csillapítású modell esetén a vett teljesítmény a vevőpontban:

$$P_R|_{dB} = P_T|_{dB} - 20\log_{10}(d) - a \cdot d$$

, ahol d az adó és a vevő közötti távolság, a pedig a veszteségi tényező (*0.3 ... 0.6dB/m*). (7)

3.1.5 Partícionált modell

Ebben a modellben a szakaszcsillapítást egy előre meghatározott m értékkel és a távolsággal becsüljük az alábbi képlet alapján (7):

$$P_R = P_T - \begin{cases} 20\log_{10}(d), & m < d \leq 10m \\ 20 + 30\log_{10}\left(\frac{d}{10}\right), & 10m < d \leq 20m \\ 29 + 60\log_{10}\left(\frac{d}{20}\right), & 20m < d \leq 40m \\ 47 + 120\log_{10}\left(\frac{d}{40}\right), & d > 40m \end{cases}$$

3.1.6 ITU-R P.1238 beltéri modell

Az alábbi modell akkor alkalmazható, ha az adó és a vevő ugyan azon az emeleten helyezkedik el.

$$L_{P\_ITU}|_{dB} = 10\alpha \log_{10}(d) + \beta + 10\gamma \log_{10}(f)$$

, ahol d az adó és a vevő közötti távolság, f az üzemi frekvencia – GHz-ben megadva. Az α, β, illetve γ együtthatóknak nincs fizikai jelentése, az értékük az alábbi táblázatból olvasható ki. (8)

Környezet	LoS/NLoS	Frekvenciatartomány (GHz)	Távolságtartomány (m)	α	β	γ
Iroda	LoS	0.3-83.5	2-27	1.46	34.62	2.03
	NLoS	0.3-82.0	4-30	2.46	29.53	2.38
Folyosó	LoS	0.3-83.5	2-160	1.63	28.12	2.25
	NLoS	0.625-83.5	4-94	2.77	29.27	2.48
Ipar	LoS	0.625-70.28	2-101	2.31	24.52	2.06
	NLoS	0.625-70.28	5-108	3.79	21.01	1.34

3.2 Determinisztikus modellek

A determinisztikus modellek sugárkövetésen vagy Maxwell-egyenletek közvetlen megoldásán alapszanak. A sugárkövetéses megoldás egy tisztán geometriai feladat megoldását jelenti, ennek ellenére bizonyos esetekben csak nehézkesen alkalmazható, mivel a tervezett területen lévő objektumok alakja, anyaga és orientációjától függően – már amennyiben képesek vagyunk a környezet pontos felvételére egy adatbázisban – nagyon számításigényes lehet az eljárás. A Maxwell-egyenletek közvetlen megoldására parabolikus típusú egyenletek alkalmasak, vagy az egyenletek véges differenciális alakú időtartományú megoldása. (6)

3.2.1 Sugárkövetésen alapuló modell (Ray tracing)

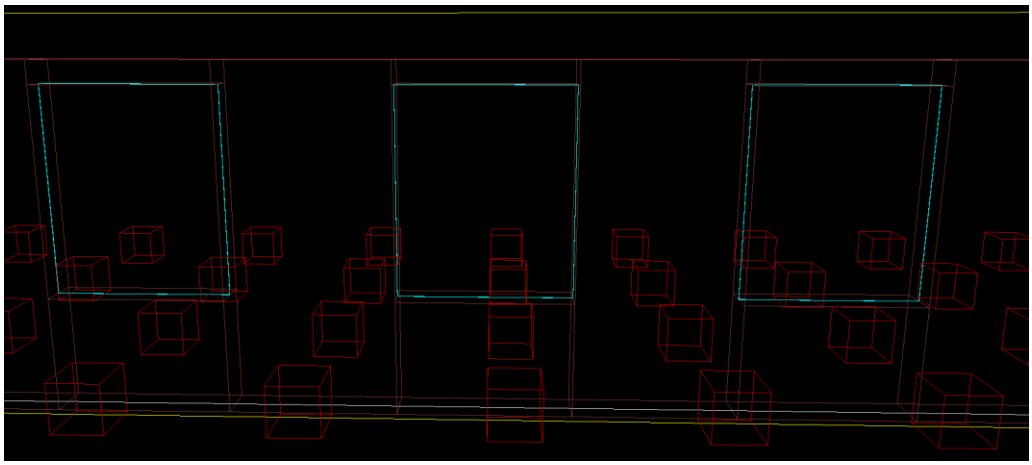
A sugárkövetés elvű hullámterjedési modellek a teljes tartományú térmodellezés helyett a geometriai optikán alapulnak. A terjedő hullámokat véges térszögtartományokra bontva, az ezeken terjedő komponenseket függetlenül kezelve és a határfelületeken fellépő jelenségeket – reflexió, transzmisszió, diffrakció – érvényesítve a teljes megoldást ezen összetevők egyes vizsgálati pontokban kiszámított eredőjeként állítják elő. A sugárkövetés módszerét a gyakorlatban általában harmadrendű tetszőleges terjedési mechanizmus kombinációig terjesztik ki, vagy a követett hullámösszetevőt egy előzetesen megadott küszöbtérerősség szint alá csökkenéséig követik. A munkám során a szimulációknál sugárkövetéses modellt használtam. (6)

3.2.2 Maxwell-egyenletek közvetlen megoldására alapuló modell

Egy a Maxwell-egyenletek közvetlen megoldására alapuló modell az FDTD (időtartománybeli véges differencia) módszer. Az FDTD a Maxwell-egyenletek differenciális alakjának időtartománybeli megoldása, amit egyszerűsége miatt először az áramkörök tranziens viselkedésének vizsgálatára használtak. Ennek a módszernek a tárgyalására ezen beszámolóban nem térek ki részletesen, mivel a feladat során ilyen modellel nem foglalkoztam. (6)

4 Adatbázisok

A beltéri hullámterjedési feladatok geometriai leírása azonos a sugárkövetéses és FDTD feladatok megoldására. Az épület adatbázis létrehozására a falakat zárt sík sokszögekre bontjuk, melyeket az alkotó pontokon kívül vastagságukkal és elektromos anyagra jellemző állandójukkal jellemezzük. A sugárkövetéses eljárás igénye szerint a poligonok nem tartalmazhatnak nyílásokat, tehát az ablakok, ajtók leírása geometriai felbontással történhet. (6)



ábra 3: Geometriai felbontás illusztrálása a feladatban is alkalmazott modellen

Ahhoz, hogy számítógéppel lehessen tervezni egy terület elektromágneses ellátottságát, és a terjedési tulajdonságokat, digitális adatokra van szükség a környezetről valamilyen digitális terep modell (DTM) formájában.

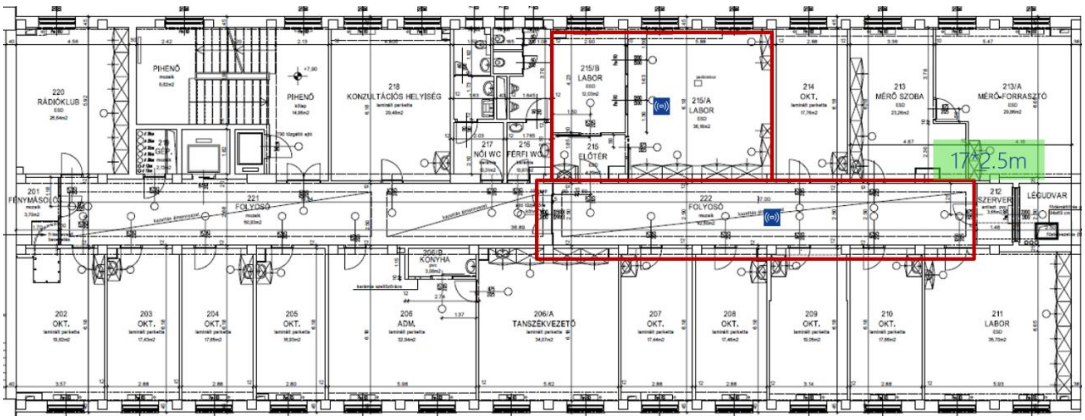
A terepről készített digitális térkép egy adott területrészt egy a pontjához tartozó információval jellemez. Bármilyen ügyesen is választják meg ezt a jellemző pontot, ez az esetek jelentős többségében információvesztéssel jár. A DTM modellek két fő csoportját megkülönböztetve beszélünk szabályos és strukturális modellekről.

A szabályos adatbázisok esetén a terepre jellemző adatot egy szabályos rács metszéspontjaihoz rendelik hozzá (raszteres adatbázis), míg a strukturális adatbázisok esetén a domborzati viszonyok figyelembevételével választják ki a jellemző pontot (pl. vektoros adatbázis). A strukturális modellekkel nagyobb pontosság érhető el, de növekszik a hozzáférési algoritmus bonyolultsága is. (9)

5 Modellezés és szimuláció

A modellt és a szimulációt a REMCOM egy régebbi termékével, a Wireless inSite 2.0.5-ös alkalmazásával végeztem. Ebben a szoftverben kevesebb mindent lehetséges beállítani, mint egy új változatában, azonban jelenleg ez állt rendelkezésre a feladat megoldásához.

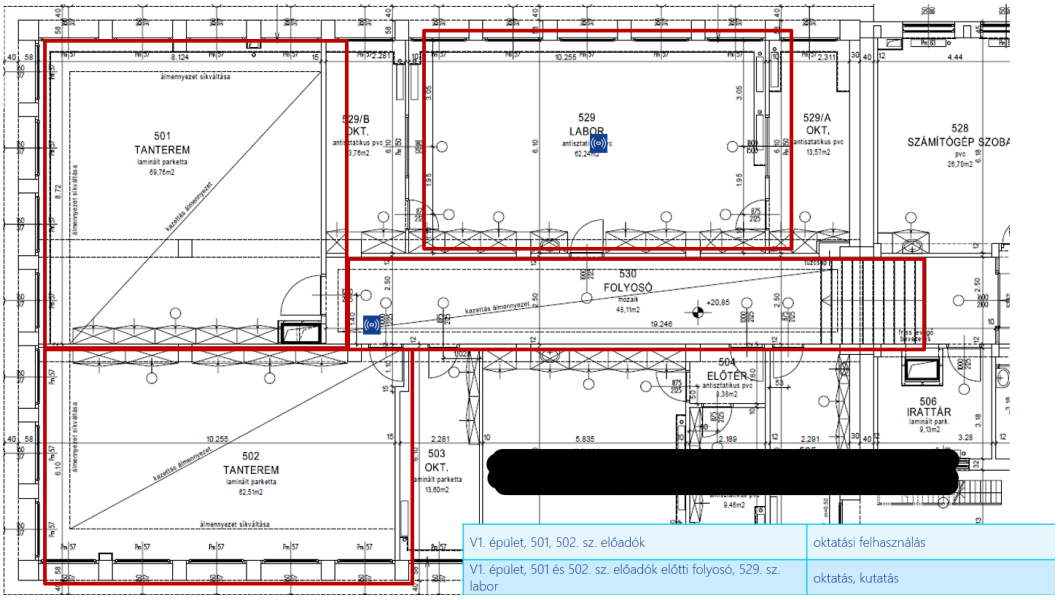
A modell megalkotásához tervrajzok álltak a rendelkezésemre. Sajnálatosan a kapott tervrajzok nem minden esetben voltak tökéletesen olvashatóak, illetve nem szerepelt rajtuk minden adat – például az egyes falak építőanyagai sem -, így ez tovább nehezítette a modell megalkotását. A hibák mérséklése érdekében több fal geometriai méretének, illetve csillapításának is mérését is elvégeztem. Az utóbbi eredményeket (fal csillapítások) az optimalizáló szoftver készítése során használtam fel.



ábra 4: V1 épület második emeletének tervrajza, a lefedendő terület jelölésével

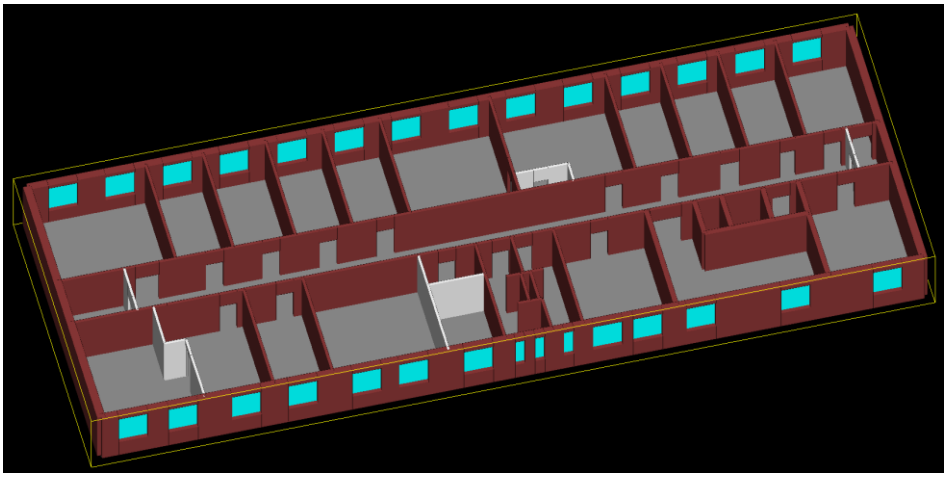
A tervezőprogramban találhatóak beépített építőanyagok, amikhez az anyag tulajdonsága hozzá van rendelve. Például a tégla permittivitásához 4.44, konduktivitásához (vezetés) 0.001, illetve érdességéhez 0 érték van automatikusan rendelve amit, ha jobbnak látunk lehetőségünk van megváltoztatni. Ezen értékeket a szimulációk futtatása során nem állítottam el az alapértelmezett értékről, de a későbbiekben megvizsgáltam ezek pontosságát (pontosabban az egyek faltípusokhoz tartozó csillapítás értékeket). Két helyszínre készítettem modelleket, illetve szimuláltam. Ez a két helyszín a V1 épület második emelete, ahol a lefedendő terület a fenti ábrán piros téglalapokkal van jelölve, illetve ugyancsak a V1 épület ötödik emelete, ahol a lefedendő területek szintén jelölve vannak az ábrán. A V1 második emelete esetén a lefedendő

terület mérete igen kicsi és nem bonyolult a geometriai elrendezés, ezért a munkám során azt tűztem ki célul, hogy az egész területet optimálisan fedjem le, és ne csak a jelölt részeket.

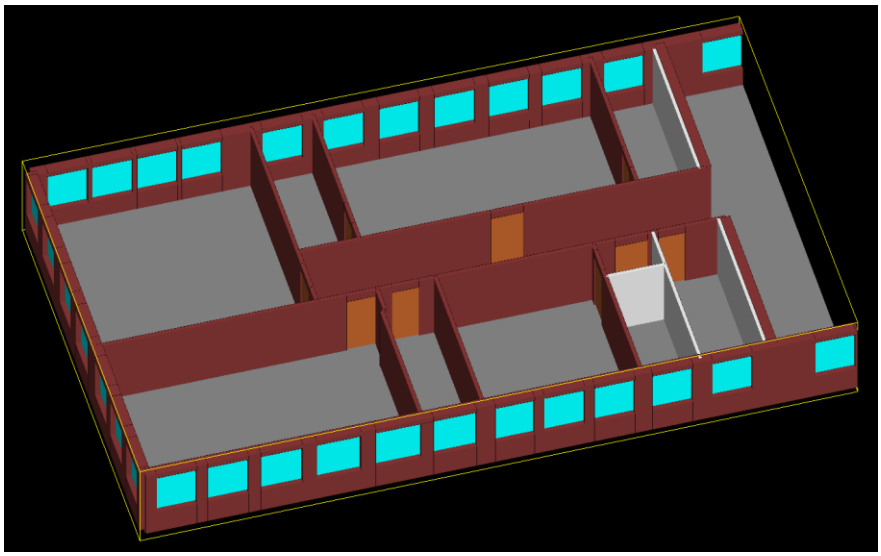


ábra 5: V1 épület ötödik emeletének tervrajza, a lefedendő terület jelölésével

A tervezőprogramban létrehozott modelleket az alábbi ábrákon mutatom be (a lefedni tervezett területeket kiemelve):



ábra 6: V1 épület második emeletének modellje a tervező szoftverben

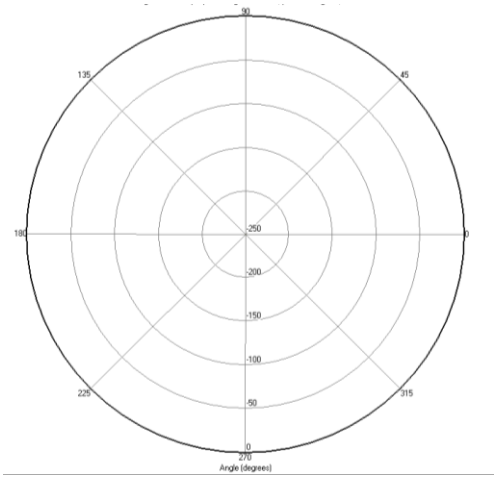


ábra 7: V1 épület ötödik emelet vizsgált részének modellje a tervező szoftverben

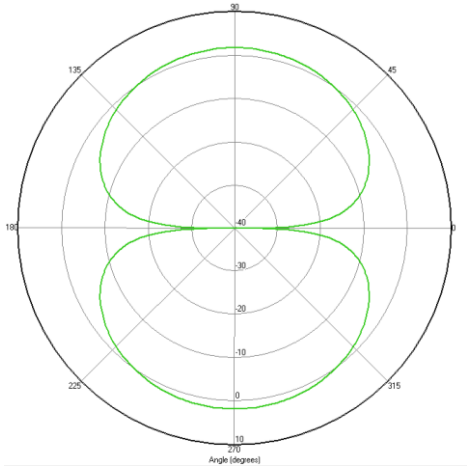
Egy helyszínre két modellt volt szükséges létrehoznom, mivel arról is szerettem volna információt kapni, hogy mennyiben térnek el a szimuláció után kapott paraméterek a nyitott, illetve zárt ajtók esetén (az ajtók fémet tartalmaznak, így nagy az árnyékolásuk).

Elsőként tehát az épületet (az vizsgálni kívánt szintet) alkottam meg. Ezek után a szimulációhoz létre kellett hozni adó és vevő antennákat.

Az adó antennák lesznek azok az antennák, amiknek egy optimális elhelyezését keressük. Ezeket az antennákat a terveken jelölt pontokon helyeztem el. Az adó antennáknak 'omnidirectional', aza körsugárzó antennákat választottam. Ezen antennának – a már korábban említett – 3.6 GHz-es vivőjű, szinuszos hullámformájú jelet állítottam be. Az adóantennák 1mW (0dBm) teljesítménnyel sugároznak. Az adóantennák iránykarakterisztikája az alábbi ábrákon láthatóak:



ábra 8: Omni antenna iránykarakterisztikája

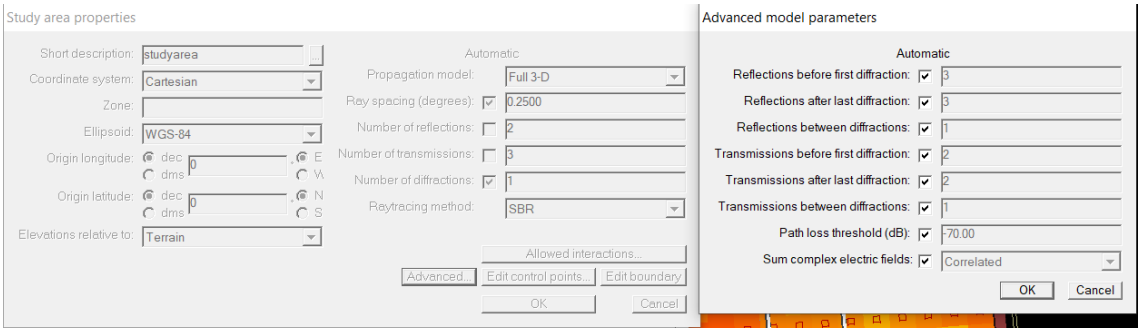


ábra 9: Omni antenna iránykarakterisztikája

A vevő antennáknak – a valóságban nem létező – izotróp iránykarakterisztikájú antennákat választottam, amik az adó által kisugárzott jelet veszik. Az adóantennák elhelyezésének magasságát 1.3 métermagasra választottam, mivel a vevők (telefonok) egy álló ember kezében nagyjából ilyen magasságban helyezkednek el. A vevőantennákat egy rács mentén helyezem el 1 méteres felbontással (azaz 1 méter távol egymástól). Amennyiben ezt az értéket csökkentjük, úgy pontosabbann tudunk számításokat végezni, de a szimuláció futásának hossza is jelentősen megnő.

A szimuláció futtatása előtt szükséges kiválasztanunk a szimulációhoz tartozó paramétereket. Itt igen sok paramétert változtathatunk. Az én esetemben azért volt szükség bizonyos paraméterek változtatására, mivel a saját számítógépemnek nem volt akkora számítási kapacitása, hogy egy szimuláció ésszerű időintervallum alatt lefusson. Ez okból a a reflexiók számát és a transzmissziók számát kettőre, illetve háromra

módosítottam. Itt például a reflexiók száma paraméter azt adja meg, hogy mennyi a reflexiók maximális száma, amit egy nem diffrakciós útvonal képes elviselni. A transzmissziók száma paraméter hasonlóan van definiálva.



ábra 10: Szimuláció lehetséges beállításai

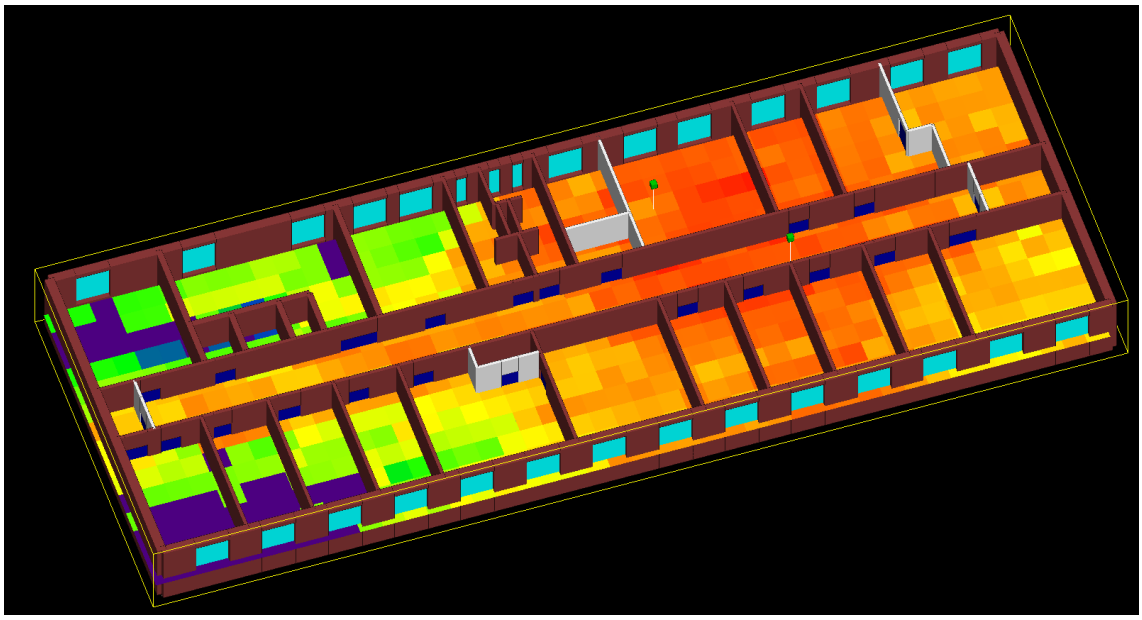
Terjedési modellként a Full-3D modellt választottam, mivel ez az egyetlen modell ebben a tervezőszoftverben, ami figyelembe vesszi a transzmissziót a felületeken, és emiatt ez az egyetlen sugárkövetéses modell ami alkalmazható beltéri tervezéshez. (10)

A Full-3D terjedési modellhez SBR (Shooting and Bouncing Ray)-en alapuló sugárkövetéses módszert választottam. Ennek a módszernek a számítási ideje durván arányos a következővel:

$$\frac{(N_R + N_T + 1)!}{N_R! * N_T!}$$

, ahol N_R a reflexiók száma, N_T a transzmissziók száma. (10) A szimuláció lefuttatása előtt lehetőségünk van a kimeneti paraméterek kiválasztására is.

A szimulációnál többek között kiválasztott kimeneti paraméter volt a vett teljesítmény, ami azt adja meg, hogy a vevőantennák mekkora jelteljesítményt vesznek az egyes adóantennáktól. Ezeket az értékeket vizuálisan is meg lehet tekinteni a programban.



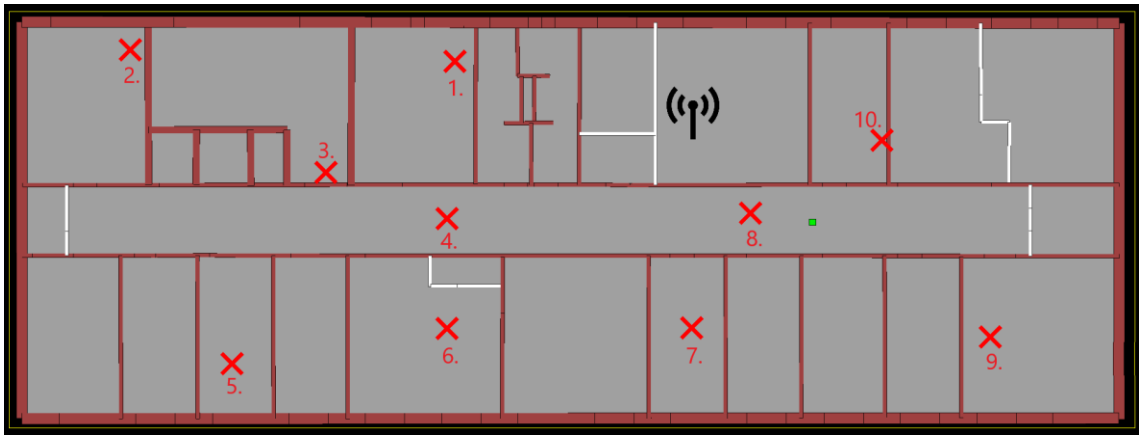
ábra 11: Szimuláció eredménye a modellen (V1 épület 2. emelet)

6 Hullámterjedési modellek összehasonlítása

Az optimalizáláshoz szükséges meghatározni az egyes modellek alkalmazhatóságát, így feladatként tűztem ki, hogy összevetem a (2) fejezetben felsorolt empirikus és félempirikus modelleket determinisztikus sugárút keresés alapú terjedési modellel a 700MHz-4GHz frekvenciasávban.

Az összevetést úgy végeztem el, hogy az általam készített modellekből kiválasztottam a V1 épület második emeletéről készült modellt (ajtók nélkül), amin véletlenszerűen jelöltem be pontokat. A modellre különböző frekvenciákon – a vizsgált frekvenciatartományon belül – szimulációkat futtatok le, és leolvasom, hogy a jelölt vételi pontokban mekkora a vett jel teljesítménye. A jelölt pontokra a különböző modellekkel is kiszámítom a vett jel teljesítményét.

A modellek közül a valószínűleg a determinisztikus modell és a szimuláció segítségével kapott értékek állnak a legközelebb a valósában mérhető értékekhez.



ábra 12: V1 épület 2.emelet modellje felülnézetből és a bejelölt random pontok

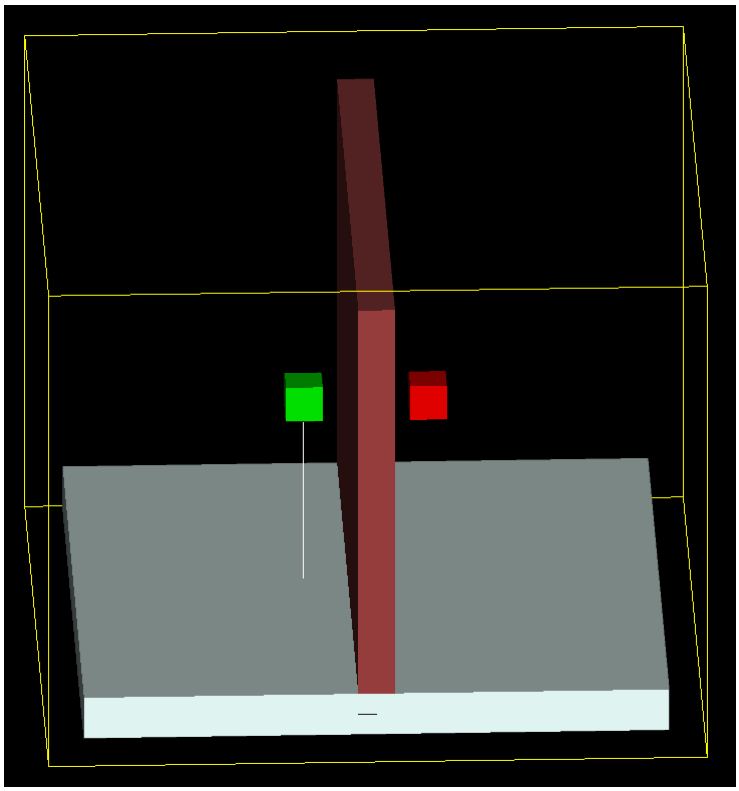
Az egyes pontokban csak az ikonnal jelzett adóból származó vett teljesítményt fogom vizsgálni. Az eredményeket táblázatokban foglalom össze.

6.1 Eredmények kiértékelése

Az szimulációs eredményekből egyértelműen látható, hogy a 2-es és az 5-ös pontokba biztosan nem jut megfelelő teljesítményű jel. A szoftver leírásából kiderül, hogy a kimenet -250dBm, ha a vett jelteljesítmény (P_R) kisebb vagy egyenlő mint $1 \times$

$10^{-25}mW$ azaz, ha $P_R \leq 1 \times 10^{-25}mW$, akkor a kimenet -250dBm. (10) Ezt a két pontot így a további modelleknél külön szükséges volt kezelni, mivel ezen pontok miatt átlag számításnál nagy mértékben befolyásolják a kapott eredményt.

A modellező programmal szimulációkat futtattam arra nézve, hogy meghatározzam az egyes falak csillapítását. Erre azért volt szükség, mivel a Motley-Keenan modellhez ezek meghatározása szükséges volt. A szimuláció során egy egyszerű 'mérési' elrendezést alkalmaztam, amivel a valóságban is kimértem a falak csillapításértékeit a későbbiekben. A csillapítások meghatározása során először szabadtérben – ismert távolságban – helyeztem el az adót és a vevőt, ismert frekvencián lefuttattam a szimulációt, és feljegyeztem, hogy a vevőpontban mekkora a vett teljesítmény. Ezek után az adó és a vevő közé elhelyeztem a vizsgálni kívánt anyagú és méretű falat, és szimuláció futtatása után ismételten feljegyzem, hogy mi a vett teljesítmény érték a vevőpontban. A szabadtéri esetben a vett teljesítmény, és az imént említett módon kapott vett teljesítmények különbségéből kapható meg az adott fal csillapítása. Az eredményeket táblázatba foglaltam.



ábra 13: Mérési elrendezés a modellező szoftverben

Először meghatároztam, hogy az ábrán jelölt random pontok mekkora távolságra vannak az adótól (ezen adatokat a modellezőszoftverből nyertem ki). Az egyes pontok távolságát az adótól táblázatba foglaltam:

Pont [sorszám]	**Távolság – d [m]**
1.	9.86
2.	21.63
3.	13.86
4.	10.76
5.	20.12
6.	13.06
7.	9.02
8.	4.48
9.	13.87
10.	7.01

Az egyes modellekhez tartozó számításokat Matlab segítségével végeztem el. Az eredményeket táblázatba foglaltam, és kiszámoltam az abszolút eltérést a szimulációkban kapott értékektől és ezek %-os értékét, annak érdekében, hogy könnyebben össze lehessen hasonlítani az egyes modelleket. Az eredményekből látható, hogy a legnagyobb eltérések a 2-es és 5-ös pontokban vannak, így elvégeztem olyan számításokat, ahol az ezen pontokhoz tartozó eredményeket nem veszik figyelembe. Azokkal a számításokkal sokkal jobb eredményeket kaptam, amik feltételezéseim szerint a valóságot is jobban tükrözik.

A rádiós hozzáférési hálózat tervezését 3.6GHz-es frekvencián lesz szükséges elvégeznem, így az erre a frekvenciára vonatkozó adatokat kiemelten vizsgáltam. A legkisebb átlagos eltérést a szimulációból származó eredményhez képest 3.6GHz frekvencián az ITU-R P.1238 beltéri hullámterjedési modell adta, ennek értéke ~9.3% (táblázatból kiolvasva, amennyiben a 2-es és 5-ös pontokból származó értékeket nem vesszük figyelembe a számítás során). A második legpontosabb a Motley-Keenen modell

volt, itt az előbb említett eltéréshez tartozó érték ~10.1%. Egyszerű implementálhatósága miatt az optimalizáló szoftverben ezt a modellt alkalmaztam.

6.2 Falcsillapítások kimérése

Az egyes falcsillapítások meghatározásához a helyszínen végeztem méréseket. A mért falakat 1, 2 és 3GHz-es frekvencián, modulálatlan szinuszos jellel mértem.

6.2.1 Méréshez falhasznált eszközök

- 2 darab széles frekvenciasávban működő antenna
- 1 darab jelgenerátor
- 1 darab spektrumanalizátor
- Mérőszalag

6.2.2 Mérési elrendezés

A jelgenerátoron 1, 2 és 3GHz-en szinuszos, 0dBm teljesítményű jelet állítunk elő, ezt kisugározzuk a szabadtérbe egy antenna segítségével és egy adott távolságban (ami ezen mérések esetén 85cm volt) megmérjük a jelet egy másik antenna és egy spektrumanalizátor segítségével.

ábra 14: Referencia mérés, vett teljesítmény mérése falak nélkül

Ezen (referencia) mérés után, a falakat úgy mérjük mag az egyes frekvenciákon, hogy az antennák pozícióját egymáshoz képest nem változtatjuk, és a fal középvonala az pontosan a két antenna közé essen. Az elrendezések összeállítását mérőszalag segítségével végeztem el. Fontos volt továbbá, hogy az antennák merőlegesek legyenek az épp mérni kívánt falra, illetve a magasságuk is azonos legyen. Egy adott fal csillapítást egyszerűen számolhatjuk a mért értékekből, ugyanis a fal csillapítását a referenciamérésnél mért csillapítás és a falnál végzett csillapítás érték különbsége adja. Az így kimért csillapítás értékeket táblázatba foglaltam:

Frekvencia [GHz]	**Szabadtéri mérés (85cm) [dB]**	**Tégla csillapítása (25cm) [dB]**	**Tégla csillapítása (14cm) [dB]**	**Gipszkarton csillapítása (11cm) [dB]**	**Ajtó (fém) (4cm ajtó/2cm fém) [dB]**
1	22.71	3.46	2.47	11.23	18.95
2	26	2.32	3.06	3.87	21.39
3	28.75	12.69	8.71	3.16	21.96

ábra 15: 14cm-es téglafal csillapításának mérése

7 Optimalizálás Genetikus Algoritmussal

Az optimalizálás feladata, hogy úgy helyezzük el az adót/adókat, hogy minél nagyobb lefedettséget érjünk el velük, azaz minél nagyobb az a terület, amely területen belül a vevő egy bizonyos teljesítményszint felett képes venni a jelet az adótól.

A feladatból egyértelműen látható, hogy egy olyan megoldás nem működhet, amiben sorban kipróbáljuk az egyes eseteket és ezekből kiválasztjuk a legjobbat, mivel a lehetséges esetek száma közel végtelen. Nem tudjuk, hogy mik lehetnek a megoldás kimenetelei (a lehetséges kimenetelek száma nagyon nagy), azonban egy adott megoldásról/implementációról lehetséges eldönteni, hogy mennyire megfelelő. Ilyen esetekben lehetséges Genetikus Algoritmus alkalmazása.

7.1 Genetikus Algoritmus általános működése

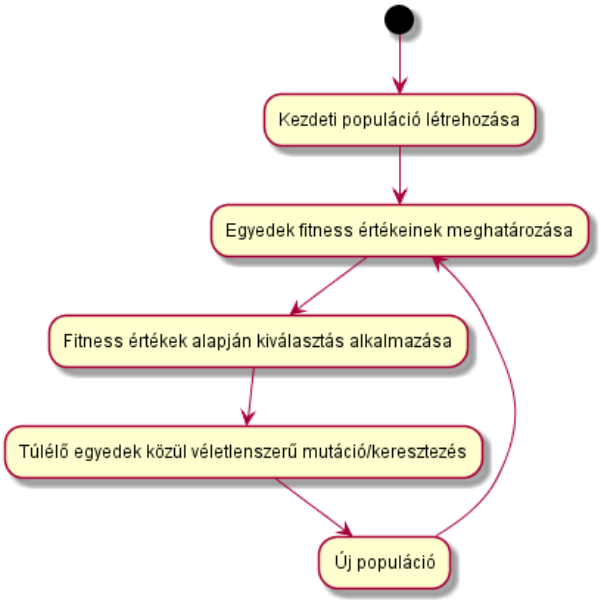
Az iménti bevezetőben említettem egy esetet, ahol Genetikus Algoritmust – továbbiakban GA-t – alkalmazhatunk. Általánosságban olyan feladatok megoldása során alkalmazhatjuk ezt az algoritmust, ahol nem tudjuk pontosan, hogy mi lesz a kimenet, viszont egy kimenetről/implementációról lehetséges eldönteni, hogy mennyire jó. Ezt a jóságot fitness-nek nevezzük és arra törekszünk, hogy minél nagyobb fitness értéket kapjunk. Az algoritmus egy párhuzamos keresési technikát valósít meg.

A GA a biológiai modellen alapszik, és több kifejezést is a biológiához tartozó szaknyelvből veszünk át. Az algoritmus azon alapszik, hogy ahogy a valóságban, itt is létrejönnek egyedek, amely egyedek fő célja a túlélés. Egy egyed egy idő után viszont elpusztul, de a tudását igyekszik továbbadni egy utódjának. Az egyed ezen tudása a véletlenek miatt módosulhat (mutáció/keresztezés).

Biológiai értelemben a kromoszóma a sejtek örökletes tulajdonságait hordozó anyaga. Ezt mi mesterséges értelemben egy karakterfüzérnek tekinthetjük. A kromoszómát a gének alkotják, tehát gén az öröklődő tulajdonságokat hordozó kromoszóma egy része. Az előző példa analógiájára mesterségesen a gén az a karakterfüzérben egy karakter/jellemző. Ezen karakternek van egy értéke, a biológiában ezt nevezzük allélnak. A karakternek van továbbá pozíciója, aminek neve a biológiában lókusz. A karakterfüzérnek (kromoszómának) van egy bizonyos szerkezete, ezt a biológiában genotípusnak nevezzük és ez az örökletes tulajdonságok összessége. A

paraméterkészletet nevezik a biológiában fenotípusnak, ami az előlény alaktani és élettani sajátosságainak összessége.

Az előző kifejezéseket használva, első lépésben képesnek kell létrehoznunk egy kromoszómát, azaz karakterfüzért, amiben az egyes karakterek a gének. A gének jelentését és paraméterkészletét is a mi feladatunk meghatározni. A karakterfüzérekből (ezeket egyedeknek is fogom nevezni) ezután létrehozzuk a kezdeti populációt, ami kromoszómák egy csoportja, ahol az egyes gének konkrét értékeket is kapnak a paraméterkészletből. A feladat bonyolultságától függ, hogy a kezdeti populáció mekkora. Szükségünk van egy módszerre arra nézve, hogy a már említett fitness értéket meg tudjuk határozni az egyes egyedekhez, mivel ez új generáció létrehozása ez alapján történik. Az új generációba valamilyen módon átmásoljuk (szelekció) az előző populációban lévő egyedeket a fitness értékeik figyelembevételével. Erre több módszer is létezik. Az egyik az elicista kiválasztási mód, aminek lényege, hogy csak a jók adhassák tovább az információjukat, ez azonban sok esetben – köztük a biológiában – nem működik megfelelően. Egy másik módszer az úgynevezett rulettkerék algoritmus, ahol a relatív jósággal arányosan felosztunk egy területet, majd a területen belül kiválasztunk egy random pontot. Látható, hogy itt is egy jobb fitness értékű egyednek nagyobb a valószínűsége, hogy ki legyen választva, de ez a módszer a kisebb jóságú egyedeknek is meghagyja az esélyt, hogy kiválasztásra kerüljenek. A másolás mellett pedig bizonyos módosításokat végezhetünk az egyedeken. Ilyen változtatás például a mutáció, amikor egy random gén értékét megváltoztatunk a kromoszómában a paraméterkészlet értékein belül, vagy a keresztezés, ahol két kromoszómából egy harmadikat kapunk oly módon, hogy a kimeneti kromoszóma egyik része az egyik, másik része a másik egyedből tevődik össze. Miután létrehoztuk a következő generációt ugyan azt tesszük vele, mint az elsővel. Ez addig tart amíg a programot le nem állítjuk, vagy megadhatjuk, hogy milyen fitness értéket szeretnénk elérni, de elképzelhető, hogy az sose fog előállni és így a program a végtelenségig futna. Az alap operátorok mellett (szelekció, mutáció, keresztezés) léteznek alacsonyabb szintű operátorok is, például dominancia, inverzió, törlődés, halkítás, kikapcsolás, illetve populációra vonatkozó operátorok, mint a migráció, házassági korlátozás, jósági transzformáló függvények is. Ezekkel azonban nem foglalkoztam alaposabban, az általam írt szoftverben a GA csak az alapoperátorokat használja. (11)



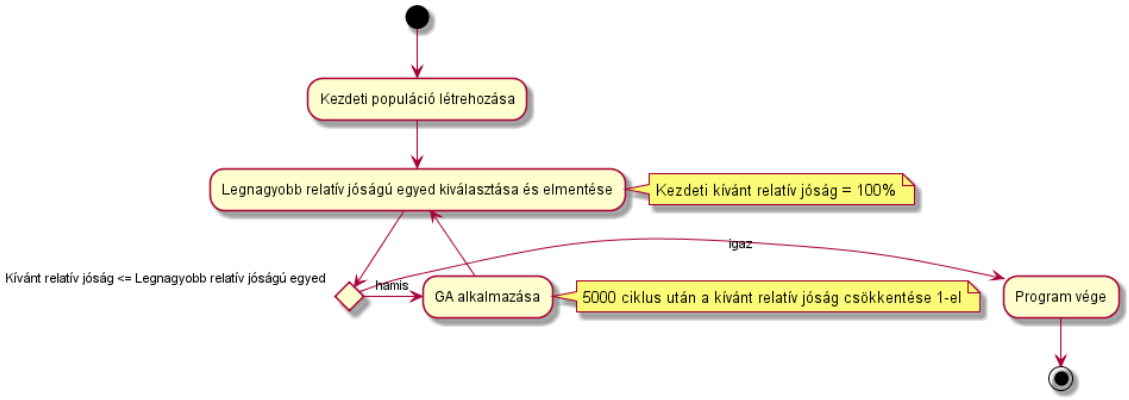
ábra 16: Genetikus algoritmus működése

7.2 Szoftver megvalósítása a gyakorlatban

A szoftvert Java programozási nyelven került megvalósításra. A szoftver bemenete egy térkép a lefedendő területről, ami ez egyes falak pozícióit és csillapításértékait tartalmazza, illetve egy egész szám, hogy hány adóantennát szeretnénk felhasználni. A kimenet az adóantenna/antennák pozíciója/pozíciói.

A program tehát első lépésként egy szöveges fájlból beolvassa az imént említett térképet, és egy egész számot – a területen elhelyezni kívánt adóantennák számát. Ezután az adó/adók random pontokra kerülnek elhelyezésre a területen belül. Egy kromoszóma/egyed tehát pontot/pontokat tartalmaz, amik az adóantenna/antennák pozícióját jelentik. Ebben az esetben a populáció mérete 10 volt, azaz 10 egyed alkot egy populációt/generációt. Ez az érték fix, a program futása során minden generációnak ekkora a mérete. A GA kapcsán a 3 alapoperátor került megvalósításra, a szelekció (rulettkerék algoritmus), mutáció, kereszteződés. Egy adott egyedhez a fitness értéket úgy határozom meg, hogy a térképen rácsosan helyezek el pontokat, amely pontokban megállapítom az adótól érkező vett jelteljesítményt a Motley-Keenan modell segítségével. A falak csillapításánál a gyakorlatban kimért csillapításértékeket veszem figyelembe. Egy pont jónak számít, ha a vett jelteljesítmény ez bizonyos – általam megadott – szint fölött van. A jó pontok száma adja meg a fitness értéket, és a relatív fitness, hogy ezen jó pontok száma hány százaléka a terület összes pontjának. Tehát a relatív fitness megadja, hogy a terület hány százalékát fedi le az adott implementáció. A

program kezdeti leállási feltétele, hogy a teljes terület le legyen fedve, azaz a kívánt relatív fitness 100 (%). Ezen érték 5000 generáció után egyel csökken, annak érdekében, hogy a program ne fusson a végtelenségig. A kezdeti populációban meghatározom az egyedek relatív fitness értékeit, és a legnagyobb fitness értékhez tartozó egyedet eltárolom. A program futása során, ha egy ennél nagyobb jóságú egyedet talál, akkor a nagyobb jóságú egyed (relatív fitness értéke, és az egyedhez tartozó értékek) kerülnek eltárolásra. Amennyiben ezen legnagyobb jóságú egyednek a relatív fitness értéke nagyobb vagy egyenlő, mint a kívánt relatív fitness, a program lefutott, és visszaadja az egyedhez tartozó adatokat, azaz az adó/adók koordinátáit. Az adók a programban a plafonon, 3 méter magasan helyezkednek el, a rácspontok pedig 130cm-es magasságban a padlótól mérve. A program a futása során amennyiben talál egy olyan egyedet aminek a jósága nagyobb mint az eddig talált maximális jóságú egyed, akkor kirajzolja a térképen azokat a rácspontokban található vevőket, ahol a vett teljesítmény érték megfelelő. A szoftver amennyiben eljut a leállási feltételhez kirajzolja, illetve koordinátákban visszaadja a megfelelő adóantenna pozíciót/pozíciókat.



ábra 17: Optimalizáló szoftver magas szintű működése

A vizsgálatokat egyelőre a V1 épület 2. emeletére végeztem el 3.6GHz-es frekvencián. Elkészítettem az emelethez tartozó térképet és lefuttattam a programot 1, 2, és 3 adóantenna esetére, úgy hogy azon pontokat tekintettem jónak, ahol a teljesítmény legalább -75dBm volt (az adóantennák egyenként 0dBm-es teljesítményt sugároznak ki, és a teljesítménye az egyes adóantennáknak nem adódik össze). A futtatás során azt az esetet tekintettem amikor az ajtók nincsenek (nyitva vannak). A megjelenítés nem teljesen pontos, mivel a felhasznált rajzolófüggvény csak egész értékeket fogad.

Egy adóantenna esetén a program kimenete a következő:

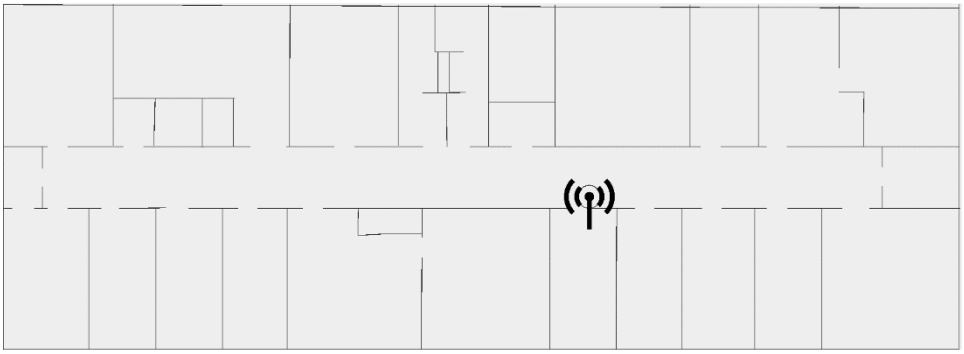
Top max positions:

Fitness: 69.84126984126983

Appropriate transmitter positions (top max):

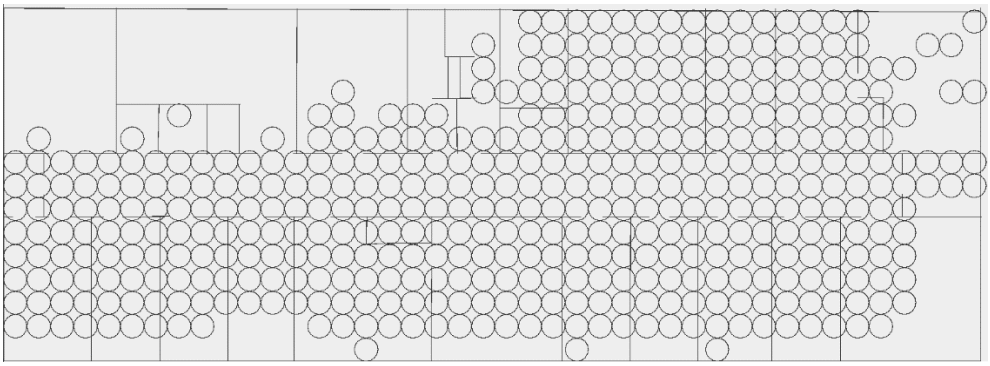
x: 25.105426255815793 y: 6.183684740723124

A visszatérési értéknél a ’*Fitness*’ a relatív jóságot jelenti.

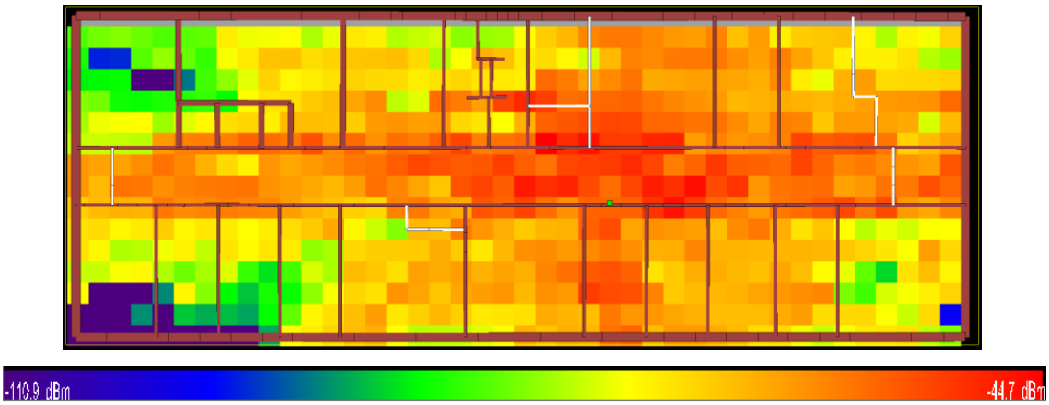


ábra 18: Optimalizáló szoftver kimenete futtatás után – adóantenna pozíciója

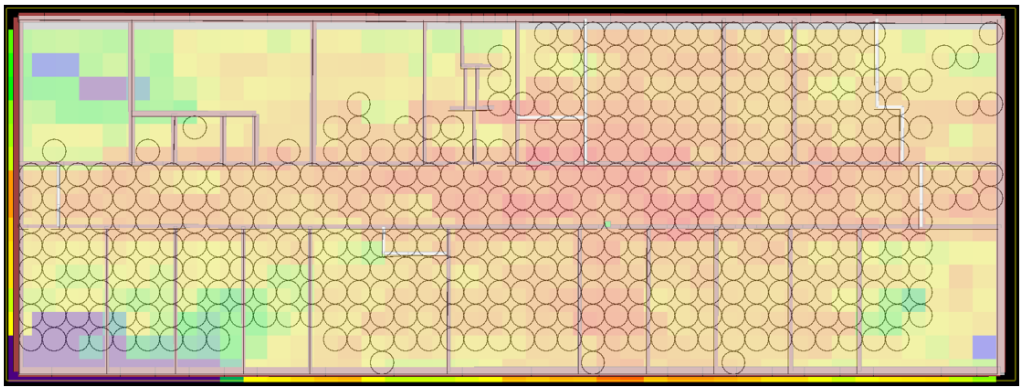
A jobb láthatóság kedvéért az eredményként kapott és az ábrán jelzett adópozícióba egy antenna ikont helyeztem el (utólag).



ábra 19: Optimalizáló szoftver kimenete futtatás után - az adó által lefedett terület



ábra 20: Szimuláció futtatása után a vett jelteljesítmények megjelenítése

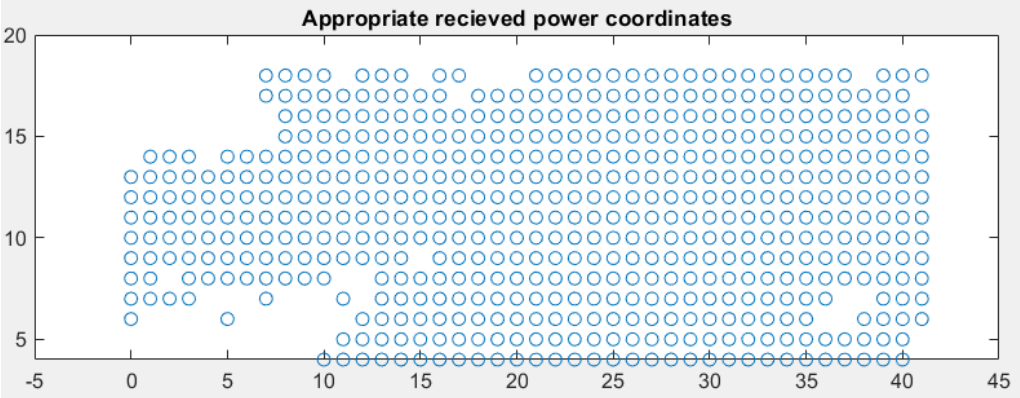


ábra 21: Az optimalizáló program kimenete és a szimuláció kimenetének összehasonlítása

A szimuláció generált egy fájlt, amiben az egyes vevőkhöz tartozó vett teljesítményeket tekinthetjük meg. Ezt a fájlt Matlab segítségével vizsgáltam tovább. Megvizsgáltam, hogy a szimuláció szerint hány darab megfelelő vevő pont van, azaz ahol a vett jelteljesítmény nagyobb vagy egyenlő mint -75dBm. A szimuláció szerint 542 darab ilyen megfelelő pont van, ami az összes pont 86.0317%-a, tehát azzal az elrendezéssel, amit az optimalizáló program adott kimenetként ekkora lefedettséget értünk el a determinisztikus modellben. Ez igen jó eredmény, viszont várható volt. Azért jó eredmény, mert az szoftver szerint talált optimális megoldás szerint a területnek csak a 69.8412%-át tudta lefedni. Az optimalizáló programban a falcsillapításokat a fizikai mérések alapján állítottam be, amik jóval nagyobbak, mint a modellező programban az egyes falakhoz kiszámított csillapítás értékek, így várható volt, hogy abban az esetben csak kisebb terület lefedése lehetséges ugyan akkora teljesítménnyel üzemelő antenna esetén.

```
>> received_power_tool
Enter the threshold in dBm: -75
Number of points over the threshold:
   542

Coverage [%]:
   86.0317
```



ábra 22: Matlab program kimenete

Amennyiben az szoftverben annyinak tekintem a falak csillapítását, ahogy azt a szimulációkból kiszámoltam a következő eredményeket kaptam:

Top max positions:

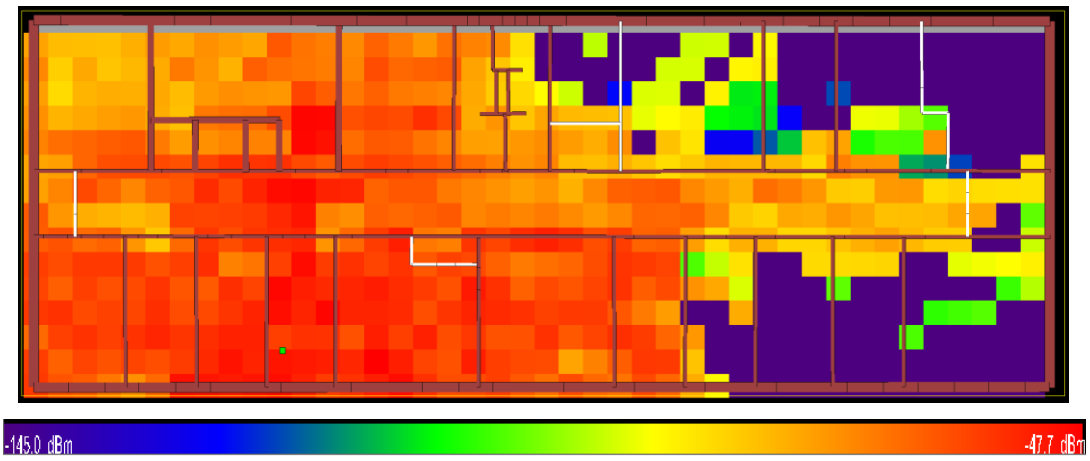
Fitness: 97.93650793650794

Appropriate transmitter positions (top max):

x: 10.233607323164431 y: 1.515912843616662

ábra 23: Optimalizáló program kimenete - adóantenna helyzete

A szoftver szerint így a lefedhető terület egy adóantennával 97.9365% azeredményben kapott elrendezéssel.

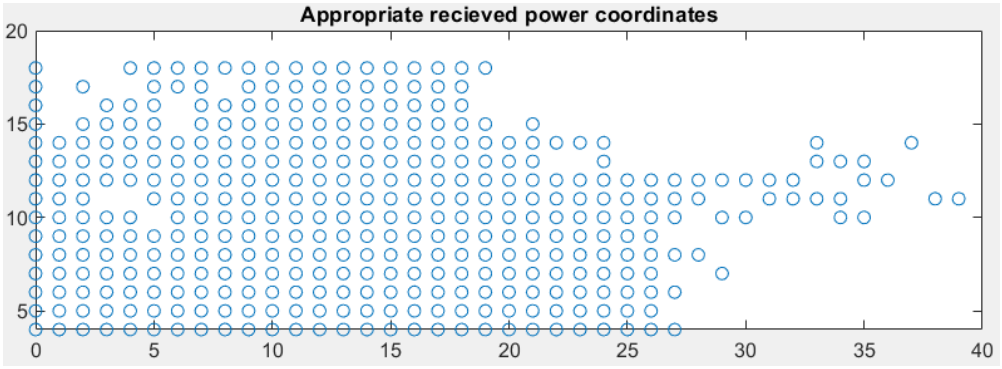


ábra 24: Szimuláció kimenete - vett jelteljesítmények

A szimulációból kapott értékeket elemezve azonban látható, hogy a lefedettség ebben az adópozícióban igen csak lecsökkent, amely igen nagy csökkenés számszerűsítve 25%.

```
>> received_power_tool
Enter the threshold in dBm: -75
Number of points over the threshold:
   386

Coverage [%]:
   61.2698
```



ábra 25: Matlab program kimenete

Ezen számításokat elvégzem kettő és három adóantenna esetére is, valamint a V1 épület 5. emeletére, amit szintén szükséges rádiós lefedettséget tervezni. Ezen számítások eredményei a konferencián fognak bemutatásra kerülni.

8 Eredmények kiértékelése, elvégzett feladatok összegzése

Az előzetes számításokat – dimenzionálás – még jelenleg nem végeztem el arra az esetre, ami a valóságban implementálásra fog kerülni a kijelölte területeken, ezek a konferenciáig kiszámításra és bemutatásra kerülnek.

A hullámterjedési modelleket sikeresen összehasonlítottam a 700MHz-4GHz-es frekvenciasávban a determinisztikus modellel. Feltételezhetjük, hogy a determinisztikus modell közelíti a legjobban a valóságos helyzetet. A legpontosabb modellnek 3.6GHz-es frekvencián – amin a valóságban az adó működni fog – az ITU-R P.1238 beltéri modell bizonyult a legpontosabbnak, de a Motley-Keenan modellt felhasználva is pontos értékeket kaptam, így egyszerűbb megvalósíthatósága miatt az optimalizáló szoftverben ezt a modellt implementáltam.

A falak csillapítása a helyszínen kimérésre került 1GHz, 2GHz és 3GHz-es frekvenciákon.

Az GA-t alkalmazó optimalizáló szoftver implementálásra került Java nyelven. A V1 épület 2. szintjéről egy térképi adatbázist is elkészítettem, amin vizsgáltam a program helyes működését. A vizsgálatok alapján a szoftver alkalmas lehet egy optimális megoldás megtalálására, de ehhez az egyes falak csillapításértékeit megfelelően szükséges megválasztani.

További teendők között szerepel még másik területről térképi adatbázis létrehozása és vizsgálata az optimalizáló szoftverrel, illetve szimulációk futtatása azon területen is. Az optimalizáló szoftverben továbbfejlesztési lehetőség, hogy adott pontban a szoftver a pont elhelyezkedésétől függően automatikusan kiválassza azt a hullámterjedési modellt, amivel a számítás a közelebbi értéket ad a valóságban tapasztalható értékhez képest.

9 Irodalomjegyzék

1. *Model for power consumption of wireless access networks.* **M. Deruyck, E. Tanghe, W. Joseph, W. Vereecken, M. Pickavet, L. Martens, B. Dhoedt.** 2010., IET Science,, old.: 156-159.

2. **Lajos, Dr. Nagy.** *Deterministic indoor wave propagation modelling.* hely nélk. : Budapest University of Technology and Economics, Department of Broadband Infocommunications and Electromagnetic Theory.

3. **https://www.tmit.bme.hu/sites/default/files/attachments/gyak_radios_ok_2015.pdf.** *Infokommunikáció gyakorlatanyag: 2. Rádiós összeköttetések.*

4. **Attila, Szalay Zoltán.** *ISM sávú rádiós modulok mérési alkalmazhatósága.* Budapest : TDK dolgozat, 2011.

5. *Empirical indoor propagation models for LoRa radio link in an office environment.* **Silvano Bertoldo, Miryam Paredes, Lorenzo Carossi, Marco Allegretti, Patrizia Savi.** Torino : ismeretlen szerző, 2019. 13th European Conference on Antennas and Propagation (EuCAP 2019).

6. **Lajos, Dr. Nagy.** *Determinisztikus beltéri hullámterjedési modellek.* hely nélk. : Budapesti Mûszaki Egyetem, Szélessávú Hírközlés és Villamosságtan Tanszék.

7. *An Indoor Path Loss Prediction Model Using Wall Correction Factors for Wireless Local Area Network and 5G Indoor Networks.* **H. A. Obeidat, R. Asif, N. T. Ali, Y. A. Dama, O. A. Obeidat, S. M. R. Jones, W. S. Shuaieb, M. A. Al-Sadoon, K. W. Hameed, A. A. Alabdullah, R. A. Abd-Alhameed.** 2018., AGU 100, old.: 546.

8. **ITU-R.** *Recommendation ITU-R P.1238-11, Propagation data and prediction methods for the planning of indoor radiocommunication systems and radio local area networks in the frequency range 300 MHz to 450 GHz.* Geneva : ITU-R, 2021. old.: 4.

9. **István, Laki.** *Hullámterjedési modellek implementálása Planet tervező rendszerhez.* 1999.

10. **Remcom.** *Wireless InSite: Users manual, version 2.0.5.* hely nélk. : Remcom, 2004.

11. **László, Dr. Kutor.** *Intelligens Rendszerek Elmélete 4 / Optimum keresés genetikus algoritmusokkal.* hely nélk. : Óbudai Egyetem, Neumann János Informatikai Kar, 2014.

Táblázatok, eredmények

	700MHz	1GHz	1.5GHz	2GHz	2.5GHz	3GHz	3.6GHz	4GHz
1	-59.5	-69.9	-74.8	-71.7	-70.8	-74	-79.8	-76.8
2	-250	-250	-250	-250	-250	-250	-250	-250
3	-63	-66.6	-76.2	-73.9	-82.6	-86.6	-80	-78.5
4	-59	-53.4	-57.9	-64.8	-64.9	-66.2	-66	-62.1
5	-250	-250	-250	-250	-250	-250	-250	-250
6	-54.5	-56.4	-61.7	-73.1	-66.1	-75.3	-75.8	-84.5
7	-51.6	-50.6	-54.6	-62.2	-52.2	-62.3	-67.7	-56.6
8	-47	-45.7	-48.3	-48.8	-52.3	-54.2	-54.9	-53.1
9	-66.2	-55.9	-67.7	-75.3	-82.2	-79.8	-76.8	-80.3
10	-42.4	-44.9	-48.6	-48.3	-66.7	-50	-53.4	-58.5

ábra 26: Az jelölt pontokban, adott frekvencián a vett jelteljesítmény értéke [dBm]

	700MHz	1GHz	1.5GHz	2GHz	2.5GHz	3GHz	3.6GHz	4GHz
1	0.804537815	23.8636624	20.26590909	15.00711297	6.6693503	11.943649	19.11604	12.811849
2	71.58204	74.38284	72.45412	72.8146	70.03932	69.52588	70.0124	70.40636
3	1.303174603	12.496997	16.53687664	14.21163735	15.937288	19.377021	13.496125	11.441783
4	2.406779661	3.70411985	9.153367876	5.403395062	7.2986133	1.5413897	0.145303	8.2434783
5	72.87348	75.75428	72.38552	72.82604	69.49076	70.61728	71.38384	70.2978
6	8.738715596	1.1712766	5.15802269	13.84240766	7.8962179	8.4952191	10.835488	18.459408
7	3.193604651	1.47411067	1.589194139	5.198553055	16.292529	0.6608347	7.1314623	14.111484
8	4.746170213	0.72800875	1.840993789	7.147336066	3.3003824	2.2321033	3.2666667	9.2438795
9	10.29410876	1.63989267	12.25243722	15.9310757	20.873358	17.261028	13.138802	14.290785
10	14.75966981	9.70178174	9.21399177	16.30786749	13.171064	18.5972	13.449813	5.8073504
Average:	19.07022811	20.491697	22.08504332	23.86900254	23.096888	22.02516	22.197594	23.511418
Average without 2 & 5:	5.780845139	6.84748121	9.501349152	11.63117317	11.42985	10.013555	10.072463	11.801252

ábra 27: A jelölt pontokban a Motley-Keenan modellel becsült vett jelteljesítmény százalékos eltérése a szimulációs értékekhez képest

	700MHz	1GHz	1.5GHz	2GHz	2.5GHz	3GHz	3.6GHz	4GHz
1	3.69731092	7.29914	8.66337	1.22943	2.7637	0.46	5.16378	0.05091
2	70.81648	69.5772	68.1685	67.169	66.3937	65.7603	65.2247	64.7608
3	5.68269841	4.62192	3.93701	2.4341	6.00847	8.52125	0.69938	4.10115
4	6.6979661	23.6888	20.1582	11.2198	14.0348	14.1876	16.5624	25.7504
5	71.23132	69.9921	68.5834	67.5839	66.8086	66.1752	65.6396	65.1756
6	20.6020183	22.0321	17.2577	2.38947	16.1648	4.07503	5.15501	4.29905
7	17.1003876	25.5372	22.7905	11.8045	36.936	17.278	9.90133	33.5035
8	7.22148936	17.0508	18.0414	21.9525	17.4971	16.3	17.2559	23.4151
9	0.58987915	24.6664	8.13929	0.54329	5.53856	0.71316	4.90872	1.78045
10	33.9875	33.4269	30.5154	36.4996	1.75037	38.9022	32.5657	22.9913
Average:	23.762705	29.7893	26.6255	22.2825	23.3896	23.2373	22.3077	24.5828
Average without 2 & 5:	11.9474062	19.7904	16.1879	11.0091	12.5867	12.5547	11.5265	14.4865

ábra 28: A jelölt pontokban a One slope modellel becsült vett jelteljesítmény százalékos eltérése a szimulációs értékekhez képest

	700MHz	1GHz	1.5GHz	2GHz	2.5GHz	3GHz	3.6GHz	4GHz
1	33.4495798	36.2983	33.8803	31.0216	31.5374	19.8412	25.6673	25.3314
2	72.01556	67.6896	63.3636	63.3636	64.2288	54.7116	54.7116	56.442
3	19.7534921	13.6857	15.4655	12.8345	23.6933	9.61282	2.15588	3.81745
4	28.5488136	10.9809	8.6076	18.3392	20.123	3.81239	3.52091	0.92721
5	73.47496	69.451	65.427	65.427	66.2318	57.379	57.379	58.9886
6	11.1212844	2.53741	0.32593	15.8702	8.93661	0.9842	1.63734	14.8557
7	28.0153101	17.6796	15.4504	25.7812	13.291	11.422	18.4873	5.68887
8	53.2221277	46.9899	45.2058	45.7672	50.2532	42.9048	43.6328	43.4094
9	23.5929003	2.89177	4.79838	14.407	23.2792	1.85276	1.98112	5.91843
10	27.0415094	23.2976	21.9251	21.4402	44.1628	10.0912	15.8157	25.5515
Average:	37.0235537	29.1502	27.445	31.4252	34.5737	21.2612	22.4989	24.093
Average without 2 & 5:	28.0931272	19.2952	18.2074	23.1826	26.9096	12.5652	14.1123	15.6875

ábra 29: A jelölt pontokban a lineáris csillapítású modellel becsült vett jelteljesítmény százalékos eltérése a szimulációs értékekhez képest

	700MHz	1GHz	1.5GHz	2GHz	2.5GHz	3GHz	3.6GHz	4GHz
1	61.9933078	71.5629	73.4258	72.2768	71.9244	73.1385	75.0909	74.1178
2	87.58336	87.5834	87.5834	87.5834	87.5834	87.5834	87.5834	87.5834
3	62.8023006	63.5842	68.172	67.1815	70.6381	71.9943	69.6839	69.1046
4	55.6989429	60.7596	63.8093	67.663	67.7128	68.3468	68.2509	66.257
5	88.33764	88.3376	88.3376	88.3376	88.3376	88.3376	88.3376	88.3376
6	52.8548193	58.3718	61.9476	67.8819	64.4806	68.8203	69.026	72.215
7	62.9046602	62.2449	65.0108	69.286	63.4021	69.3353	71.7812	66.2472
8	65.0788204	71.4976	73.0319	73.3082	75.0945	75.9675	76.274	75.4697
9	54.8188082	56.597	64.162	67.7792	70.4838	69.5961	68.4085	69.7854
10	46.8100629	62.3287	65.1967	64.9805	74.6411	66.1712	68.3251	71.0865
Average:	63.8882722	68.2868	71.0677	72.6278	73.4298	73.9291	74.2761	74.0204
Average without 2 & 5:	57.8702153	63.3683	66.8445	68.7946	69.7972	70.4213	70.855	70.5354

ábra 30: A jelölt pontokban a partícionált modellel becsült vett jelteljesítmény százalékos eltérése a szimulációs értékekhez képest – itt valószínűleg a paraméter rossz megválasztása miatt jelentős eltéréseket kaptam

	700MHz	1GHz	1.5GHz	2GHz	2.5GHz	3GHz	3.6GHz	4GHz
1	3.83804971	22.7763	22.2322	14.7226	10.3809	11.7096	16.1301	11.0568
2	76.52572	75.051	73.3746	72.1852	71.2626	70.5088	69.8715	69.3194
3	17.2842025	13.4874	18.8867	12.3384	18.7793	20.3546	11.7923	8.34854
4	8.3	2.83258	2.07876	4.20201	0.79569	0.10287	2.8203	11.5002
5	76.83496	75.3603	73.6839	72.4945	71.5719	70.8181	70.1808	69.6287
6	7.01907631	1.03227	0.85381	12.248	0.53434	9.24608	7.74274	15.608
7	4.19145631	4.79862	4.79689	3.22733	19.7301	3.34446	2.5452	19.0055
8	12.2380697	0.32516	2.98634	8.02459	5.20535	4.99428	6.55792	12.7693
9	0.44394786	3.08605	8.69114	13.958	18.3746	13.5581	8.1069	10.3934
10	46.6918239	12.104	12.1926	19.046	10.3364	23.3802	18.5084	10.5362
Average:	25.3367306	21.0854	21.9777	23.2447	22.6971	22.8017	21.4256	23.8166
Average without 2 & 5:	12.5008283	7.55529	9.08981	10.9709	10.5171	10.8363	9.27548	12.4022

ábra 31: A jelölt pontokban a ITU-R P.1238 beltéri modellel becsült vett jelteljesítmény százalékos eltérése a szimulációs értékekhez képest

Frequency [MHz]	BRICK_14	BRICK_25	PLASTERBOARD_11	METAL_2
400	-1	-2.1	-0.1	-230.5
700	-2.4	-2.6	-0.2	-225.6
1000	0.1	-2	-1.3	-222.4
1500	-0.2	-1.2	-3	-218.9
2000	-0.6	-0.1	-0.2	-216.6
2500	-0.6	-1	-3.4	-214.6
3000	-0.4	-2.1	-1.7	-212.9
3600	-0.1	-1.8	-0.7	-211.3
4000	-0.5	-0.1	-0.6	-210.4

ábra 32: A szimulációból kapott csillapításértékei az egyes anyagoknak [dB]

Frequency [MHz]	BRICK_25	BRICK_14	PLASTERBOARI	METAL_2 (Door)
1000	-3.46	-2.47	-11.23	-18.95
2000	-2.32	-3.06	-3.87	-21.39
3000	-12.69	-8.71	-3.16	-21.96

ábra 33: A helyszínen kimért csillapításértékek

Felhasznált kódok

- ## Optimalizáló szoftver

Main.java

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    public static void main(String[] args){

        int number_of_transmitters;
        ArrayList<Wall> walls;
        FileIn map = new FileIn();
        Scanner scanner = new Scanner(System.in);
        ArrayList<Transmitter> transmitters = new ArrayList<Transmitter>();

        System.out.println("Number of transmitters: ");
        number_of_transmitters = Integer.parseInt(scanner.nextLine());
        for (int i = 0; i < number_of_transmitters; i++) {
            transmitters.add(new Transmitter(new Point(0,0), 3,0, 3.6));
        }

        walls = map.read();
        PrintWall printWall = new PrintWall();
        printWall.printWalls(walls);


        PositionCalculator poscalc = new PositionCalculator(transmitters,
walls);
        poscalc.searchOptimizedPosition();

    }
}
```

FileIn.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

public class FileIn {
        // FileReader = read the contents of a file as a stream of
characters. One by one
        //              read() returns an int value which contains the byte
value
        //              when read() returns -1, there is no more data to be
read

    public ArrayList<Wall> read() {
```

```
        ArrayList<Wall> walls = new ArrayList<Wall>();
        try {
            BufferedReader bufReader = new BufferedReader(new
FileReader("v1_floor_2.txt"));
            ArrayList<String> listOfLines = new ArrayList<>();
            String line = bufReader.readLine();
            String[] arrOfStr;
            while (line != null) {
                listOfLines.add(line);

                line = bufReader.readLine();
            }

            bufReader.close();

            for (int i = 0; i < listOfLines.size(); i++) {
                arrOfStr = listOfLines.get(i).split("\\t", 6);

                walls.add(new Wall(Double.parseDouble(arrOfStr[0])-0.15,
Double.parseDouble(arrOfStr[1])-4.07, Double.parseDouble(arrOfStr[2])-0.15,
Double.parseDouble(arrOfStr[3])-4.07, Double.parseDouble(arrOfStr[4]),
Material.valueOf(arrOfStr[5])));
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return walls;
    }
}
```

Canvas.java

```
import javax.swing.*;
import java.awt.*;

public class Canvas extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
    }
}
```

Point.java

```
import static java.lang.Math.pow;
import static java.lang.Math.sqrt;

public class Point {
    double x, y;

    public Point(double x, double y) {
```

```
        this.x = x;
        this.y = y;
    }

    public Point() {
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    public void setXY(double x, double y){
        this.x = x;
        this.y = y;
    }

    public double distance2d(Point p) {
        double d;
        d = sqrt( pow((p.x - this.x), 2) + pow((p.y - this.y), 2) );
        return d;
    }

    @Override
    public String toString() {
        return "Point{" +
                "x=" + x +
                ", y=" + y +
                '}';
    }


}
```

Material.java

```
public enum Material {
    BRICK_25(12.69),          //wall width in [cm], attenuation in [dB]
    BRICK_14(8.71),
    PLASTERBOARD_11(3.16),
    METAL_2(21.96);


    private final double attenuation;

    Material(double i) {
        this.attenuation = i;
```

```
    }

    public double getAttenuation(){
        return this.attenuation;
    }
}
```

Transmitter.java

```
import static java.lang.Math.pow;
import static java.lang.Math.sqrt;

public class Transmitter {
    Point position;
    double height;      //[m]
    double power;       //[dBm]
    double frequency;   //[GHz]


    public Transmitter(Point position, double height, double power, double
frequency) {
        this.position = position;
        this.height = height;
        this.power = power;
        this.frequency = frequency;
    }

    public Transmitter(Point position){
        this.position = position;
        this.height = 3;
        this.power = 1;
        this.frequency = 3.6;
    }

    public Point getPosition() {
        return position;
    }

    public void setPosition(Point position) {
        this.position = position;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double height) {
        this.height = height;
    }

    public double getPower() {
        return power;
    }

    public void setPower(double power) {
        this.power = power;
    }
```

```
    public double getFrequency() {
        return frequency;
    }

    public void setFrequency(double frequency) {
        this.frequency = frequency;
    }

    public double distance3d(Point p, double h){                //returns the
distance between the transmitter and the receiver position
        double d;
        d = sqrt( pow((p.x - this.position.x), 2) + pow((p.y -
this.position.y), 2) + pow((h - this.height), 2));
        return d;
    }

    @Override
    public String toString() {
        return "Transmitter{" +
                "position=" + position +
                ", height=" + height +
                ", power=" + power +
                ", frequency=" + frequency +
                '}';
    }
}
```

Wall.java

```
public class Wall {
    Point startingpoint, endpoint;
    double hight;
    Material material;
    double attenuation;


    public Wall(Point startingpoint, Point endpoint, double hight) {
        this.startingpoint = startingpoint;
        this.endpoint = endpoint;
        this.hight = hight;
    }

    public Wall(double xStartingpoint, double yStartingpoint, double
xEndpoint, double yEndpoint, double hight, Material material) {
        startingpoint = new Point(xStartingpoint, yStartingpoint);
        endpoint = new Point(xEndpoint, yEndpoint);
        this.hight = hight;
        this.material = material;
        this.attenuation = this.material.getAttenuation();
    }


    @Override
    public String toString() {
        return "Wall{" +
                "startingpoint=" + startingpoint +
                ", endpoint=" + endpoint +
```

```
                ", hight=" + hight +
                ", material=" + material +
                ", attenuation=" + attenuation +
                '}';
    }
}
```

PrintWall.java

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public class PrintWall {

    public void printWalls(ArrayList<Wall> walls){
        JFrame f = new JFrame();
        JPanel p = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                for(int i = 0; i < walls.size(); i++) {
                    g.drawLine((int)(walls.get(i).startingpoint.getX() *
30.0), (int)(walls.get(i).startingpoint.getY()*30.0),
(int)(walls.get(i).endpoint.getX()*30.0),
(int)(walls.get(i).endpoint.getY()*30.0));
                }
            }
        };

        f.setSize(1500, 1000);
        f.add(p);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }

    public void printTransmitters(ArrayList<Wall> walls, double[]
transmitters){
        JFrame f = new JFrame();
        JPanel p = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                for(int i = 0; i < walls.size(); i++) {
                    g.drawLine((int)(walls.get(i).startingpoint.getX() *
30.0), (int)(walls.get(i).startingpoint.getY()*30.0),
(int)(walls.get(i).endpoint.getX()*30.0),
(int)(walls.get(i).endpoint.getY()*30.0));
                }

                for (int i = 0; i < transmitters.length; i = i + 2) {
                    g.drawOval((int)(transmitters[i]*30.0),
(int)(transmitters[i+1]*30.0), 30, 30);
                }
            }
        };

        f.setSize(1500, 1000);
```

```
        f.add(p);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }


    public void printNotEnoughPowerPoints(ArrayList<Wall> walls,
ArrayList<Point> not_enough_power){
        JFrame f = new JFrame();
        JPanel p = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                for(int i = 0; i < walls.size(); i++) {
                    g.drawLine((int)(walls.get(i).startingpoint.getX() *
30.0), (int)(walls.get(i).startingpoint.getY() * 30.0),
(int)(walls.get(i).endpoint.getX() * 30.0),
(int)(walls.get(i).endpoint.getY() * 30.0));
                }

                for (int i = 0; i < not_enough_power.size(); i++) {
                    g.drawOval((int)(not_enough_power.get(i).getX() * 30.0),
(int)(not_enough_power.get(i).getY() * 30.0), 30, 30);
                }
            }
        };

        f.setSize(1500, 1500);
        f.add(p);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

PositionCalculator.java

```
import java.util.ArrayList;

import static java.lang.Math.pow;
import static java.lang.Math.sqrt;


public class PositionCalculator {
    ArrayList<Transmitter> transmitters;
    ArrayList<Wall> walls;

    public PositionCalculator(ArrayList<Transmitter> transmitters,
ArrayList<Wall> walls) {
        this.transmitters = transmitters;
        this.walls = walls;
    }

    public void searchOptimizedPosition() {
        boolean stop = false;
        GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm();
        Point max = new Point();
        max = maxWall();
        double req_fitness = 100.0;
```

```
        double top_max = 0.0;
        double[] fitnesses;
        double[] gene;
        double tmpx, tmpy;
        int flag = 1;
        ArrayList<double[]> population = new ArrayList<double[]>();
        ArrayList<Point> topmax_array = new ArrayList<Point>();
        Point tmp_point = new Point();
        gene = new double[2 * transmitters.size()];
        double max_f = 0.0;
        int a = 0;
        int indicate = 0;
        double[] top_max_positions = new double[0];

        for (int i = 0; i < 10; i++) {
            for (int j = 0; j < 2 * transmitters.size(); j++) {
                tmpx = (double) Math.abs(Math.random() * max.getX());
                tmpy = (double) Math.abs(Math.random() * max.getY());
                if (j == 1) {
                    gene[j] = tmpy;
                } else if (j == 0 || j % 2 == 0) {
                    gene[j] = tmpx;
                } else {
                    gene[j] = tmpy;
                }
            }
            population.add(gene);
            gene = new double[2 * transmitters.size()];
        }
        //A population already created with random genes (first population)

        fitnesses = new double[population.size()];

        while (stop == false) {

            for (int i = 0; i < population.size(); i++) {
                fitnesses[i] = fitCalc(population.get(i), top_max);
            }

            //max fitness
            for (int i = 0; i < population.size(); i++) {
                if (fitnesses[i] > max_f) {
                    max_f = fitnesses[i];
                    indicate = i;
                }
            }

            if (top_max < max_f) {
                top_max = max_f;
                top_max_positions = new double[population.get(0).length];
                top_max_positions = population.get(indicate);
                System.out.println("Top max: " + top_max);
            }


            a++;

            //examine a single population if there is a solution
            if (checkPopulation(population, fitnesses, req_fitness, top_max))
```

```
{
                System.out.println("------------------------");
                System.out.println("Top max positions: ");
                System.out.println("Fitness: " + top_max + "\nAppropriate
transmitter positions (top max):\n");
                for (int j = 0; j < population.get(0).length; j = j + 2) {
                    System.out.println("x" + j + ": " + top_max_positions[j]
+ " y" + (j + 1) + ": " + top_max_positions[j + 1]);
                }
                PrintWall printWall = new PrintWall();
                printWall.printTransmitters(this.walls, top_max_positions);

                System.out.println("End of the program");
                stop = true;
            } else {
                //if no solution can be found, then creating a new population
                population = geneticAlgorithm.nextPopulation(population,
fitnesses, max);
            }

            if (flag == 5000) {
                req_fitness = req_fitness - 1.0;
                flag = 0;
            }
            flag++;

        }

    }

    //return: 'good' points in %
    public double fitCalc(double[] gene, double topmax) {
        int good_points = 0;
        double sum_receiver_points = 0;
        Point max;
        double threshold = 75;
        max = maxWall();
        PrintWall printWall = new PrintWall();
        ArrayList<Point> trans_points = new ArrayList<Point>();
        ArrayList<Point> receiver_points = new ArrayList<Point>();
        //ArrayList<Point> bad_receiver_points= new ArrayList<Point>();
        ArrayList<Point> full_points = new ArrayList<Point>();
        ArrayList<Point> goods_points = new ArrayList<Point>();
        for (int i = 0; i < max.getX(); i++) {
            for (int j = 0; j < max.getY() - 1; j++) {
                receiver_points.add(new Point((double) i, (double) j));
                full_points.add(new Point((double) i, (double) j));
            }
        }

        sum_receiver_points = receiver_points.size();


        for (int i = 0; i < gene.length; i = i + 2) {

            trans_points.add(new Point(gene[i], gene[i + 1]));
        }
```

```
        for (int i = 0; i < trans_points.size(); i++) {
            for (int j = 0; j < receiver_points.size(); j++) {
                if (isThereEnoughPower(receiver_points.get(j), 1.3,
trans_points.get(i), 3.0, threshold) &&
(!goods_points.contains(receiver_points.get(j)))) {
                    good_points++;
                    goods_points.add(receiver_points.get(j));
                    //receiver_points.remove(j);
                }
            }
        }


        if (((double) good_points / sum_receiver_points * 100.0) > topmax &&
topmax != 0.0) {
            printWall.printNotEnoughPowerPoints(this.walls, goods_points);

        }


        return (double) good_points / sum_receiver_points * 100.0;
    }

    public Point maxWall() {
        Point max = new Point();
        max.setXY(0.0, 0.0);
        for (int i = 0; i < this.walls.size(); i++) {
            if (this.walls.get(i).startingpoint.getX() > max.getX()) {
                max.setX(this.walls.get(i).startingpoint.getX());
            }
        }
        for (int i = 0; i < this.walls.size(); i++) {
            if (this.walls.get(i).startingpoint.getY() > max.getY()) {
                max.setY(this.walls.get(i).startingpoint.getY());
            }
        }
        for (int i = 0; i < this.walls.size(); i++) {
            if (this.walls.get(i).endpoint.getX() > max.getX()) {
                max.setX(this.walls.get(i).endpoint.getX());
            }
        }
        for (int i = 0; i < this.walls.size(); i++) {
            if (this.walls.get(i).endpoint.getY() > max.getY()) {
                max.setY(this.walls.get(i).endpoint.getY());
            }
        }
        //System.out.println("Max wall: " + max);
        return max;
    }

    public boolean isThereEnoughPower(Point receiver, double rec_hight, Point
transmitter, double trans_height, double threshold) {
        double attenuate;
        double trans_rec_dist;
        double frequency = 3.6e9;
        double lambda = 3e8 / frequency;

        trans_rec_dist = sqrt(pow(((transmitter.getX() - receiver.getX())),
2) + pow(((transmitter.getY() - receiver.getY())), 2) + pow(((trans_height -
```

```
rec_hight)), 2));

        attenuate = 20 * Math.log10((4 * Math.PI * trans_rec_dist) / lambda);


        for (int i = 0; i < walls.size(); i++) {
            if (cross(walls.get(i), receiver, transmitter)) {

                attenuate = attenuate + walls.get(i).attenuation;
            }
        }
        if (attenuate > threshold) {
            return false;
        } else {
            return true;    //true, if the power is enough (-->transmitter
working on 0dBm)
        }
    }

    public boolean cross(Wall wall, Point receiver, Point transmitter) {
        Point i1, i2, ia;
        double A1, A2, b1, b2;

        i1 = new Point();
        i1.setX(Math.min(receiver.getX(), transmitter.getX()));
        i1.setY(Math.max(receiver.getX(), transmitter.getX()));

        i2 = new Point();
        i2.setX(Math.min(wall.startingpoint.getX(), wall.endpoint.getX()));
        i2.setY(Math.max(wall.startingpoint.getX(), wall.endpoint.getX()));

        ia = new Point();
        ia.setX(Math.max(i1.getX(), i2.getX()));
        ia.setY(Math.min(i1.getY(), i2.getY()));


        if (i1.getY() < i2.getX()) {
            return false;
        }

        if ((receiver.getX() - transmitter.getX()) != 0) {
            A1 = (receiver.getY() - transmitter.getY()) / (receiver.getX() -
transmitter.getX());
        } else {
            A1 = (receiver.getY() - transmitter.getY()) / 0.1;
        }

        if ((wall.startingpoint.getX() - wall.endpoint.getX()) != 0) {
            A2 = (wall.startingpoint.getY() - wall.endpoint.getY()) /
(wall.startingpoint.getX() - wall.endpoint.getX());
        } else {
            A2 = (wall.startingpoint.getY() - wall.endpoint.getY()) / 0.1;
        }

        b1 = receiver.getY() - A1 * receiver.getX();
        b2 = wall.startingpoint.getY() - A2 * wall.startingpoint.getX();


        if (A1 == A2) {
```

```
            return false;
        }


        double xa;
        if ((A1 - A2) != 0) {
            xa = (b2 - b1) / (A1 - A2);
        } else {
            xa = (b2 - b1) / 0.1;
        }

        if (xa < ia.getX() || xa > ia.getY()) {
            return false;
        }
        return true;    //true, if crosses
    }

    //return true, if we can find a gene in the population which fitness is
higher than the required
    public boolean checkPopulation(ArrayList<double[]> population, double[]
fitnesses, double req_fitness, double top_max) {
        for (int i = 0; i < population.size(); i++) {
            if (top_max >= req_fitness) {
                return true;
            }
        }
        return false;
    }
}
```

GeneticAlgorithm.java

```
import java.util.ArrayList;
import java.util.Random;

public class GeneticAlgorithm {

    public ArrayList<double[]> nextPopulation(ArrayList<double[]>
previous_population, double[] fitnesses, Point max_point){
        ArrayList<double[]> next_population = new ArrayList<double[]>();
        Random r = new Random();
        Random random = new Random();
        double[] tmp2;
        ArrayList<double[]> prev_population_with_fitnesses;
        ArrayList<double[]> prev_population_ordered = new
ArrayList<double[]>();

        prev_population_with_fitnesses = order(previous_population,
fitnesses);



        tmp2 = new double[prev_population_with_fitnesses.get(0).length-1];
        for (int i = 0; i < previous_population.size(); i++) {
            for (int j = 0; j < prev_population_with_fitnesses.get(0).length-
1; j++) {
                tmp2[j] = prev_population_with_fitnesses.get(i)[j];
            }
```

```
            prev_population_ordered.add(i, tmp2);
            tmp2 = new double[prev_population_with_fitnesses.get(0).length-
1];
        }

        for(int i = 0; i < prev_population_with_fitnesses.size(); i++) {
            double result = (double) r.nextInt(100);
            for (int j = i; j < prev_population_with_fitnesses.size(); j++) {
                if (result <
prev_population_with_fitnesses.get(j)[prev_population_with_fitnesses.get(0).l
ength-1]) {
                    if ((int) random.nextInt(1000) % 5 == 0 && i <
previous_population.size() - 3) {

next_population.add(crossover(prev_population_ordered.get(i),
prev_population_ordered.get(i + 1)));
                        j = previous_population.size();
                    } else {
                        if ((int) random.nextInt(1000) % 3 == 0) {

next_population.add(mutation(prev_population_ordered.get(i), max_point));
                            j = previous_population.size();
                        }else{

next_population.add(prev_population_ordered.get(i));
                            j = previous_population.size();
                        }
                    }
                }
            }
        }

        return next_population;
    }

    //one-point crossover
    public double[] crossover(double[] gene1, double[] gene2){

        double[] gene_x = new double[gene1.length];
        Random r = new Random();
        int cross_point;
        cross_point = r.nextInt(gene1.length + 1);
        for(int i = 0; i < gene1.length; i++){
            if(i <= cross_point){
                gene_x[i] = gene1[i];
            }else{
                gene_x[i] = gene2[i];
            }
        }

        return gene_x;
    }


    public double[] mutation(double[] gene, Point max_point){

        double[] gene_x = new double[gene.length];
        Random r = new Random();
        int which_gene;
```

```
        which_gene = r.nextInt(gene.length);
        int tmp;

        for(int i = 0; i < gene.length; i++){
            if(i == which_gene && i % 2 ==0){
                tmp = (int)Math.abs(max_point.getX()) + 1;
                gene_x[i] = (double)r.nextInt(tmp) * Math.random();
                //System.out.println("Halloo: " + gene_x[i]);
            }else if(i == which_gene && i % 2 !=0) {
                tmp = (int) Math.abs(max_point.getY()) + 1;
                gene_x[i] = (double) r.nextInt(tmp) * Math.random();
            }else{
                gene_x[i] = gene[i];
            }
        }

        return gene_x;
    }

    public ArrayList<double[]> order(ArrayList<double[]> population, double[]
fitnesses){
        ArrayList<double[]> ordered_population_with_fitnesses = new
ArrayList<double[]>();
        double[] norm_fitnesses = new double[fitnesses.length];
        double[] gene_with_fitness_tmp;
        double sum_fitnesses = 0.0;
        double sum = 0.0;
        double[] tmp;
        boolean sorted = false;
        double[] gene_tmp;


        for(int i = 0; i < population.size(); i++){

            sum_fitnesses = sum_fitnesses + fitnesses[i];
        }


        for(int i = 0; i < population.size(); i++){

            norm_fitnesses[i] = (fitnesses[i] / sum_fitnesses) * 100.0;
        }

        gene_tmp = new double[population.get(0).length + 1];
        for(int i = 0; i < population.size(); i++) {
            gene_tmp[population.get(0).length] = norm_fitnesses[i];
            for (int j = 0; j < population.get(0).length; j++) {

                    gene_tmp[j] = population.get(i)[j];
            }
            ordered_population_with_fitnesses.add(i, gene_tmp);
            gene_tmp = new double[population.get(0).length + 1];
        }



        while(!sorted){
            sorted = true;
            for (int i = 0; i < population.size()-1; i++) {
```

```
if(ordered_population_with_fitnesses.get(i)[ordered_population_with_fitnesses
.get(0).length-1] >
ordered_population_with_fitnesses.get(i+1)[ordered_population_with_fitnesses.
get(0).length-1]){
                    gene_with_fitness_tmp =
ordered_population_with_fitnesses.get(i);
                    ordered_population_with_fitnesses.set(i,
ordered_population_with_fitnesses.get(i+1));
                    ordered_population_with_fitnesses.set(i+1,
gene_with_fitness_tmp);
                    sorted = false;
                }
            }
        }


        tmp = new double[population.get(0).length + 1];
        for(int i = 0; i < population.size(); i++){
            sum = sum +
ordered_population_with_fitnesses.get(i)[population.get(0).length];
            for (int j = 0; j < population.get(0).length + 1; j++) {
                if(j != population.get(0).length){
                    tmp[j] = ordered_population_with_fitnesses.get(i)[j];
                }
                else{
                    tmp[j] = sum;
                }
            }

            ordered_population_with_fitnesses.set(i, tmp);
            tmp = new double[population.get(0).length + 1];
        }
        return ordered_population_with_fitnesses;
    }
}
```

- **Matlab kódok**

Szimulációs eredményeket feldolgozó kód

```
function received_power_tool

fileID = fopen('v1_2_uj.power.t001_02.r001.p2m', 'r');
threshold = input("Enter the threshold in dBm: ");
received_power = [];

x_coordinate = [];
y_coordinate = [];

x_coordinate_all = [];
y_coordinate_all = [];
all_points = 0;
ids = [];
while ~feof(fileID)
    tline = fgetl(fileID);
    str = tline;
```

```
    splitted_line = str2double(strsplit(str));
    if str(1) ~= '#'
        received_power = [received_power splitted_line(7)];
        x_coordinate_all = [ x_coordinate_all
double(splitted_line(3))];
        y_coordinate_all = [ y_coordinate_all
double(splitted_line(4))+4];
        if splitted_line(7) > threshold
            x_coordinate = [ x_coordinate double(splitted_line(3))];
            y_coordinate = [ y_coordinate double(splitted_line(4))+4];

        end
    end
end

nexttile
plot(x_coordinate,y_coordinate, 'o'); %a pontok megjelenitese
koordinaterendszerben
title('Appropriate recieved power coordinates');

over_threshold = 0;

for n = 1 : length(received_power)
    all_points = all_points + 1;
    if received_power(n) > threshold
        over_threshold = over_threshold + 1;
    end
end

disp("Number of points over the threshold:");
disp(over_threshold)


x_in_examined = []; %vizsgalt terulet koordinatai (piros teglalap)
y_in_examined = [];
power = []; %ebben a tombben tarolom el a vizsgalt teruleten levo vett
teljesitmenyertekeket, majd ebbol szamolok atlagot

for n = 1 : length(received_power)
%     if ((x_coordinate_all(n) <= 11) && (y_coordinate_all(n) <= 20))
|| (x_coordinate_all(n) <= 25 && x_coordinate_all(n) >= 11 &&
y_coordinate_all(n) <= 13 && y_coordinate_all(n) >= 10) ||
(x_coordinate_all(n) <= 22 && 11 <= x_coordinate_all(n) &&
y_coordinate_all(n) <= 21 && 13 <= y_coordinate_all(n))  %ahol lefedni
kívánjuk a területet azokon a pontokon vizsgálódunk, ez a v1, 5
emeleten vizsgalt terulet
    if ((x_coordinate_all(n) <= 30 && x_coordinate_all(n) >= 21 &&
y_coordinate_all(n) <= 20 && y_coordinate_all(n) >= 13) ||
(x_coordinate_all(n) <= 39 && 20 <= x_coordinate_all(n) &&
y_coordinate_all(n) <= 13 && 10 <= y_coordinate_all(n)))  %ahol
lefedni kívánjuk a területet azokon a pontokon vizsgálódunk, ez a v1,
2 emeleten vizsgalt terulet
        x_in_examined = [ x_in_examined x_coordinate_all(n)];
        y_in_examined = [ y_in_examined y_coordinate_all(n)];
        power = [power received_power(n)];
    end
end


sum = 0.0
for n = 1 : length(power)
```

```
    sum = sum + power(n);
end



examined_appropriate_counter = 0;
for n = 1 : length(power)
    if (power(n) > threshold)
        examined_appropriate_counter = examined_appropriate_counter +
1;
    end
end


disp("Coverage [%]:");
disp((100/length(received_power))*over_threshold);

fclose(fileID);

end
```

Motley-Keenan

```
d = [ 9.86 21.63 13.86 10.76 20.12 13.06 9.02 4.48 13.87 7.01];
l = [9.8 15 10 7.6 12.4 7.6 4.8 2.4 7.2 2.4];
frequency = 7e8;
c = 3e8;
lambda = c / frequency;

for i = 1:10
    Lp(i) = (20*log10((4*pi)/lambda) + 20*log10(d(i)) + l(i)) * (-1);
end
disp(Lp');
```

One slope

```
d = [ 9.86 21.63 13.86 10.76 20.12 13.06 9.02 4.48 13.87 7.01];
N = 33;
frequency = 700;
c = 3e8;
lambda = c / frequency;

for i = 1:10
    Lp(i) = (20*log10(frequency) + N*log10(d(i)) - 28) * (-1);
end
disp(Lp');
```

Lineal

```
d = [ 9.86 21.63 13.86 10.76 20.12 13.06 9.02 4.48 13.87 7.01];
a = 3.8;
frequency = 1e9;
```

```
c = 3e8;
lambda = c / frequency;

for i = 1:10
    Lp(i) = (20*log10(d(i)) + a * d(i)) * (-1);
end
disp(Lp');
```

Partitioned

```
d = [ 9.86 21.63 13.86 10.76 20.12 13.06 9.02 4.48 13.87 7.01];
m = 1;
frequency = 1e9;
c = 3e8;
lambda = c / frequency;

for i = 1:10
    if m < d(i) && d(i) <= (10*m)
        Lp(i) = (20*log10(d(i))) * (-1);
    elseif (10*m) < d(i) && (20*m) >= d(i)
        Lp(i) = (20 + 30*log10(d(i)/10)) * (-1);
    elseif (20*m) < d(i) && (40*m) >= d(i)
        Lp(i) = (29 + 60*log10(d(i)/20)) * (-1);
    else
        Lp(i) = (47 + 120*log10(d(i)/40)) * (-1);
    end
end
disp(Lp');
```

ITU-R P.1238

```
d = [ 9.86 21.63 13.86 10.76 20.12 13.06 9.02 4.48 13.87 7.01];
m = 1;
frequency = 1e9;
c = 3e8;
lambda = c / frequency;

for i = 1:10
    if m < d(i) && d(i) <= (10*m)
        Lp(i) = (20*log10(d(i))) * (-1);
    elseif (10*m) < d(i) && (20*m) >= d(i)
        Lp(i) = (20 + 30*log10(d(i)/10)) * (-1);
    elseif (20*m) < d(i) && (40*m) >= d(i)
        Lp(i) = (29 + 60*log10(d(i)/20)) * (-1);
    else
        Lp(i) = (47 + 120*log10(d(i)/40)) * (-1);
    end
end
disp(Lp');
```