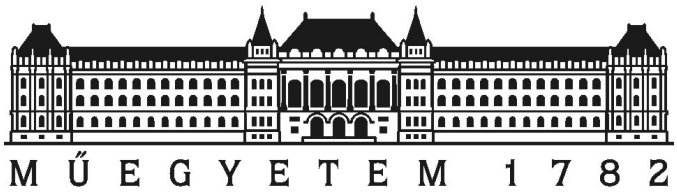


Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Szélessávú Hírközlés és Villamosságtan Tanszék

Koncentrált paraméterű hálózatok tervezése gráfalapú genetikus algoritmussal
TDK dolgozat

Baráth László

Konzulens: Dr. Pávó József

2022

Tartalomjegyzék

Kivonat

Míg a digitális áramkörök tervezési nehézségeinek jelentős részét napjainkra átvették a számítógép-alapú megoldások, addig az analóg áramkörök tervezése során a számítógépes megoldások főleg a kész design szimulációját és esetleg a hálózat paramétereinek optimalizálását végzik el. A munkafolyamat így főként a mérnök általi tervezésből és a vázlat numerikus optimalizációjából áll.

A gépi tervezés általában egy kész áramköri topológia paramétereinek (alkatrészek értékei, toleranciája) optimalizálását jelenti numerikus algoritmusok segítségével. Erre több ismert sztochasztikus és determinisztikus módszer létezik. Az optimalizált áramkörrel szemben többfajta követelmény is támasztható, idő vagy frekvenciatartománybeli viselkedés, a tulajdonságok stabilitása az alkatrészek paramétereinek szórásából adódóan, termikus viselkedés, stb.

Léteznek módszerek az áramköri topológia optimalizálására is. A leginkább kutatott irány az áramkörök tervezése genetikus algoritmusokkal. Ez a módszer egyaránt alkalmas a hálózat topológiájának és paramétereinek optimalizálására (akár párhuzamosan is), és mentes a gradiens-módszer bizonyos hibáitól is. Genetikus algoritmusok akár valós, rekonfigurálható hardverelemeken való design-teszteléssel is megvalósíthatók.

Az eddig topológia-optimalizációra használt genetikus algoritmusok többnyire olyan módon kezelték a hálózatokat, amely nem gráf-alapú. Ennek folyományanként mint a mutáció, mind a keresztezés lépés során olyan véletlenszerű variációk is létrejöhettek, amelyeknek nincs értelme. Ennek oka, hogy az algoritmus teljesen véletlenszerű változtatásokat végez, nem pedig a hálózat állapota alapján változtat, a hálózatot gráfként kezelve, mint egy emberi tervezőmérnök tenné. Munkám során egy gráfalapú genetikus algoritmust dolgozok ki, amely csak logikus változtatásokat végez, és ezt az algoritmust lineáris hálózatok szintézisével tesztelem.

Abstract

While the design of digital circuits are usually done by fully automatized algorithms, analog circuit designs are still requiring some interaction with human designers. One example is when only the parameters of a circuit that is otherwise predefined by an engineer is optimized by some automated procedure.

The number of deterministic or stochastic optimization procedures used successfully for the optimization of the parameters of various circuits are growing. One can apply several constraints for these algorithms, e.g.: time or frequency domain response, stability and error depending on the precision and imperfections of the components, thermal stability, etc.

In the same time, the number of algorithms applicable for the design of the topology of a circuit is fewer and they are not that widely applied in the practice. Among these, most of the algorithms are using evolution/genetic algorithms (GAs) because they can simultaneously optimize both the topology and the parameters of the components. Also, as GAs are stochastic procedures, they are free from the shortcomings of the deterministic optimization methods. GAs can even be implemented to evaluate designs on real reconfigurable hardware.

Most of the GAs used for circuit design tend to use a kind of representation of the circuit where the information related to the graph topology is missing or vague, they usually treat the circuit as a series of numbers and manipulate these lists without the understanding of the underlying graph structure. As a result during the steps of the optimization often invalid circuits are generated, that make no sense at all. In my work I created a graph-based genetic algorithm, that allows only logical and valid changes in the circuit topology. This new approach is tested by using it for the design of different linear circuits.

1. fejezet

Bevezetés

A genetikus algoritmusok (GA-k) a biológiai evolúció modellezésén alapuló, sokféle problémakörre sikeresen alkalmazható véletlenalapú optimalizációs algoritmusok. Több népszerű programozási nyelven elérhetőek genetikus algoritmusokhoz készült könyvtárak, ilyen a `PyGAD` [1] (Python), a `genevo` [2] (Rust), az `OpenGA` [3] (C++) és a `GAlib` [4] (C++). Mind analóg, mind digitális áramköri tervezésben alkalmaztak már sikeresen GA-kat, de a digitális alkalmazásokkal ellentétben az analóg áramkörök esetében mindössze egy előre kijelölt hálózati topológia paramétereinek (komponensek értékei, tranzisztorok szélessége és csatornahossza, stb.) optimalizálása terjedt el, a kezdeti topológia megválasztása a tervezőmérnök feladata maradt.

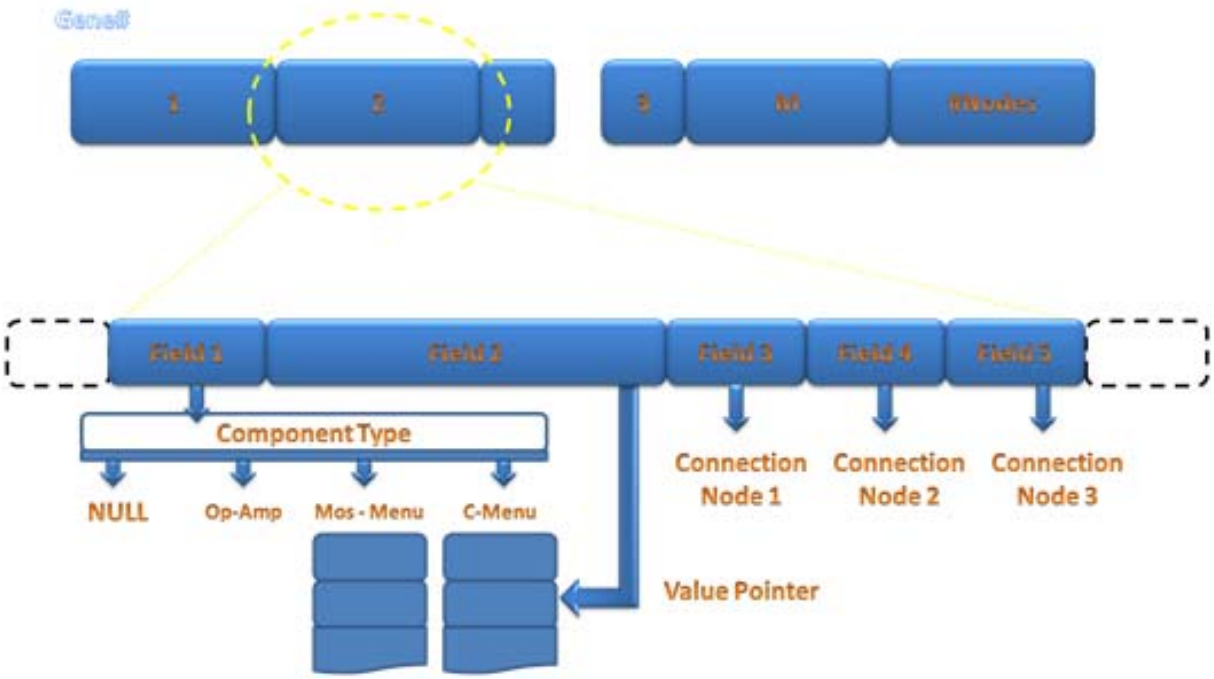
A szűkebb területre korlátozás egy lehetséges magyarázata, hogy a legtöbb GA megvalósítás fix hosszúságú numerikus kódolást használ, ami nem képes jól reprezentálni egy hálózatot. A korábbi munkák ahol GA-kat használtak hálózattervezésre, szinte mind eltértek valamilyen módon ezen limitációktól, amely általában a fix hossz megkötésének eltörlését jelentette.

Genetikus algoritmussal való topológia- vagy struktúra-optimalizálás nem új ötlet, de eddig főként digitális áramkörök tervezésére használták őket, mint [5], [6], [7], [8], de analóg áramkörökre is alkalmazták őket - [9], [10] és [11]. Egy érdekesebb hibrid [12] és [13] (továbbá a szerző további munkássága), ugyanis a szerző rekonfigurálható digitális hardveren (`FPGA`-n) implementált ezen módszerrel áramköröket, amelyeknek analóg viselkedését írta elő.

A fentebb hivatkozott hasonló témájú publikációk közül a legtöbb nem gráfalapú algoritmusokkal, hanem általános numerikus kódolásra tervezett GA dolgozott, és ennek megfelelően számcsoportokkal reprezentálták az áramkörök építőelemeit. A legtöbbször az egyes koponensek esetében (minimum) kettő szám írta le az adott komponens csatlakozásait, egy a típusát és további számok az esetleges paramétereit (komponensérték, tranzisztorszélesség, tűrés). Az 1.1 ábrán a [10]-ban használt reprezentáció látható.

A fenti reprezentáció mindenképpen használható, ámde az ezen dolgozó, általános célra tervezett GA-k algoritmusai nincsenek tudatában annak, hogy ezek egy gráfot reprezentálnak, így a mutációs és rekombinációs lépések könnyen értelmetlen gráfhoz vezethetnek. Felmerült egy olyan GA kidolgozásának ötlete, amely ezen plusz információk tudatában, a hálózatot gráfként kezelve optimalizálna. Ennek fő előnye a módosítások szűrésének lehetősége és újfajta műveletek, például egyszerűsítés végrehajtása lenne, amely remélhetőleg az optimalizálás során létrehozott áramkörök minőségének javulását jelentené.

Ezen elméleti síkon felmerült ötlet megvalósításához ezeket az algoritmusokat át kell tervezni, hogy a módosításaik során tudatában legyenek a gráf szerkezetének, ezzel növelve az egyes generált áramkörök minőségét a variáció csökkentésének rovására, azzal



1.1. ábra. Komponens reprezentáció [10]-ből

az indoklással, hogy a selejtes áramköröket az evolúciós lépések amúgy is szinte azonnal kiszűrték volna, és az ilyen ellenőrzések és új műveletek számítási igénye elhanyagolható az egyes áramkörök kiértéléséhez (szimulációjához) képest.

Munkámban egy ilyen gráfalapú genetikus algoritmust terveztem, amelyet egyszerűbb lineáris áramkörtervezési célokon teszteltem. Célom bemutatni az elméleti és gyakorlati problémákat a témában, azok néhány lehetséges megoldását adni, a kapott módszert a gyakorlatba átültetni és néhány egyszerű tervezési feladatban bemutatni ezen módszer működőképességét. Nem célom a példaként kijelölt tervezési feladatra feltétlenül optimális megoldást adni, a problémafelvetés az algoritmus működőképességének próbája.

Hasonló gráfalapú GA-k felhasználásával a vegyészetben is próbálkoztak már [14], [15], illetve gépi tanulásban is került már felhasználásra architektúra-optimalizálásra [16]. Fontos megjegyezni, hogy ezt az elnevezést ("gráfalapú genetikus algoritmus") más, eltérő jellegű megoldásokra is használják, ilyen például [17].

1.1. Gyakorlati kivitelezés során vállalt limitációk és egyszerűsítések

A gráfalapú genetikus algoritmussal való hálózattervezés elméleti lehetőségei szerteágazóak, így jelen munkámban a lehetőleg minél általánosabb elméleti elemzés és kidolgozott eredmények csupán egy szűkebb részhalmazát ültettem át a gyakorlatba. A célom a módszer működőképességének demonstrálása, nem a kijelölt próbafeladat minél jobb megoldása.

Limitációk és egyszerűsítések a megvalósított szimulációkban

- **Kétpólusokra korlátozás** - munkámban csakis kétpólusokból álló hálózatokkal foglalkoztam

- **Lineáris hálózatok tervezése** - csakis lineáris hálózatokat kezelni képes áramkörelemző programot használtam

- **Hálózatok átvitelének vizsgálata** - a módszerrel sok egyéb viselkedésre is lehet optimalizálni, például disszipáció, érzékenység, stb., de ezekkel nem foglalkoztam.

Ezen limitációk egyike sem a gráfalapú GA-k használatából ered, a korlátozások a megvalósítást jellemzik, annak elkészítését hivatottak egyszerűsíteni, hatókörét limitálni.

Ezen korlátozások mellett teszteléshez az analóg szűrők tervezésének problémakörét választottam.

A célom nem egy ezen feladatra optimalizált tervezési módszer kidolgozása, hanem egy általánosabb tervezési módszer működőképességének bemutatása egy szűkebb problémán. Koncentrált paraméterű szűrők tervezésére már rengeteg elterjedt megoldás létezik. Jelen dolgozat nem ezen módszerek kibővítését tűzte ki célul, hanem a gráfalapú genetikus algoritmusok elméleti és gyakorlati megvalósításával kísérletezik, amelyet ezen a problémakörön tesztel. A feladatkitűzés és megfogalmazás ugyan hasonlít egy hagyományos szűrőtervezési problémára, ámde a probléma megoldása jelentősen eltér az itt használt megoldásoktól, így az ezekkel való összehasonlításnak nem minden esetben lenne értelme.

2. fejezet

Elméleti áttekintés

2.1. GA részei

Genetikus algoritmusokat sokféle módon lehet megvalósítani, de néhány építőelemük mindig közös. A működés alapja az evolúció szimulálása, egy (általában fix méretű) populációval, az egyes példányok valamilyen szempont szerinti rangsorolása mellett. Minden generációt az előzőből származtatunk, amelynek módja többféle lehet:

- Elitizmus (elitism) - változtatás nélküli átvétel az előző generációból
- Mutáció (mutation) - módosítás(ok) végrehajtása az előző generáció egy példányán a mutációs operátor segítségével
- Keresztezés (crossover) - kettő (vagy több) előző generáció beli példány rekombinálása egy vagy több új példánnyá a keresztezés operátor használatával

Az egyes lépések közös tulajdonsága, hogy általában a forráspéldányokat a megfelelőségükkel (fitness function) súlyozott véletlen eloszlás szerint választják, azaz annál nagyobb eséllyel kerül bele valamilyen formában a következő generációba egy példány, minél alkalmasabb volt a kitűzött feladatra.

Ezen alkalmasságot a korábban is említett *fitness function* (továbbiakban: fitness függvény) számítja ki az egyes példányokra, és ez okból ez a legfontosabb része egy probléma GA-val való megoldásának a helyes fitness függvény meghatározása - a `PyGAD` [1] csomag például használható alapbeállításokkal rendelkezik minden más operátorral, a felhasználótól csak a fitness függvény meghatározását várja el, és ezen kívül csak az algoritmus paramétereket kell beállítani.

A szimulációnak többféle megállási feltétele is lehet, például adott számú generáció elérése, adott szimulációs idő, adott értékű fitness elérése, adott (kis) fitness javulás az utolsó N generációban, stb.

Az általam használt algoritmus:

1. Első generáció inicializálása a mutációs operátor ismételt alkalmazásának segítségével véletlenszerűen generált példányokkal
2. **Minden további generációra**
   (a) Fitness függvény kiszámítása minden példányra
   (b) Legrosszabb példányok törlése

(c) Új generáció azonos példányszámra való feltöltése mutációval, keresztezéssel és egy-az-egyben átvétellel

3. A futás során talált legjobb példány mentése (nem feltétlen a legutolsó generációból)

2.2. Reprezentáció

Munkám alapvető újítása a hálózati gráf reprezentálásának és kezelésének újragondolása. A hálózatot gráf formájában tároljuk, és gráfként manipuláljuk minden algoritmus és operátor során.

A hálózat gráfként való reprezentációjában a hálózat csomópontjai a gráf pontjai, a hálózat kétpólusai pedig a gráf élei. Ez a gráf irányított, minden kétpólusnak egyértelmű az irányítása. Ez a gráf továbbá nem egyszerű gráf, megengedi a párhuzamos éleket tetszőleges számban (multigráf). Ez a reprezentáció sokpólusokból álló hálózatok esetén is használható, ha a sokpólusokat csatolt kétpólusokkal helyettesítjük - ekkor viszont a mutációs és keresztezési operátorokban ezekre külön figyelmet kell fordítanunk.

Gráfot reprezentálni többféle módon is lehet. Külön ki kell emelni a grafikus reprezentációt, mint az ábrázolás preferált módját. A számítógépes reprezentációk közül a leggyakoribb talán a *szomszédossági lista* reprezentáció, azaz minden csomópontra az azzal szomszédos csomópontok listáját (és a kapcsolat paramétereit, tehát a kétpólus típusát és paramétereit, vagy ezekre való hivatkozást) tárolunk.

Egy másik, de általában egyszerű gráfokra használt reprezentációs séma a *szomszédossági mátrix* forma, azaz az N csomópontú mátrixot egy $\overline{\overline{M}}$ $[N \times N]$ mátrix írja le, ahol $\overline{\overline{M}}_{i;j}$ jellemzi az $i.$ és $j.$ csomópont közti kapcsolatot. Ez a forma multigráfoknál nem gyakran használt, de súlyozott egyszerű (irányított) gráfok leírására nagyon jó. Lineáris áramkörök reprezentálására alkalmas lehet, ha például a mátrix elemei $(R; L; C)$ vektorok, ahol R, L és C értelemszerűen a két csomópont közé (párhuzamosan) kapcsolt ellenállás, induktivitás és kapacitás értékét adja meg.

Egy harmadik lehetséges reprezentáció az *él-lista*, azaz felsoroljuk a gráf összes élét azok végpontjaival együtt. A korábban hivatkozott GA reprezentációk általában ezt használták.

Fontos megjegyezni, hogy egy programon belül nem feltétel hogy mindenhol ugyanazt a reprezentációt használjuk. Feladattól függően az egyes reprezentációk alkalmasabbak lehetnek a feladata megoldására, és célszerű lehet konvertálni a reprezentációk között, vagy akár többféle reprezentációt tárolni párhuzamosan, de ekkor ügyelni kell, hogy a különböző formában tárolt gráfok megegyezzenek.

2.2.1. Átalakítás külső reprezentációkba

Alapvető igény a program által kezelt belső reprezentáció oda- és visszaalakítása más reprezentációkba. Ide tartozhat például egy formátum az áramkörök tárolására fájlban, valamilyen emberek által is érthető formátum az esetleges hibák felderítéséhez, valamilyen áramkörszimulátor saját formátuma az eredmények ellenőrzéséhez és továbbhasználatához vagy éppen valamilyen grafikus formátum a kapott áramkör vizuális megjelenítéséhez.

2.3. Mutációs és keresztezési operátorok

Mivel a legtöbb eddig használt mutációs és keresztezési operátor számcsoportokon működik, szükséges ezek helyett újfajta, gráfalapon működő operátor kifejlesztése.

Ezen operátorok valamilyen leképezést valósítanak meg az előző generáció egy vagy több példányából az új generáció egy vagy több példányába, véletlenszerű alapon. A mutációs operátor szinte kizárólag egy-az-egy leképezés, tehát egyszerre egy példányból egy újat kapunk, míg a keresztezés több-az-egyhez vagy több-a-többhöz is lehet, de a leggyakoribb a $2 \to 1$ vagy a $2 \to 2$ operátor.

2.4. Követelmények a mutációs és keresztezési operátorokra

2.4.1. Építési tulajdonság

A legfontosabb tulajdonságot, amit ezen operátoroknak (és az ezeket használó algoritmusnak) teljesíteni kell, építési tulajdonságnak neveztem el.

Építési tulajdonság. *Minden általunk érdekesnek tartott hálózat felépíthető a mutációs és keresztezési operátorok ismételt alkalmazásával.*

A tulajdonság szükségessége nyilvánvaló, még ha a benne szereplő fogalmak egzakt definiálása nem is triviális, hiszen erősen alkalmazásfüggő, hogy milyen hálózatokat tartunk érdekesnek - egy szűrőtervezési problémában például nem feltétlenül akarunk ellenállásokból álló hálózatokat is vizsgálni, de egy erősítő tervezésekor kifejezetten szükséges hogy az operátorok ilyeneket is generáljanak.

2.4.2. Az építési tulajdonság következményei a szigorúbb operátorokra

Kézenfekvőnek tűnhet az operátorok korlátozása elemi szinten értelmes lépésekre, például azonos típusú komponensek soros és párhuzamos kapcsolásának tiltása, hiszen ezen kapcsolatok hálózatelméleti szempontból helyettesíthetők egyetlen, azonos típusú, de más értékű komponenssel.

Az ilyen tiltások több előnnyel is rendelkeznének, hiszen a felesleges kétpólusok sok szempontból károsak, lassítják az áramkörök elemzését, növelik a mutáció és keresztezés futásidejét, stb. Ugyanezen problémákra már több más megoldás is született, például [9]-ben a fitness függvény segítségével a komponensek minimális számára is optimalizáltak, amely így a felesleges kétpólusok eltűnéséhez is vezetett. Gráfalapú mutációs és keresztezési algoritmusok mellett azonban lehetőség nyílna már az operátorokba is beépíteni a megfelelő feltételeket.

Ezen tiltások implementálása során viszont meg kell tartani az építési tulajdonságot, ami bonyolultabb algoritmusokhoz vezet. Lehetséges hogy egy tiltás csak bizonyos építési utakat tiltana, de ezzel is csökkenti az adott eredmény elérésének valószínűségét, és ezért mérlegelni kell a tiltás implementálása előtt.

Egy lehetséges megoldás ezen problémára, amely az áramkörök egyszerűségét segíthet elérni az építési tulajdonság megtartása mellett egy új operátor bevezetése (vagy a mutációs operátor kiterjesztése ezen művelet használatára véletlen alapon), amely a gráfot bejárva megkeresi és javítja az ilyen felesleges részeket. Munkámban ezen megoldás és a [9]-ben

használt méretre optimalizálás hibridje mellett döntöttem, azaz a gráfalapú egyszerűsítő művelet (a mutációs operátorba beépítve) kiszűri az egyszerűbb eseteket (nem használt kétpólusok és csomópontok eltávolítása), de nem végez kétpólus-összevonást (például ellenállások soros eredőjének behelyettesítése), mindeközben a fitness függvény előnyben részesíti a kisebb komponensszámú áramköröket.

Implementáltam egy másik egyszerűsítő műveletet is, ami elvégzi a kétpólus-összevonásokat is. Ezen művelet gyakori használata (ideértve a kiértékelés előtti gyors egyszerűsítést is) viszont negatívan érintette az algoritmus futását, csak lassabban javult a fitness-érték a generációk között, adott számú generáció után a végső legjobb fitness érték rosszabb lett mint enélkül. Ennek oka valószínűleg a művelet diverzitás-csökkentő hatása - egy lehetséges jó irányba tett lépéseket az egyszerűsítés eltöröl. Az egyszerűsítések hatására minden tovább nem egyszerűsíthető áramköri topológia konvergenciapont a lehetséges topológiák terében, amely az optimumra való konvergenciát könnyűszerrel lelassíthatja. Az egyszerűbb áramkörökben a nem egyszerűsített áramkörökhöz képest kevesebb ponton lehet módosítani, ami szintén a variációk csökkenéséhez vezet. Az egyszerűsítésnek eközben fontos pozitív hozadéka az egyszerűbb áramkörök jóval gyorsabb elemzésének lehetősége, így be kell állatani az egyszerűsítés (korlátozás) és a variálás egyensúlyát.

2.5. A használt mutációs operátor

A mutációt célszerű elemi műveletek formájában, azok között véletlenszerűen döntve implementálni. Az elemi lépések között mindenképpen kell lennie a gráfot élekkel és csomópontokkal bővítő műveletnek, illetve élek és csomópontok eltávolítására is kell lehetőség. Mivel az élek információt hordoznak, ezen információ módosítására is kell lehetőség. A módosítások át is fedhetik egymást részben, nem probléma ha egy lehetséges állapot több különböző módon is létrejöhet, sőt, kifejezetten előnyt jelenthet.

A példaprogramban hat fajta elemi mutációs műveletet implementáltam, amelyek között minden mutációs lépésben súlyozott véletlenszerű alapon döntök. A súlyozás többféle lehet, akár a generáció sorszámától függően (például az evolúció kezdeti szakaszában több és szerteágazóbb mutáció, míg a későbbi szakaszában kisebb módosítások előnyben részesítése).

Az elemi műveleteket fel kell készíteni azon esetre is, amikor a végrehajtásuk lehetetlen. Ezt úgy oldottam meg, hogy az elemi műveleteim maximum egyszeri véletlenszerű választást végeznek, ha ez nem vezet érvényes választásra (akár azért mert nincs érvényes választás, akár más okból), akkor módosítás nélkül visszatérnek, ekkor a mutációs operátor új műveletet választ és újrapróbálkozik, amíg a próbálkozások száma el nem ér egy megadott határértéket.

2.5.1. Kétpólus beszúrása két csomópont közé, avagy élbeszúrás

Két csomópont véletlenszerű kiválasztása, és egy véletlenszerűen generált új alkatrész beszúrása a két csomópont közé. Ellenőrzi, hogy a két választott csomópont ne egyezzen meg, így megelőzve az *intronok* (hurokélek a hálózati gráfban) keletkezését. A generált komponensek típusát limitálni lehet.

2.5.2. Kétpólus beszúrása egy másikkal soros kapcsolásban, avagy csomópont-beszúrás

Véletlenszerű kétpólust választ és generál egy új komponenst. A korábbi kétpólust lecseréli az új kétpólus és a korábbi soros eredőjére, egy új középső csomópont beszúrásával. A két lehetséges soros eredő közül véletlenszerűen választ.

2.5.3. Kétpólus cseréje más kétpólusra

Egy véletlenszerűen kiválasztott kétpólust egy újonnan generált kétpólusra cserél.

2.5.4. Kétpólus értékének módosítása

Egy véletlenszerűen választott kétpólust lecserél egy újonnan generált, azonos típusú kétpólusra.

Mind itt, mind az előző műveletekben (2.5.1, 2.5.2 és 2.5.3) a komponensértékek az E48-as értéksorból kerülnek kiválasztásra, a hatványkitevő alkatrésztípustól függő véletlen értéke mellett.

2.5.5. Kétpólus törlése

Töröl egy véletlen alapon kiválasztott kétpólust, és véletlenszerűen helyettesíti rövidzárral (ennek esélye paraméter lehet, pl. 50%).

2.5.6. Hálózat egyszerűsítése

A korábban (2.4.2) említett gráfegyzerűsítő művelet. Összevonja a rövidzárral összekötött csomópontokat egy csomóponttá (törölve a köztük levő komponenseket), és törli az olyan csomópontokat amelyekhez csakis egy kétpólus csatlakozik (a kétpólussal együtt, folytatva a törlést ha ezzel új "lebegő" csomópont keletkezik).

Egy másik ilyen műveletet is implementáltam, amely elvégzi kétpólusok összevonását is. Ahogy 2.4.2-ben is írtam, ezen művelet használata negatív következményekkel is járhat az algoritmus futására nézve, rosszabb eredményekre vezethet.

2.5.7. A mutációs operátok és az intronok

Az intronok és egyéb hasonló, elméletileg is haszontalan komponensek keletkezését már a mutációs és keresztezési szakaszban is ki lehet szűrni. Ezek rendszeresen problémát jelentenek a hasonló célú optimalizációk során, mivel felesleges erőforrásnak számítanak. Megelőzésükre több módszer is született, például [9]-ben az áramkör méretének optimalizálásával próbálták megelőzni őket. A gráfalapú megközelítés egyik előnye, hogy ezen felesleges alkatrészek keletkezése direkt módon megelőzhető.

2.6. A használt keresztezési operátorok

Sokféle keresztezési operátor implementálható. Munkámban kétfajta keresztezési operátort is implementáltam, de végül a véletlenszerű csere algoritmusát (2.6.1) alkalmaztam.

A keresztezés egyik problémája, hogy egymástól igen eltérő topológiájú hálózatokra is értelmezhető algoritmust kell terveznünk.

2.6.1. Véletlenszerű csere

Ez az algoritmus kizárólag a két hálózat komponensein működik, a hálózat gráfjával nem törődik, csak a két hálózat között cserél fel kétpólusokat. A felcserélt kétpólusok száma véletlen alapon függ a két hálózat kétpólusai számának minimumától.

2.6.2. Összefüggő csere

Ez egy bonyolultabb algoritmus, amely csomópont-csomópont alapon cserél kétpólusokat. Először egy-egy pontból kiindulva egy-egy fát térképez fel a két gráfban, feljegyezve és egymásnak megfeleltetve a két gráfban a megtalált pontokat (mind a két hálózatban ugyanannyi csomópontot kijelölve). Ezután mindkét gráfban megkeres minden olyan kétpólust, amely csakis ezen ponthalmazon belül rendelkezik kapcsolatokkal, majd az ezekből alkotott részgráfokat cseréli ki a két gráf között, a feltérképezés során felépített megfeleltetésnek megfelelően.

2.7. Fitness számítása

Egy GA legfontosabb része a problémát leíró fitness függvény, ennek hibás megválasztása jó GA mellett is szuboptimális vagy hibás eredményekre vezethet. Általános elméleti módszer ezen komponens megtervezésére nem adható, de gyakran használható hibajellegű mennyiségek összegzése és a hiba minimumára való optimalizálás. A gyakorlati megvalósítás során én is ezt használtam, a fitness-függvény minimalizálására törekedett a programom, és a példányok kiválasztásánál a kapott pontszám reciprokával súlyoztam a populációt. A fitnesst így a hibával azonosítva a "jobb" fitness a kisebb értéket jelenti, és a GA célja a fitness minimalizálása.

3. fejezet

Gyakorlati kivitelezés

Ahogy 1.1-ben is írtam, az általános elmélet csakis egy szűkebb szeletét implementáltam a gyakorlatban. A program Rust nyelven íródott, a következő csomagok felhasználásával:

- `dyn-clone 1.0.9` - interfész (trait object) másolása
- `rand 0.8.4` - véletlenszámok generálása
- `regex 1.6.0` - mintaillesztés
- `bimap 0.6.2` - kétirányú egy-az-egyhez adatszerkezet
- `rayon 1.5.3` - párhuzamos végrehajtás
- `kdam 0.2.7` - folyamatjelző (progressbar)
- `duct 0.13.5` - külső program hívása

A programkód elérhető a `https://github.com/Sasszem/graph-ga` címen.

3.1. Áramkörök elemzése

3.1.1. Saját áramkörelemző modul használata

Az áramkörök elemzéséhez először egy saját, korábban készült lineáris áramkörelemező modul Rust nyelven történő reimplementációját használtam fel, amely a hálózat gráfjából felírja a hálózati egyenletek teljes rendszerét, majd a kapott mátrix invertálásával számítja ki a válasz és gerjesztés arányát. Frekvenciatartománybeli viselkedés számításához adott frekvencián kiszámítja a dinamikus kétpólusok komplex impedanciáit, majd azokat egy komplex együtthatós egyenletrendszerré alakítja, így az átviteli együttható pontonként való kiszámításával számítja az átviteli függvényt.

3.1.2. ngspice

A saját megoldással a program lassú volt és néhány esetben pontatlan eredményekre is jutott, így a saját elemzőprogramot lecseréltem `ngspice`-ra, amely egy bevált áramkörszimulátor. A programom a belső reprezentációból az áramköröket SPICE formátumba konvertálja, azt átadja az `ngspice`-nak és a szimuláció eredményét dolgozza fel. Míg kisebb áramkörökre a saját megoldás jóval gyorsabb volt, úgy nagyobbakra egy nagyságrendnyi sebességnövekedést értem el ezzel a módszerrel.

3.2. Többlépéses evolúció

Egy lehetséges és viszonylag gyakran használt módosítás a GA alapműködésén a futás két vagy több fázisra osztása, a különböző fázisokban eltérő fitness-függvényt, mutációs vagy keresztezési operátort használva.

Az én megvalósításomban két lépésre osztottam a futást amelyek között főleg a mutációs operátorok adták a különbséget. Az első fázisban nagyobb módosításokat célul véve határoztam meg a mutációs műveletek halmazát és azok súlyozását, míg a második fáziban kisebb módosításokra terveztem, nagyobb súlyt kaptak az értéket módosító és egyszerűsítő műveletek mint az új alkatrészek beszúrása.

4. fejezet

Felmerült gyakorlati problémák és megoldásaik

4.1. A környezet hatása

A tervezni kívánt áramkörök valamilyen környezetbe lesznek illesztve, és ezen környezet hatását is figyelembe kell venni az elemzés során, az áramkör gráfjába ezen külső kapcsolatokat is bele kell venni. Ezt én speciális kétpólus-típusok implementálásával tettem meg, amelyek rendelkeznek egy jelzéssel, hogy a mutációs és keresztezési operátorok számára ezen kétpólusok módosítása korlátozott vagy tiltott. Implementáltam egy jelzési rendszert csomópontokra is, amely hasonlóan tiltja a megjelölt csomópontok módosítását. Az új kétpólusok hasonló vagy azonos SPICE parancsokat generálnak mint a módosítható verzióik.

A mutációs és keresztezési operátorokat ellenőrizni kellett, hogy ezen csomópontok és kétpólusok módosítása biztosan ki van-e zárva.

4.2. Kétpólusok gyakorlati reprezentációjának kérdése

A programban gyakran kell többféle, közös interfészt megvalósító képtólust együtt kezelni a közös interfész alapján. Ilyen esetekben az objektumorientált megközelítés és egy heterogén kollekció használata szokott lenni a használt megoldás, de bizonyos esetekben szükség lehet a közös interfész mögötti konkrét típus ismeretére (pl. soros eredő csak akkor képezhető két kétpólusból, ha azok ugyan olyan típusúak), amit ez a tervezési minta nem enged meg. A minta kiegészíthető ilyen lehetőséggel, de ez nem eredményez átlátható kódot. Egy másik minta, a funkcionális programozásban gyakran használt összeg-típus (sum type) ezen problémáktól mentes, de más, saját problémákkal rendelkezik. A két felmerült módszer közül én az elsőt (közös interfészt implementáló objektumok heterogén kollekciója) választottam.

4.3. A fitness-function megfelelő meghatározása

A szűrők tervezési feladatait olyan fitness függvénnyel írtam le, amely adott frekvenciákon kiszámította a rendszer átvitelét a megfelelő gerjesztés és lezárás mellett, majd a kapott értékekből hibát számított egy önkényesen megválasztott (és nem is biztos hogy fizikailag realizálható) átviteli függvényt referenciának véve. A hibaszámítás a 4.1 szerint történt, a konstansok önkényes megválasztásával. $f_1 \dots f_N$ azon frekvenciákat jelöli, amelyeken az

áramkör viselkedését (például szimulációval) megvizsgáltuk, célszerűen ezen frekvenciákát logaritmikus felvéve, $H(s)$ pedig az áramkör átviteli függvényének értéke az $s = j2\pi \cdot f$ helyen, $|H(s)|^{\mathrm{dB}}$ pedig ennek abszolútértéke dB-ben. Az átviteli függvényt a feszültségforrás feszültségéről mint gerjesztésről a lezáró ellenállás feszültségére mint válaszra értelmeztem. Látható, hogy a függvény más módon számítja a hibát áteresztő- és zárósávban. A zárósávban egy maximális érték (jelen példában $-46\,\mathrm{dB}$) túllépése számít hibának, míg az áteresztősávban egy előírt értéktől ($-6\,\mathrm{dB}$) való eltérés, amit az eltérés mértékétől függő konstanssal szorzunk (ezzel meghatározva egy toleranciasávot). Az amplitúdómenetből származó hibán kívül az áramkör komponenseinek számának konstansszorosát is egy hibakomponensnek vettem, ezzel előnyben részesítve az egyszerűbb áramköröket. A konstansok meghatározása önkényesen történt, a beállításuk nem triviális, próbálkozással történhet.

$$
\begin{aligned}
err &= |err_{kp} + \sum_{f \in f_1 \dots f_N} err_f(f)| + 1 \\
err_{kp} &= 200 \cdot \#\{\text{áramkör komponenseinek száma}\} \\
err_f(f) &= \begin{cases}
\max\left(|H(s)|^{\mathrm{dB}} + 46\,\mathrm{dB}; 0\right) \cdot 100 & \text{zárósávban} \\
\begin{cases}
\left||H(s)|^{\mathrm{dB}} - 6\,\mathrm{dB}\right| \cdot 100, & \text{ha } \left||H(s)|^{\mathrm{dB}} - 6\,\mathrm{dB}\right| < 0.5\,\mathrm{dB} \\
\left||H(s)|^{\mathrm{dB}} - 6\,\mathrm{dB}\right| \cdot 5000, & \text{egyébként}
\end{cases} & \text{áteresztősávban} \\
0 & \text{egyébként}
\end{cases}
\end{aligned}
\tag{4.1}
$$

A rosszul megadott fitness függvény sokszor okozott váratlan vagy rossz eredményeket, például aluláteresztő szűrő tervezésekor egy fordítva felírt relációsjellel már sáváteresztőt specifikáltam, amely természetesen nem a várt eredménnyel járt.

Különösen kell figyelni a fitness függvény implementációja során hogy semmiképpen se keletkezzen negatív, NaN vagy Inf érték, mivel ezekkel a program hibás eredményre jut vagy össze is omolhat, hiszen ezek minden szempontból értelmetlen értékek.

4.4. Futási sebesség

A program futási sebessége sok tényezőtől függ, mint például a szimulálni kívánt generációk száma, az egy generáción belüli populációméret, az egyes áramkörök komplexitása, stb. Megfigyelés szerint a legtöbb időt az áramkörök szimulációja emészti fel, így itt lehet a legnagyobb sebességnövekedést elérni.

Tapasztalat alapján ha egy egyszerű feladatra sikerül egyszerű áramköröket generálni, akkor a futás gyors lesz, de bonyolultabb feladat esetén ehhez képest akár egy nagyságrendnyi eltérés is lehet. Részben ezzel magyarázható az a tapasztalati megfigyelés, hogy egy futtatás során egyre kevesebb ideig tart kiszámolni egy-egy generációt.

A saját elemzőprogram lecserélése `ngspice`-ra jelentős sebességnövekedéssel járt, akárcsak a szimulációs lépés (fitness meghatározása) párhuzamosítása. Ezen lépés triviálisan párhuzamosítható, mivel a populáció minden áramkörére külön kell kiszámolni a fitness függvényt, ezek nem hatnak egymásra, nincs közös hozzáférésű adat. A párhuzamosítást a `rayon` csomaggal végeztem, amely minimális módosítást követelt csak meg a program többi részében, és jelentősen növelte a processzor kihasználtságát és kb. egy nagyságrenddel csökkentette a szimuláció futásidejét.

4.5. A GA finomhangolása

A genetikus algoritmus beállítása során a fitness függvényen túl is rengeteg paramétert lehet állítani, még ha az egyes operátorok azonosak is maradnak. Példa erre a populációméret, a generációk száma, az elitizmus, mutáció és keresztezés aránya, az eldobott leggyengébb példányok száma, stb. Ezen paraméterek beállítása kísérletezés alapján történt, amely nem mindig vezetett jó eredményre.

Különösen nehéz meghatározni az egyszerűsítések mértékét és mennyiségét, mivel minden egyszerűsítés korlátozzal a GA módosítási lehetőségeit, amely szembemegy a GA működésével, így a gyakran használt egyszerűsítéseknek csak a teljesen értelmetlen dolgokat szabad szűrni, és optimumot kell találni a további egyszerűsítések használatában.

4.6. Eredmények és fitness szórása

Az általam használt paraméterezéssel a GA nem mindig volt stabil, két, paramétereiben megegyező futtatás esetenként jelentősen eltérő eredményekhez vezetett, nagyságrendnyi különbségekkel a futás során elért legjobb fitness-értékkel.

5. fejezet

Eredmények

Több különböző feladat elé állítottam tesztelésképpen a programomat, és azokat változó sikerrel oldotta meg. A kevésbé jó eredmények igen gyakran a rosszul megválasztott fitness függvénynek tudhatók be, de a GA egyéb paramétereinek beállítása sem optimális.

A program működését két példán keresztül mutatom be.

5.1. Beszédátviteli aluláteresztő szűrő tervezése

5.1.1. Specifikáció

Beszédátvitelhez kell aluláteresztő szűrőt tervezni, kapacitív és induktív elemekből. Áteresztősáv $0 - 3.4\,\text{kHz}$, zárósáv $4 - 20\,\text{kHz}$. A szűrő $50\,\Omega$-os rendszerbe illeszkedik, ennek megfelelően kell a környezetet szimulálni. A belső ellenállás és lezárás viszonyából adódóan az áteresztpsávan $-6\,\text{dB}$-es feszültségerősítést várunk el a forráshoz képest, míg a zárósávban $-46\,\text{dB}$ az elvárt maximális erősítés.

A feladatot a 4.1 szerinti hibafüggvénnyel specifikáltam, amely a fenti szöveges megfogalmazáson túl extra követelményeket is támaszt, mint például az áteresztősávban elvárt maximális eltérés az előírt értéktől. A feladatot többször is megoldottam a paraméterek és algoritmus változtatása mellett. A programkód és néhány kapott eredmény a `https://github.com/Sasszem/graph-ga/tree/main/tasks/voice_lpf` mappa alatt érhető el. Az alábbiakban az itt 1-es számmal jelölt futtatás adatait prezentálom.

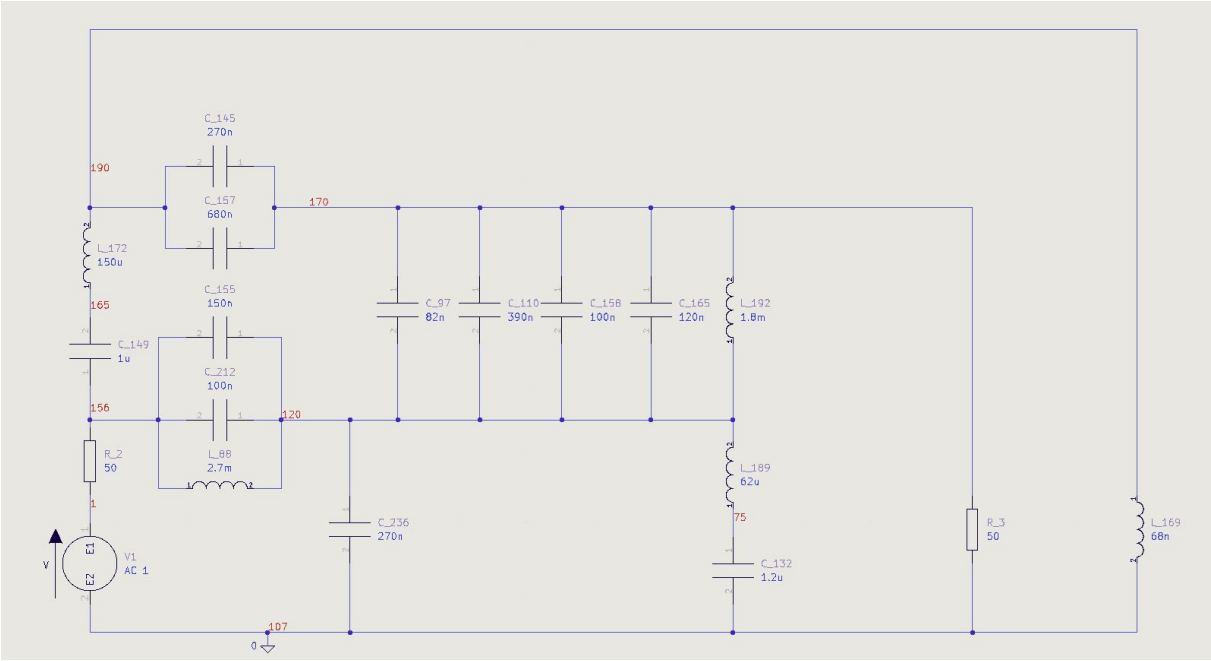
Ennél a futtatásnál a szűrő csak a $20\,\text{Hz} - 20\,000\,\text{Hz}$-es tartományon volt specifikálva, és a mutációs műveletek között nem állt rendelkezésre a sorosan vagy párhuzamosan kapcsolt komponenseket összevonó művelet.

A futás néhány paramétere: 500 generáció 1000-es populációmérettel futtatva. A futás ideje $2675\,\text{s}$, azaz 45 perc volt. A végső áramkör SPICE leírása a következő:

```
* ----------------
* Fitness: 8304.54336048041
* SIMULATED CIRCUIT
L_192 170 120 0.0018000000000000002
C_157 190 170 0.00000068
L_88 156 120 0.0027
C_155 120 156 0.00000015
R_3 170 107 50
C_145 170 190 0.00000027
C_158 120 170 0.0000001
```

```
V_1 1 107 AC 1
C_132 107 175 0.0000012
L_189 175 120 0.000062
C_110 120 170 0.00000038999999999999997
C_236 107 120 0.00000027
C_165 170 120 0.00000012
L_169 190 107 0.000000068
R_2 156 1 50
C_97 120 170 0.000000081999999999999999
C_149 156 165 0.000001
L_172 165 190 0.00015000000000000001
C_212 156 120 0.0000001
```

Az áramkör R_3 jelű ellenállása a lezárás. Az áramkör 11 kapacitást és 5 induktivitást tartalmaz, de a kapacitások összevonhatók összesen 6 kapacitássá. Az áramkör az 5.1, átviteli függvényének abszolútértéke pedig az 5.2 és az 5.3 ábrákon látható. Látható, hogy a specifikált tartományban a legnagyobb eltérés az átmeneti sáv előírtnál nagyobb szélessége, de az előírt tartományon kívül a viselkedés nem követi egy ideális aluláteresztő szűrő viselkedését.

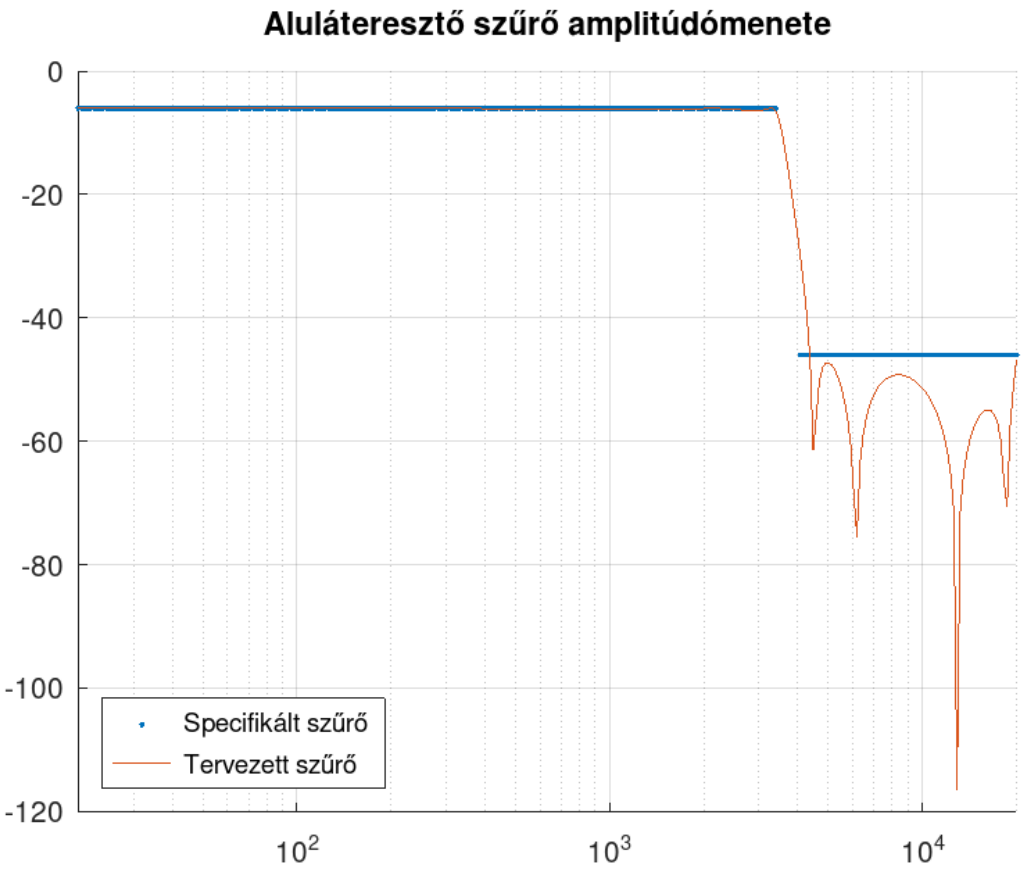


5.1. ábra. A tervezett aluláteresztő szűrő KiCad-ben

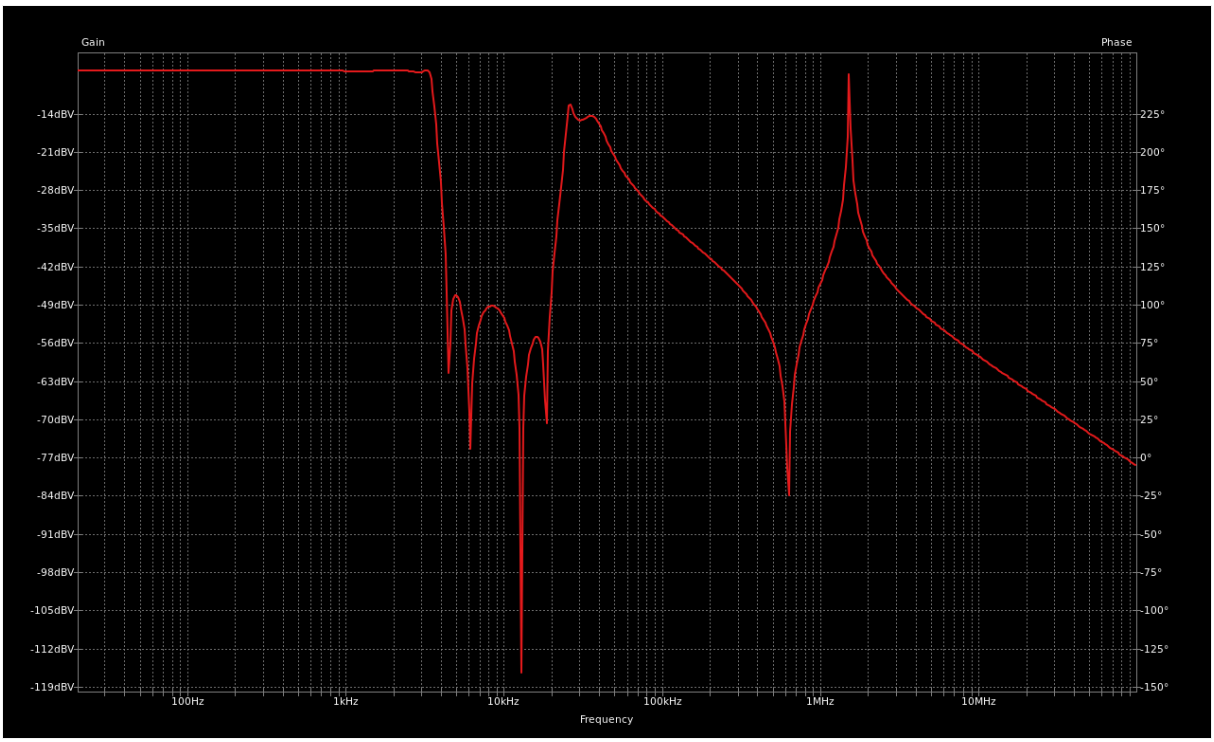
5.2. AM sávú sáváteresztő szűrő

5.2.1. Specifikáció

A műsorszóró AM sávra kell az előzőhöz hasonlóan sáváteresztő szűrőt tervezni. Áteresztősáv $525 - 1606.5\,\text{kHz}$, zárósáv $0 - 450\,\text{kHz}$ és $1700 - 5000\,\text{kHz}$. A szűrő szintén $50\,\Omega$-os, és az előzőekhez hasonlóan áteresztősávban $-6\,\text{dB}$-es feszültségerősítés van előírva, míg zárósávban ugyanez $-46\,\text{dB}$.



5.2. ábra. A tervezett aluláteresztő szűrő amplitúdómenete



5.3. ábra. A tervezett aluláteresztő szűrő amplitúdómenete nagyobb frekvenciasávban (20 Hz to (numerical range) 100 000 000 Hz)

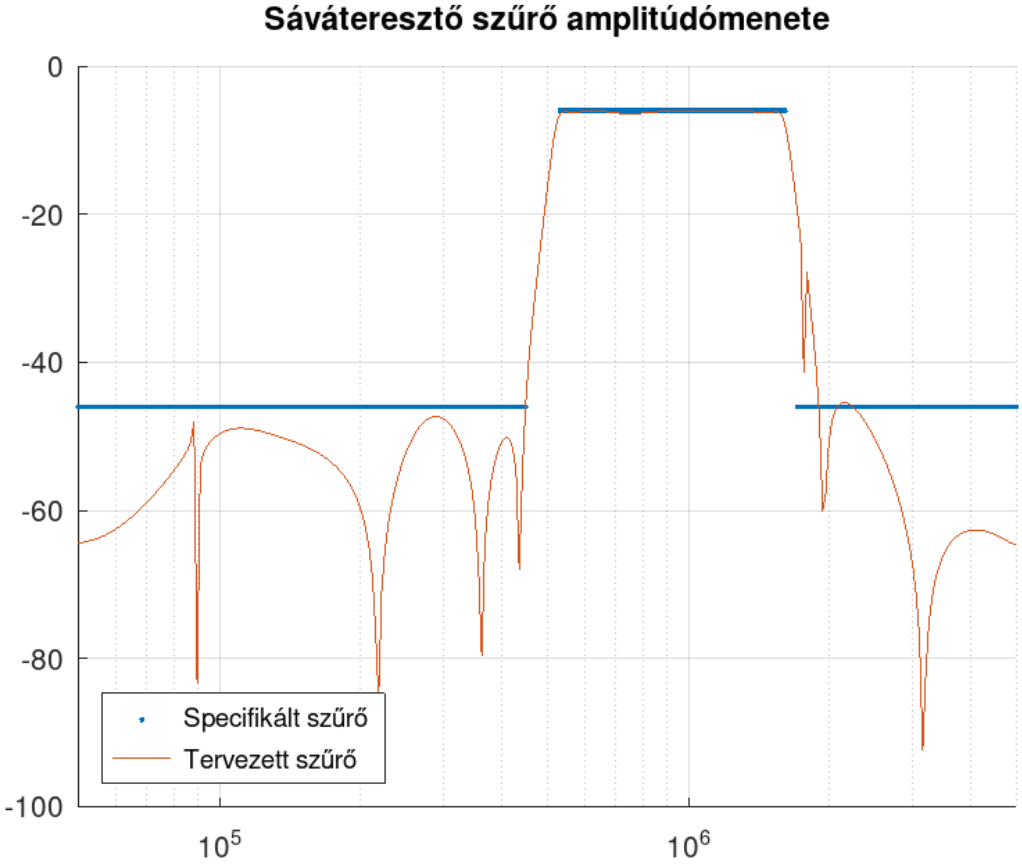
A korábbi hibaegyenletet kiegészítettem egy extra taggal, amely a kapacitások és induktivitások számának abszolút különbségével arányos:

$$err_{unbalance} = |\#\{\text{kapacitások száma}\} - \#\{\text{induktivitások száma}\}| \cdot 800$$

Ennél a futtatásnál már rendelkezésre állt a sorosan vagy párhuzamosan kapcsolt komponenseket összevonó művelet.

A futás néhány paramétere: 500 generáció 1000-es populációmérettel futtatva. A futás ideje 4683 s, azaz 1 óra 20 perc volt. A végső áramkör SPICE leírása a `https://github.com/Sasszem/graph-ga/blob/main/tasks/am_bpf/res/am_bpf_checkpoint.csv` fájlban található.

Az áramkör 42 kapacitást és 40 induktivitást tartalmaz, így realizációja nem praktikus. Az áramkör átviteli függvényének abszolútértéke az 5.4 ábrán látható. Látható, hogy a legnagyobb eltérés a specifikálthoz képest itt is a szélesebb átmeneti tartomány.



5.4. ábra. A tervezett sávátersztő szűrő amplitúdómenete

5.3. Az eredmények értékelése

A program képes a specifikációknak megfelelő vagy azt jól közelítő áramkörök tervezésére, ami a módszer működőképességét bizonyítja. Mindenképpen fontos kiemelni hogy a módszer csak a specifikációt igyekszik követni, a nem specifikált tartományokon az eredmény sem lesz specifikált, nem lehet az eredményeket kiterjeszteni, igen fontos a pontos specifikáció. A tervezett szűrők összehasonlításának más módszerrel tervezett szűrőkkel

nincs sok értelme, mivel más módon lettek spefifikálva a követelmények és teljesen más tervezési metódussal készültek. A módszert csak más GA-val van értelme direkt módon összehasonlítani, és ez az összehasonlítás csak azonos probléma és egyező körülmények között történhet. Ilyen összehasonlítást nem végeztem, mivel célom a módszer működőképességének ellenőrzése volt.

5.4. További lehetőségek

5.4.1. A gyakorlati megvalósításban használt limitációk elhagyása

Ezen módszerrel nem csak lineáris kétpólusokból álló áramkörök tervezhetők, a módszer minden további nélkül kiterjeszthető nemlineáris komponensekre, és némi körültekintéssel többpólusokra is. A többpólusokra való kiterjesztés olyan extra követelmények beszúrását követeli meg a mutációs és keresztezési operátoroktól, ami megelőzi a sokpólusok szétválását és így érvénytelen állapotra jutását.

5.4.2. Áramkörök exportálása egyéb formátumokba

Jelenleg a program kizárólag egy saját szöveges formátumba és `SPICE netlist` formátumba tudja az áramköröket exportálni. Célszerű lenne más formátumokat is implementálni, például valamilyen átamkörtervező program schematic formátumát, illetve valamilyen grafikus formátumot, esetleg LaTeX-et. Egy gyakorlati nehézség ezekkel kapcsolatban, hogy a legtöbb formátum valamilyen síkbeli elrendezést is tartalmaz, ami az itt használt reprezentációban egyáltalán nincs, így ezt külön kell előállítani, és egy általános multigráf síkbarajzolása nem triviális feladat.

5.4.3. Valós hardveren való kiértékelés

[9], [13] és [12] is felhasznált rekonfigurálható hardvert az áramkörök kiértékelésére. Gráfalapú reprezentációval is lehetséges így kiértékelni az áramköröket.

5.4.4. Másfajta optimalizálási feladatok

Munkámban csakis átviteli függvényre és komponensszámra végeztem optimalizálást, de a módszerrel természetesen más jellegű feltételek is szabhatók - időtartománybeli viselkedés, teljesítményfelvétel, alkatrészek értékeire való érzékenység (ideértve a hőmérséklet hatását), stb. Egy példányon több különböző szimuláció vagy elemzés is futtatható, illetve ugyanaz a szimuláció lefuttatható többször is változó paraméterekkel (pl. érzékenységvizsgálathoz).

6. fejezet

Összegzés

Munkámban felvetettem a gráfalapú belső reprezentációt használó GA-k használatát koncentrált paraméterű hálózatok tervezésére. Körüljártam elméleti síkon a reprezentáció és a működés kérdéseit és lehetséges konkrét implementációkat is adtam a fontosabb műveletekre (mutáció, keresztezés, fitness-számítás). Az elméletben kidolgozott módszert a gyakorlatba is átültettem egy Rust nyelven írt program formájában, amelyekkel néhány szűrőáramkört is megterveztem, demonstrálva hogy a módszer képes a specifikációknak megfelelő vagy azt jól közelítő áramköröket tervezni. Demonstráltam továbbá hogy a módszer továbbra is rendelkezik a GA-k általános problémáival, a paramétereket hasonlóan kell finomhangolni, de új problémaként megjelent az egyszerűsítések és a diverz variálás egyensúlyának megtalálása is. Felvetettem néhány további kutatási-fejlesztési irányt a témában, illetve néhány további alkalmazási területet és módszert a vizsgált szűkebb témakörön túl

Irodalom

[1] *PyGAD - Python Genetic Algorithm!* 2022. URL: https://pygad.readthedocs.io/en/latest/ (elérés dátuma 2022. 10. 23.).

[2] *innoave/genevo at v0.4.0.* 2022. URL: https://github.com/innoave/genevo/tree/v0.4.0 (elérés dátuma 2022. 10. 23.).

[3] Arash Mohammadi és tsai. „OpenGA, a C++ Genetic Algorithm Library". *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on.* IEEE. 2017, 2051–2056. old.

[4] *GAlib: Matthew's Genetic Algorithms Library.* 2007. URL: http://lancet.mit.edu/ga/ (elérés dátuma 2022. 10. 23.).

[5] K.H. Chong és tsai. *Digital Circuit Structure Design via Evolutionary Algorithm Method.* 2007. jan. DOI: 10.3923/jas.2007.380.385. URL: http://dx.doi.org/10.3923/jas.2007.380.385.

[6] Parisa Soleimani és tsai. „Using Genetic Algorithm in the Evolutionary Design of Sequential Logic Circuits". (2011). DOI: 10.48550/ARXIV.1110.1038. URL: https://arxiv.org/abs/1110.1038.

[7] Carlos Coello, A. Christiansen és Arturo Hernandez-Aguirre. „Using Genetic Algorithms to Design Combinational Logic Circuits". 6 (1996. jan.). DOI: 10.1007/978-3-7091-6492-1_73.

[8] Xuesong Yan és tsai. *Electronic Circuit Automatic Design Based on Genetic Algorithms.* en. 2011. DOI: 10.1016/j.proeng.2011.08.555. URL: http://dx.doi.org/10.1016/j.proeng.2011.08.555.

[9] S. Ando és H. Iba. „Analog circuit design with a variable length chromosome". *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512).* 2. köt. 2000, 994–1001 vol.2. DOI: 10.1109/CEC.2000.870754.

[10] Badar K Khan és Yaser A Khalifa. „An evolutionary method for analog circuits optimization utilizing Mosfet-C filters". *2011 18th IEEE International Conference on Electronics, Circuits, and Systems.* 2011, 362–365. old. DOI: 10.1109/ICECS.2011.6122288.

[11] Miri Weiss Cohen, Michael Aga és Tomer Weinberg. „Genetic Algorithm Software System for Analog Circuit Design". *Procedia CIRP* 36 (2015). CIRP 25th Design Conference Innovative Product Creation, 17–22. old. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2015.01.033. URL: https://www.sciencedirect.com/science/article/pii/S2212827115000360.

[12] Adrian Thompson és Clive Davidson. *Creatures from primordial silicon - Let Darwinism loose in an electronics lab and just watch what it creates. A lean, mean machine that nobody understands. Clive Davidson reports.* 1997. nov. URL: https://www.newscientist.com/article/mg15621085-000-creatures-from-primordial-silicon-let-darwinism-loose-in-an-electronics-lab-and-just-watch-what-it-creates-a-lean-mean-machine-that-nobody-understands-clive-davidson-reports/.

[13] Adrian Thompson. „Silicon Evolution". *Proceedings of the 1st Annual Conference on Genetic Programming*. Stanford, California: MIT Press, 1996, 444–452. old. ISBN: 0262611279.

[14] Jan H. Jensen. „A graph-based genetic algorithm and generative model/Monte Carlo tree search for the exploration of chemical space". *Chem. Sci.* 10 (12 2019), 3567–3572. old. DOI: 10.1039/C8SC05372C. URL: http://dx.doi.org/10.1039/C8SC05372C.

[15] Nathan Brown és tsai. „A Graph-Based Genetic Algorithm and Its Application to the Multiobjective Evolution of Median Molecules". *Journal of Chemical Information and Computer Sciences* 44.3 (2004). PMID: 15154776, 1079–1087. old. DOI: 10.1021/ci034290p. eprint: https://doi.org/10.1021/ci034290p. URL: https://doi.org/10.1021/ci034290p.

[16] Fei Qi és tsai. *DarwinML: A Graph-based Evolutionary Algorithm for Automated Machine Learning*. 2019. DOI: 10.48550/ARXIV.1901.08013. URL: https://arxiv.org/abs/1901.08013.

[17] Daniel Ashlock, Mark Smucker és John Walker. „Graph based genetic algorithms". 2. köt. 1999. febr., 1368 Vol. 2. ISBN: 0-7803-5536-9. DOI: 10.1109/CEC.1999.782611.