FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Komprese konektivity trojúhelníkových sítí se známou geometrií s využitím neuronových sítí

Viktor Havlík

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Komprese konektivity trojúhelníkových sítí se známou geometrií s využitím neuronových sítí

Viktor Havlík

Vedoucí práce
Ing. Filip Hácha

PLZEŇ 2024

Citace v seznamu literatury:
HAVLÍK, Viktor. *Komprese konektivity trojúhelníkových sítí se známou geometrií s využitím neuronových sítí*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Filip Hácha.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Viktor HAVLÍK**
Osobní číslo:	**A21B0124P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Komprese konektivity trojúhelníkových sítí se známou geometrií s využitím neuronových sítí**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se metodou pro kompresi konektivity trojúhelníkových sítí se známou geometrií, založenou na řazení kandidátních vrcholů na základě lokálních vlastností geometrie.
2. Navrhněte alternativní metodu pro řazení kandidátních vrcholů s využitím umělé neuronové sítě.
3. Navrženou metodu implementujte a ověřte její vlastnosti na volně dostupných datech.
4. Výsledky navržené metody porovnejte s již existující metodou. Výsledky experimentů důkladně zdokumentujte.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Filip Hácha**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 1. května 2024

. . . . . .
Viktor Havlík

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Tato bakalářská práce se zabývá tématem komprese konektivity trojúhelníkových sítí a problematikou s ní spojenou. Práce navazuje na již existující, velice efektivní metodu založenou na principu kandidátních vrcholů a dává si za cíl její další zefektivnění. Toho se snaží dosáhnout za pomoci strojového učení. Práce se tak zaměřuje na několik klíčových částí původní metody a ty nahrazuje modelem umělé neuronové sítě. Cílem je zjistit, jak efektivně je model schopen, oproti statickým vzorcům výchozí metody, predikovat konektivitu trojúhelníkové sítě a zda-li tento přístup může vést k požadovanému zlepšení. Jak získané výsledky naznačují, je tomu skutečně tak a metoda s využitím neuronové sítě byla schopna dosáhnout signifikantního zlepšení.

Abstract

This bachelor's thesis addresses the topic of triangle mesh connectivity compression and the related issues. The work builds upon already existing, highly efficient method based on the principle of candidate vertices and aims for its even further optimization. It attempts to achieve this through the use of machine learning. Thus, the thesis focuses on several key parts of the existing method, which it then substitutes with a model of the neural network. The goal is to determine how effectively the model can predict triangle mesh connectivity compared to the static formulas used by the reference method and whether this approach can lead to the desired improvement. As the results indicate, this is indeed the case, and the method using the neural network was able to achieve significant improvement.

Klíčová slova

komprese • konektivita • neuronová síť • trojúhelníkové sítě • strojové učení

Poděkování

Tímto bych rád poděkoval panu Ing. Filipu Háchovi za odborné vedení a ochotný přístup. Jeho podněty byly vždy konstruktivní a věcné. Chtěl bych ocenit jeho odbornost v tomto tématu a schopnost srozumitelného výkladu. Všechny mé dotazy vždy skvěle zodpověděl, a to i ty, které široce překračovaly rozsah této práce. Za to mu patří můj velký dík.

Obsah

Úvod 1

Pro práci s trojrozměrnými objekty je zapotřebí jejich efektivní reprezentace, ve které je bude možné ukládat, načítat a opětovně s nimi pracovat. Nejběžnější forma jejich povrchové reprezentace je pomocí polygonů, kdy jsou body na povrchu tělesa členěny do obecných n-úhelníků, jejichž plochy tvoří síť, která pak popisuje obal objektu. Vzhledem ke komplexnosti reprezentovaných objektů se jako geometrické útvary popisující 3D objekt nejčastěji používají trojúhelníky, se kterými jsme schopni dobře reprezentovat ostré hrany, jehlany nebo podobné útvary. Při jejich využití mluvíme o takzvaných trojúhelníkových sítích.

Trojúhelníkovou síť, stejně tak jako každou jinou síť tvořenou obecnými n-úhelníky, lze rozdělit do dvou hlavních složek, geometrie a konektivity. Pomocí geometrie popisujeme zmíněné vrcholy na povrchu objektu, vyjádřením jejich X, Y a Z souřadnic. Konektivita popisuje topologické uspořádání vrcholů v trojúhelníkové síti. Zjednodušeně je pak možné říct, že geometrie popisuje polohy vrcholů a konektivita stěny trojúhelníků, které vrcholy tvoří. Pokud chceme takto popsaný objekt uložit na nějaké záznamové médium, bylo by předtím vhodné provést určitou formu komprese.

Komprese obecně přestavuje proces, během nějž je redukován objem ukládaných dat snižováním jejich redundance. Jinak řečeno se snažíme využít určitých specifických vlastností dat a najít v nich přebytečné či zcela zbytečné části, které lze buďto zakódovat efektivněji, případně zcela vypustit. Naším cílem je v konečném důsledku zaznamenat stejné množství informace na menším úložném prostoru, což vede ke snížení nákladů na její uchování či při případném přenosu po síti. V našem případě bude touto klíčovou vlastností, které se při kompresi budeme snažit využít, korelace mezi uspořádáním jednotlivých bodů v prostoru a trojúhelníkovou sítí, kterou tvoří. Máme-li totiž hranu mezi dvěma vrcholy a jsou-li nám známy polohy okolních bodů této hrany, lze na základě jejich rozmístění určit, který z nich s hranou nejpravděpodobněji tvoří trojúhelník reprezentující povrch kódovaného objektu. V naší práci byl jako tento mechanismus, kterým budeme okolní vrcholy takto ohodnocovat, zvolen model umělé neuronové sítě. Ten bude danou pravděpodobnost určovat na základě geometrických vlastností vzniklých trojúhelníků.

Komprese trojúhelníkových sítí

2

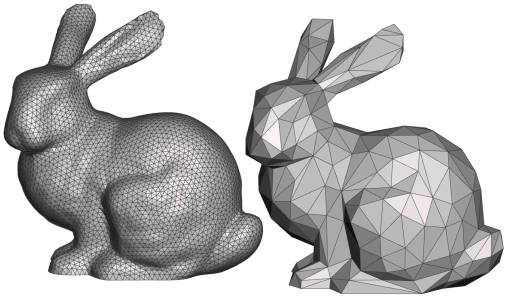
2.1 Trojúhelníkové sítě

Jak již bylo zmíněno, trojúhelníkové sítě jsou speciálním případem polygonálních sítí, které se skládají ze vzájemně propojených trojúhelníků a lze je rozdělit na geometrickou a konektivní složku. Vznik těchto sítí je úzce spojen s rozvojem počítačové grafiky a modelování, který započal v šedesátých letech minulého století. Během této doby se totiž začaly rodit první nápady pro modelování 3D objektů pomocí jednoduchých polygonů, mezi které patřily i trojúhelníky. Jejich propojování umožňovalo jednoduchou tvorbu sítě, která přitom dobře aproximovala požadovaný povrch. Postupem času a s nástupem výkonnější výpočetní techniky se proto trojúhelníkové sítě začaly těšit čím dál tím větší oblibě. Současně s tím se také rozvíjely nástroje určené na jejich úpravu a tvorbu. K pokrokům docházelo i v oblasti algoritmů využívaných pro zpracování těchto sítí, kde byly představovány stále nové, sofistikovanější a efektivnější techniky.

V současné době jsou trojúhelníkové sítě v oblasti 3D grafiky považovány za standard používaný v drtivé většině aplikací. Uplatnění nacházejí zejména v oblasti kinematografie a strojírenského průmyslu. V posledních letech, především pak díky nástupu 3D tisku a virtuální reality, však dochází k jejich rozšiřování i do dalších odvětví, jako je například stavebnictví či design. V neposlední řadě pak nelze opomenout herní průmysl, který je s 3D grafikou a tedy i s trojúhelníkovými sítěmi úzce spojen a přestavuje nezanedbatelnou část podílející se na vývoji v tomto odvětví.

2.2 Komprese

I v kontextu trojúhelníkových sítí lze nad těmito strukturami provádět kompresi a to jak ztrátovou tak bezeztrátovou. Zatímco bezeztrátová komprese umožňuje plné obnovení původních dat bez jakékoli ztráty informace, ztrátová komprese vede k redukci datového objemu na úkor částečné ztráty informací. V praxi to znamená, že při použití ztrátové komprese může být obnovený model mírně odlišný od původního,



Obrázek 2.1: Výsledný model před a po jeho simplifikaci. Ryan Schmidt, zdroj: `www.gradientspace.com/tutorials/2017/8/30/mesh-simplification`, přístup dne 28.4.2024.

což však může být přijatelné pro aplikace, kde absolutní přesnost není kritická, jako například ve zmíněném herním průmyslu nebo ve vizualizacích. Na druhé straně, v aplikacích vyžadujících vysokou míru přesnosti, jako je strojírenství nebo lékařské modelování, je často nutné využít metody bezeztrátové komprese. Obě dvě tyto varianty lze u trojúhelníkových sítí dále dělit podle oblasti jejich uplatnění, a to konkrétně na kompresi zaměřenou buďto na geometrickou, nebo na konektivní složku sítě.

V případě geometrie se komprese zaměřuje na redukci informací potřebných k popisu polohy jednotlivých vrcholů v síti. Využívá se přitom metod, které snižují množství dat potřebných k popisu geometrických vlastností, například kvantizace nebo predikce polohy vrcholů na základě známé konektivity, kdy se kóduje pouze odchylka této predikce od skutečné polohy vrcholu.

Na druhou stranu, komprese konektivity se zabývá způsobem, jakým jsou jednotlivé vrcholy propojeny. Vzhledem k tomu, že v manifoldních trojúhelníkových sítích je konektivita relativně předvídatelná tj. každý trojúhelník je spojen maximálně s třemi dalšími (každou hranou s jedním), algoritmy pro kompresi konektivity se snaží využít této skutečnosti pro zredukování množství informací potřebných k popisu těchto vazeb. To je často prováděno prostřednictvím technik, které identifikují a využívají opakující se vzory během rekonstrukce sítě nebo její geometrické struktury.

Na rozdíl od komprese geometrie, která může být například díky zmíněné kvantizaci ztrátová, je komprese konektivity téměř výhradě bezeztrátová. Je tomu tak již z její významové podstaty, která definuje, zda mezi vrcholy existuje či neexistuje spojení. Pokud by při kompresi došlo ke ztrátě této informace, mohl by tento výpadek mít fatální dopad na vlastnosti a chování dekomprimovaného modelu.

Případem, kdy může být i konektivita komprimována ztrátově, je proces takzvané simplifikace sítě [AS96]. V rámci něj mohou být odstraněny celé vrcholy nebo

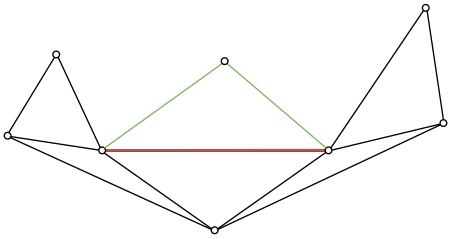
některé hrany mezi nimi, což může vést ke změně v konektivitě sítě. Cílem je zredukování celkové složitosti modelu, která je důležitým faktorem například v rychlosti jeho renderování. Ta může být totiž v určitých situacích přednější, než jeho věrné zobrazení. Příklad takovéto simplifikace je zobrazen na obrázku 2.1.

2.3 Edgebreaker

Jedná se o jednu z metod využívanou pro kompresi konektivity, která byla v roce 1999 vyvinuta profesorem J. Rossignacem [Ros99]. Metoda Edgebreaker je založena na principu toho, že existuje jen omezené množství způsobů, kterými lze při dekompresi připojovat nové trojúhelníky do vznikající sítě. Těchto způsobů je konkrétně pět a každý z nich může být reprezentován příslušným symbolem *create, left, right, end* a *split* (jejich význam bude vysvětlen níže). Důležité je, že tak jako většina dalších metod pro kompresi konektivity, ani Edgebreaker nevyžaduje znalost kódované geometrie. Díky tomu je možné zakódovat konektivitu jako první a následně využít její znalosti pro efektivnější kódování geometrie. Samotný proces dekomprese pak funguje tak, že dekodér se nejprve přesune do předem stanovené pozice v síti, kde je vytvořen první trojúhelník. Poté začne čtení jednotlivých symbolů z kódové sekvence vzniklé při kompresi a na základě jejich významů jsou do sítě připojovány nové trojúhelníky a síť se tím rozrůstá. Takto je postupováno dokud nejsou zpracovány všechny symboly a síť není kompletní. Nyní si ukážeme význam jednotlivých symbolů a situace, ve kterých nastávají.

Create

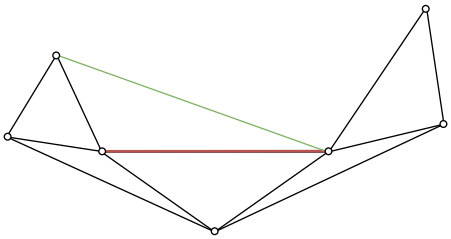
Je vybrána aktivní hrana, často také označována anglickým slovem *gate* (na obrázku vyznačena červeně), o které víme, že je součástí nově vznikajícího trojúhelníku. Dekodér dostává informaci, že trojúhelník se má připojit pomocí metody *create*. Trojúhelník je tedy tvořen vrcholy aktivní hrany a třetím vrcholem, který nám ještě není známý. Naproti aktivní hraně tak vytvoříme nový vrchol a s aktivní hranou jej propojíme (na obrázku vyznačeno zeleně). Vzniklý trojúhelník připojíme do vznikající sítě.



Obrázek 2.2: Případ *create*, kdy dochází k vytvoření trojúhelníku s nově přidaným vrcholem.

Left

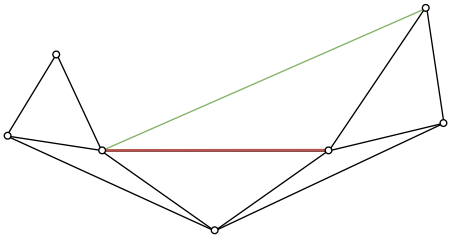
Opět je vybrána aktivní hrana, ke které se bude připojovat nový trojúhelník, tentokrát však metodou *left*. Jak již může název napovědět, protější vrchol, tvořící nový trojúhelník, se bude nacházet nalevo od aktivní hrany a to dokonce bezprostředně, hned jako první vrchol na který narazíme, budeme-li se pohybovat směrem doleva podél hranice zpracované části sítě. Důležité je poznamenat, že tento vrchol musel být již dříve zpracován a přidán metodou *create*.



Obrázek 2.3: Případ *left*, kdy dochází k propojení s vrcholem bezprostředně nalevo od aktivní hrany.

Right

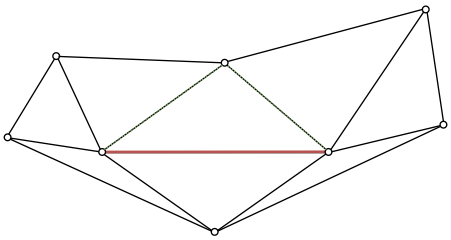
Zde se jedná o obdobnou situaci jako v metodě *left*. Postup je identický pouze s tím rozdílem, že výběr vrcholu je prováděn pohybováním se směrem doprava podél hranice zpracované části sítě. I v tomto případě musel být vybraný vrchol nejdříve zpracován metodou *create* a proto je na něj nyní možno odkazovat tímto způsobem.



Obrázek 2.4: Případ *right*, kdy dochází k propojení s vrcholem bezprostředně napravo od aktivní hrany.

End

Tato situace nastane v případě, kdy se při vytváření sítě dostaneme do bodu, kdy jsou nám známy všechny okolní body aktivní hrany a v síti zbývá "díra" ,kterou je třeba vyplnit. Pro tento případ se metody *left* i *right* vztahují ke stejnému vrcholu. V důsledku toho pak nedochází k vytvoření nové hrany, jelikož všechny stěny trojúhelníku již existují. Dojde pouze k zaznamenání toho, že tyto hrany tvoří další unikátní trojúhelník. Tímto je daná část sítě uzavřena a buďto je vybrána nová aktivní hrana se kterou algoritmus pokračuje, nebo je síť v danou chvíli kompletní a algoritmus zde končí.

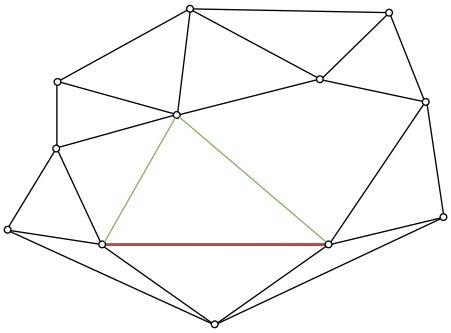


Obrázek 2.5: Případ *end*, kdy dochází k vyplnění "díry"v sítí definováním nového trojúhelníku.

Split

Jedná se o nejkomplexnější případ ze všech dosud uvedených. Nastává ve chvíli, kdy je potřeba připojit nový trojúhelník jehož protější vrchol je nám opět již známý. Tentokrát se ovšem nejedná o vrchol bezprostředně nalevo ani bezprostředně napravo od aktivní hrany, nýbrž o vrchol ležící v obecné vzdálenosti někde podél hranice zpracované sítě. Na určení toho, o jaký vrchol se jedná, však není třeba jeho pozici explicitně uvádět. Namísto toho lze tuto pozici vypočíst pomocí dopředného průchodu kódovou sekvencí až po nejbližší symbol *end*. Ve chvíli, kdy dojde ke zpracování tohoto symbolu je totiž jasné, že délka hranice musí být 3 (bude se skládat

ze tří hran). Dále je zřejmé, že zpracovávání předchozích symbolů tuto délku ovlivnilo. Je však možné přesně určit jak. Byl-li předchozím symbolem *left* nebo *right*, délka hranice se vždy zmenšila o 1, naopak pokud se jednalo o symbol *create*, délka hranice se o 1 zvýšila. Tímto sledováním změn délky hranice lze pak zpětně určit jaká má být její délka při zpracovaní symbolu *split* a tedy i přesně určit, se kterým vrcholem hranu propojit tak, aby byla tato podmínka splněna.



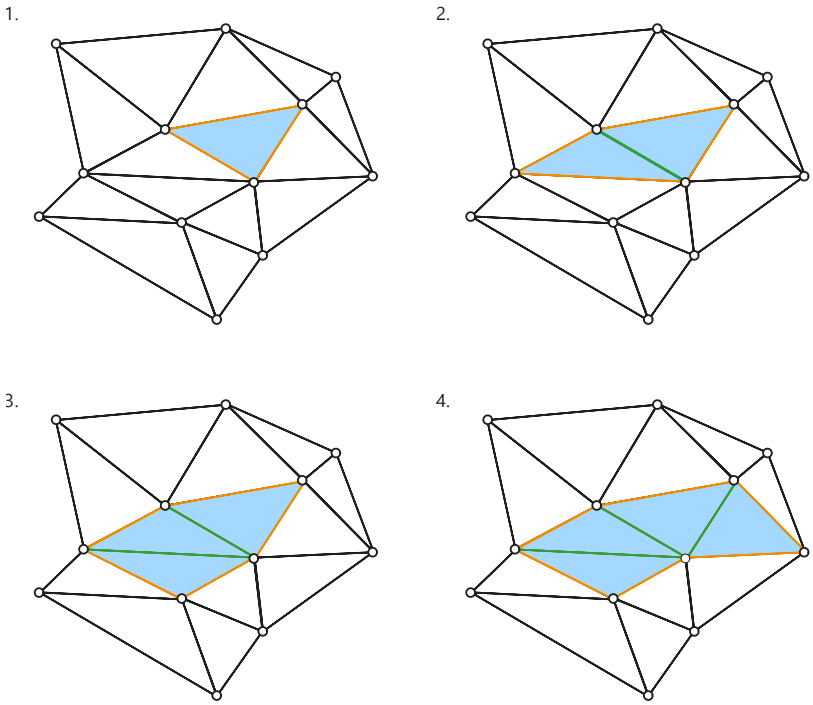
Obrázek 2.6: Případ *split*, kdy se protější vrchol nachází podél hranice kódované sítě.

2.4 Referenční metoda založená na principu kandidátních vrcholů

Na rozdíl od většiny konvenčních metod, existují i přístupy pro kódování konektivity s předem známou geometrií. Příkladem takového přístupu je metoda Priority-based encoding [Dvo+23] navržená J. Dvořákem a spol. navazující na práci P. Maraise [MGS07]. Obě tyto metody využívají informaci o známé geometrii kódované sítě k odhadu zbývající konektivity a díky této znalosti jsou schopny kódovat konektivitu značně efektivněji, což představuje jejich klíčovou výhodu. Metoda J. Dvořáka však přináší značné zlepšení v procesu kódování související s výběrem aktivních hran a ohodnocováním okolních vrcholů. Metoda se skládá ze dvou hlavních částí. První část zahrnuje mechanismus pro výběr kandidátních vrcholů a jejich ohodnocení na základě kvality, s jakou by mohly s aktivní hranou tvořit připojovaný trojúhelník. Druhá část se zabývá výběrem konkrétní hrany, k níž bude nový trojúhelník připojen. K tomu algoritmus využívá predikce pro určení, které hrany kódovat jako první, přičemž dává přednost těm, kde je predikce na základě ohodnocení okolních vrcholů nejspolehlivější. Tím se zvyšuje efektivita a přesnost celkového kódování.

2.4.1 Popis algoritmu

Na počátku je vybrán libovolný trojúhelník, u kterého algoritmus započne (obsahuje-li síť více komponent, je pro každou komponentu vybrán jeden trojúhelník) a jeho ko-



Obrázek 2.7: Znázornění postupného rozšiřovaní zpracované oblasti připojováním nových trojúhelníků.

nektivita zakódována, respektive dekódována přímo pomocí indexu vrcholů, které ho tvoří. Síť je tak rozdělena na zpracovanou a nezpracovanou část. Ve zpracované části leží zatím jen tento vybraný trojúhelník a v nezpracované části zbytek sítě. Oblast nacházející se na pomezí těchto dvou částí se nazývá *hranice*, a je tvořena jednotlivými hranami.

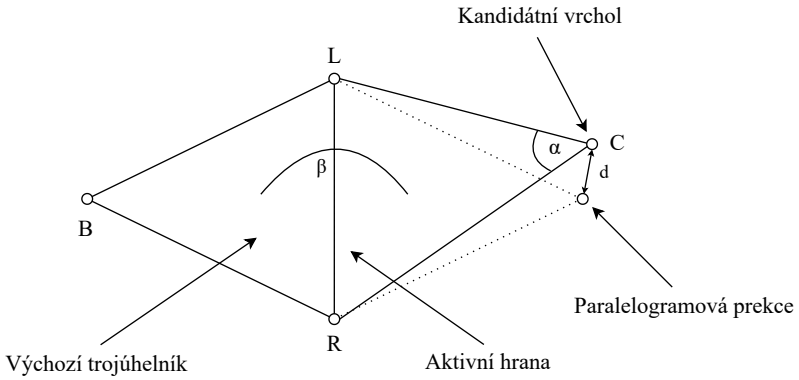
Následuje cyklus, do kterého vstupuje jak kodér tak dekodér. V něm jsou postupně z hranice vybírány *aktivní hrany* a k nim připojovány nové trojúhelníky, tj. pro každou hranu je vybrán *protější vrchol*, kterým je daný trojúhelník tvořen. Toho je docíleno tak, že k hraně je vybrána množina kandidátů na tento vrchol a každému z nich je přidělena kvalita, podle které jsou následně seřazeni. Kodéru je protější vrchol samozřejmě známý a jeho hlavní úlohou je zde informovat dekodér o jeho pozici v tomto seznamu a to jejím zápisem do výstupní sekvence. Tímto připojováním nových trojúhelníků dochází ke stálému rozšiřovaní zpracované části (viz obr.2.7). Takto je postupováno dokud nejsou zpracovány všechny hrany a komprese konektivity je tak hotová. V poslední řadě je provedena komprese ještě nad samotnou výstupní sekvencí za účelem ještě většího kompresního poměru.

2.4.2 Uzavírání vrcholů

Uzavírání vrcholů je jedním z optimalizačních prvků, využívaných tímto algoritmem. Jak bylo v popisu této metody zmíněno, síť je rozdělena na zpracovanou a nezpracovanou část. Průchodem sítě a připojováním nových trojúhelníků se pak zpracovaná část stále rozrůstá až do té doby, dokud není síť zpracovaná celá. Uvažujeme-li zpracovávání manifoldních sítí, vrcholy, které lze do zpracované sítě připojit, musí ležet buďto na její hranici, nebo mimo ni v nezpracované části. K vrcholům ležícím uvnitř se nevracíme. Není tedy možné vytvořit trojúhelník, jehož protější vrchol by ležel uvnitř zpracované části. Pro každý vrchol je tedy zavedena kontrola, zda již není *uzavřený*. V daném případě ho totiž lze trvale vyloučit jakožto potencionálního kandidáta na protější vrchol jakékoliv v budoucnu zpracovávané hrany. Kontrola může probíhat následovně. Pro každý vrchol je evidována množina hran, jichž je daný vrchol součástí. Není-li vrchol zatím součástí žádné hrany, znamená to, že jsme se k němu průchodem sítě ještě nedostali. Tudíž leží mimo zpracovanou část sítě a je v tom případě stále možným kandidátem na protější vrchol, ba je téměř žádoucí, aby byl tento vrchol do sítě ještě připojen. Jako druhá varianta je případ, kdy vrchol v naší síti již leží. Tím pádem musí existovat i množina hran, kterých je součástí (jiná možnost také ani není). Jak jinak by se totiž vrchol do sítě dostal než tak, že došlo k vytvoření hrany mezi ním a vrcholem ve zpracované síti. Jelikož víme, že jediným případem, kdy může být vrchol stále otevřený a zároveň součástí zpracované sítě je ve chvíli, kdy leží na její hranici. Zkontrolujeme tedy množinu jeho hran, kde se musí nacházet alespoň jedna taková, která je součástí hranice. Pokud tomu tak není, lze vrchol prohlásit za uzavřený a přesunout jej do seznamu uzavřených vrcholů. Podstatné je, že toto označení je definitivní a dalším průchodem sítě se stav vrcholů označených jako uzavřených již nemůže změnit. V dalších částech se tedy o tuto informaci budeme opírat a v určitých situacích bude mít na funkci algoritmu zásadní vliv.

2.4.3 Ohodnocení vrcholu

Ohodnocení vrcholu je založeno na geometrických vlastnostech trojúhelníku vzniklého propojením aktivní hrany s vybraným vrcholem a jeho vztahu k *výchozímu trojúhelníku* (jedná se o trojúhelník ležící ve zpracované části jehož je aktivní hrana součástí). Metoda pro výpočet kvality pak zohledňuje několik z těchto vlastností. Jako první je to odchylka d, představující vzdálenost kandidáta od paralelogramové predikce [TG98] podělené průměrnou délkou hrany l v kódované síti. Jako druhá vlastnost je hodnota α, rovnající se velikost vnitřního úhlu trojúhelníku v kandidátním vrcholu, přičemž kvalita kandidáta je přímo úměrná této hodnotě. Třetí vlastnost také souvisí s úhly a je jí dihedrální úhel β mezi plochou výchozího a nově vzniklého trojúhelníka. Jako poslední vlastnost je symetrie S obou trojúhelníků, udá-



Obrázek 2.8: Znázornění zohledňovaných geometrických vlastní po výpočet kvality.

vající, jak moc jsou si trojúhelníky podobné. V případě naprosté shody je $S = 0$ a s rostoucí odlišností začíná nabývat záporných hodnot. Její výpočet je popsán ve vzorci 2.1. V něm je nejprve vypočten poměr r_k a r_d mezi kratšími hranami výchozího a protějšího trojúhelníku v_k, p_k a jejich delšími hranami v_k, p_k. Následně je určen jejich průměr r. Výsledná symetrie je určena jako záporná hodnota průměrné odchylky poměrů od průměru r. Výsledný vzorec pro výpočet kvality kandidáta K_i je pak formulován vzorcem 2.2.

$$\begin{aligned} r_k = \frac{v_k}{p_k}, r_d = \frac{v_d}{p_d} \\ r = (r_k + r_d)/2 \\ S = -\left(|r - r_k| + |r - r_d|\right)/2 \end{aligned} \tag{2.1}$$

$$K_i = \alpha - \frac{d}{l} \cdot w_1 + \beta \cdot w_2 + S \cdot w_3 \tag{2.2}$$

Všimněme si, že vzorec také obsahuje váhy w_1, w_2 a w_3 pro jednotlivé parametry. Ty určují jak moc dílčí vlastnosti vzniklého trojúhelníku ovlivňují výslednou kvalitu kandidáta. Nastavení těchto vah může tedy výrazně ovlivnit efektivitu samotného kódování. Zároveň jejich optimální nastavení se různí dle jednotlivých sítí, respektive geometrických vlastností trojúhelníků, které ji tvoří.

2.4.4 Výběr kandidátů a určení protějšího vrcholu

Mějme aktivní hranu, ke které je momentálně požadováno připojit nový trojúhelník. I přes to, že jsou dekodéru známy polohy všech okolních bodů, nemá představu o tom, se kterým z nich by tento trojúhelník měl vytvořit. V tuto chvíli musí danou informaci předat kodér. Je sice možné předat ji přímo zápisem indexu daného bodu

do výstupní sekvence, ze které by ho dekodér zpětně přečetl, to bylo by to ale velice neefektivní. S rostoucí velikostí kódované sítě totiž roste i počet bitů potřebných na indexaci všech jejích vrcholů. Tím pádem by se postupně zvětšovala i celková velikost výstupní sekvence. Kromě toho by zapisované indexy byly víceméně zcela náhodné, respektive pro n indexů by byla pravděpodobnost výskytu jednoho specifického indexu $1/n$. To by kompresi dále negativně ovlivňovalo, pokud budeme uvažovat, že nad výstupní sekvencí chceme posléze provádět další formu komprese.

Existuje však řešení. Je důležité si uvědomit, že geometrie a konektivita jsou do určité míry provázány. Je tak možné usuzovat, že protější vrchol aktivní hrany se bude pravděpodobněji nacházet mezi těmi vrcholy ležícími v jejím okolí, než v odlehlé oblasti daleko od této hrany. Pokud budeme postupovat ještě dále a omezíme-li se na poměrně pravidelně teselované sítě s pravidelným dělením, je možné určit kvalitu výběru konkrétního bodu, jakožto protějšího vrcholu, pomocí jeho pozice vzhledem k dané hraně. Přesně tak je postupováno i v tomto případě. Nejprve je tedy zapotřební vytvořit seznam kandidátních vrcholů. Do něj jsou zařazeny všechny vrcholy ležící do určité vzdálenosti d od paralelogramové predikce. Vzdálenost d je konstruována jako oblast, ve které se ještě mohou nacházet vrcholy jejichž kvalita je lepší nebo stejná, jako kvalita prozatím nejlepšího nalezeného kandidáta.

Máme-li vrchol s kvalitou $q = q_p$ je pak vzhledem k jejímu výpočtu možné určit maximální vzdálenost d_{max}, ve které se ještě mohou vrcholy s lepší kvalitou nacházet:

$$d_{max} = (w_2 \cdot \pi + \pi - q_p) \cdot \frac{l}{w_1} \tag{2.3}$$

Tento vzorec tedy říká, že vrcholy, pro které platí, že $d > d_{max}$ už nemohu mít lepší kvalitu než q_p. Dále lze také určit, že pokud má mít kandidát kvalitu $q > q_p$, musí pro jeho vnitřní úhel platit $\alpha > \alpha_{min}$, kde α_{min} je definována jako:

$$\alpha_{min} = q_p - w_1 \cdot \pi \tag{2.4}$$

Tyto restrikce tak definují výslednou oblast z níž jsou kandidáti vybráni. Do třetice přichází využití informace o uzavřených vrcholech. Vyskytuje-li se mezi kandidáty jeden či více takových, pro které platí, že se nachází v tomto seznamu, je možné je z aktuálního seznamu kandidátů vyloučit. Vyhneme se tak zcela zbytečnému výpočtu kvality kandidátů, kteří už nemohou být vybráni jako protější vrchol.

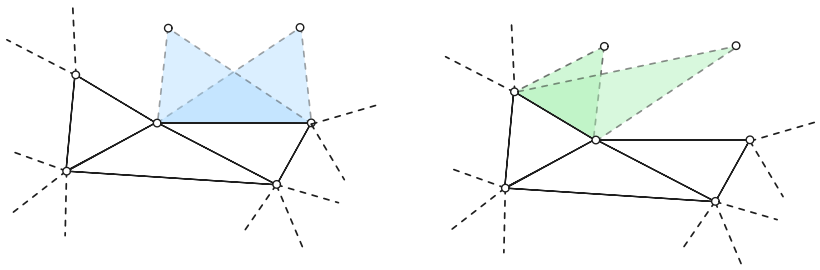
Po tomto výběru je zbylým vrcholům přidělena jejich kvalita viz sekce 2.4.3, podle níž jsou následně seřazeni v sestupném pořadí . Celý tento postup provede nezávisle jeden na druhém jak kodér tak dekodér. Výsledkem je seznam, který bude pro oba dva stejný. Stále mějme na paměti, že dosud nebylo nutné přenášet mezi

kodérem a dekodérem žádné informace. V tuto chvíli má již samotný dekodér poměrně jasnou přestavuj o tom, který z kandidátů je protějším vrcholem. Bude jím s velkou pravděpodobností první či jeden z prvních kandidátů v seznamu. Kodér tak pouze potvrzuje správnost úsudku dekodéru, případně posílá korekci ve formě relativního indexu protějšího vrcholu v seznamu kandidátů. To je možné právě z toho důvodu, že kodér celou dobu emuluje postup dekodéru k získání tohoto seznamu. Pro celý proces je ovšem zcela zásadní správnost vzorce, kterým kvalitu počítáme a podle které se také řídíme. Ten, jak bylo již předem uvedeno, vychází z předpokladu pravidelných trojúhelníků, ze kterých je síť tvořena. V případech, kdy tomu tak není, budou v seřazeném seznamu protější vrcholy často ležet na vyšších indexech. Správný odhad dekodéru se tím výrazně zhorší a kodér bude muset provádět větší korekci. To povede k celkovému zhoršení efektivity kódování.

2.4.5 Výběr aktivní hrany

V tuto chvíli již víme, jak kandidátní vrcholy vybrat a vzhledem k aktivní hraně i určit jejich kvalitu. Doposud jsme si ale neukázali, na jakém principu z množiny otevřených hran zvolíme tu, ke které nový trojúhelník připojit. Bylo tomu tak proto, že její výběr s ohodnocením kandidátů úzce souvisí. Samotný výběr hrany je pak ale velmi prostý. K této akci je využita prioritní fronta. Ve chvíli kdy algoritmus narazí na novu hranu, najde pro ni nejvhodnější kandidáty a provede jejich ohodnocení stejně tak, jako kdyby se k ní momentálně snažil připojit nový trojúhelník. Kvality kandidátů se opět seřadí v sestupném pořadí a výsledná priorita, se kterou je hrana vložena do prioritní fronty udává rozdíl kvalit prvního a druhého nejlepšího kandidáta.

I zde se ukáže, jak přínosná může být správně zvolená strategie průchodu sítě, zejména když je kombinovaná s informacemi o uzavřených vrcholech. Představme si situaci s hranou se dvěma kandidáty, kdy je obtížné určit, který z nich je protějším vrcholem, jelikož rozdíl v jejich hodnocení je malý. Nicméně, může existovat další hrana, kde je hodnocení týchž kandidátů značně rozdílné (viz obr.2.9), což umožňuje upřednostnit jednoho z nich. Je proto logické pokračovat s touto hranou. Tímto způsobem průchodu je možné nepřímo usnadnit rozhodovací proces pro ostatní hrany. Když se na vrcholu fronty objeví ta hrana, kterou jsme si nebyli jisti, předchozím průchodem sítě přes jistější hranu může být jeden z kandidátů již uzavřen, což odstraní původní nejistotu a umožní s jistotou vybrat druhý vrchol. Cílem je poukázat na to, že pro dvě hrany sdílející stejné kandidáty může být jejich hodnocení pro každou z nich značně odlišné. Využitím jistější cesty je tedy možné dosáhnout uzavření jednoho z vrcholů ještě před příchodem k nejisté hraně.



Obrázek 2.9: Ukázka toho jak se může poměr ohodnocení dvou kandidátů výrazně lišit v závislosti na výběru konkrétní hrany.

2.4.6 Sítě s hranicí

Na závěr si ukážeme, jak tato metoda postupuje při práci se sítěmi obsahujícími *hranici* (nejedná se zde o hranici v kontextu zpracované a nezpracované části sítě). Ty z hlediska konektivity tvoří svou vlastní kategorii sítí, v níž některé jejich trojúhelníky nejsou plně propojeny se zbytkem sítě. Takové trojúhelníky pak typicky obsahují jednu či dvě hrany, ke kterým není připojen sousední trojúhelník. Tyto hrany jsou pak běžně označovány jako hraniční. Pro algoritmy, jako je i tento, pracující na principu postupného průchodu sítí pak představují značný problém, jelikož přes ně, z důvodu neexistence přilehlého trojúhelníka, není možné provádět expanzi zpracovávané sítě. To je zde vyřešeno pomocí dvou mechanizmů: predikce toho, zda je hrana potencionální hranicí a vyhrazením speciálního kódu *borderCode* pro označení takovýchto hran.

Predikce hranice

Pro predikci, zda se v případě hrany jedná o hranici, je využito kvality jejího nejlepšího kandidáta. Pokud bude nižší než určitá prahová hodnota t je hrana považována za hranici a odpovídajícím způsobem pak zakódována. Pro výpočet této prahové hodnoty je před hlavním průchodem sítě, během kterého je prováděno kódování, proveden ještě jeden. V něm jsou zaznamenány kvality nejlepších kandidátů všech hran a rozděleny do dvou seznamů K_h a K_v podle toho, zda přísluší hraniční nebo vnitřní hraně. Jelikož pro hraniční hrany v síti neexistuje protější vrchol, je předpokládáno, že kvality jejich kandidátů v seznamu K_h by neměly být dobré a oproti kvalitám kandidátů vnitřních hran v seznamu K_v značně nižší. Výpočet prahové hodnoty je tak proveden podle vzorce 2.5, v němž je práh t určen jako jedna z možných kvalit v seznamech K_h a K_v, která nejlépe rozdělí tyto seznamy. V případě, že síť neobsahuje žádné hraniční hrany tj. seznam K_h je prázdný, je prahová hodnota nastavena jako $-\infty$. Tím je zajištěno, že kvalita kandidáta žádné hrany nebude nižší než tento práh a hrana nebude mylně identifikována jako hranice. Výsledný práh je

zaslán dekodéru, aby mezi jím a kodérem byla zachována synchronizace ve smyslu detekce takovýchto hran.

$$\max_{t \in K_h \cup K_v} |\{i \in K_h : i \leq t\}| + |\{j \in K_v : j > t\}| \tag{2.5}$$

Kódování hranice

Během kódování s takto zavedenou detekcí hranic pak může nastat několik případů, kterým se dosavadní způsob kódování musí přizpůsobit. První dva případy mohou nastat ve chvíli, kdy je zpracovávána hraniční hrana. Pokud je dekodérem detekována (Očekáváme, že predikce je spolehlivá a tento případ bude nejčastější.), dekodér tuto predikci pouze potvrzuje. Vzhledem k dosavadní strategii indexace silně vychylované k nule, je pro tento případ shodně použit symbol 0. Naopak pokud hrana není dekodérem detekována, musí kodér zaslat speciální symbol indikující, že tato situace nastala. Ten je, stejně jako prahová hodnota, dekodéru poskytnut předem a je roven nejvyššímu indexu ve výstupní sekvenci zvýšeného o 1 (Jelikož hodnota nejvyššího indexu není kodéru známa předem, během kódování je tento symbol ve výstupní sekvenci dočasně nahrazen hodnotou -1 a až poté zpětně dopočten.).

Poslední případ, který je třeba ošetřit je situace, kdy dojde k nesprávnému identifikování hrany jako hraniční. V tu chvíli nastává konflikt mezi kodérem, který zasílá index protějšího vrcholu hrany a dekodérem, který očekává jako následující symbol potvrzení jeho predikce hranice. Jelikož zde může dojít k záměně interpretace symbolu 0, je kodérem zasílaný index vždy inkrementován o 1. Jelikož výsledná hodnota se tak nikdy nemůže rovnat nule, dekodér dostává informaci o zamítnutí jeho predikce a zároveň dekrementací předané hodnoty dostává i informaci o indexu protějšího vrcholu. V případě, kdy hrana není hranicí a není tak ani identifikována, zůstává postup kódování samozřejmě stejný.

2.4.7 Komprese výstupní sekvence

Poté, co je kódování dokončeno a ve výstupní sekvenci je uložena konektivita sítě, je před uložením do souboru provedena komprese této sekvence. Pro tento účel bylo vybráno entropické kódování, konkrétně kódování pomocí adaptivního aritmetického kodéru. Entropické kódování komprimuje data tím způsobem, že nahrazuje vstupní symboly (v našem případě relativní indexy vrcholů) sekvencí s proměnou délkou. Pro symboly s velkou četností jsou vyhrazeny sekvence s kratší délkou a naopak málo časté symboly jsou zakódovány delšími výstupními sekvencemi. Tímto způsobem dochází k optimálnímu zakódování. Využití adaptivního aritmetického kodéru je obzvláště vhodné, protože se dynamicky přizpůsobuje měnícím se statistikám symbolů během kódování. To znamená, že kodér nevyžaduje předem známou

frekvenční distribuci symbolů a místo toho si sám postupně buduje odhad pravděpodobnosti jejich výskytu, což vede k ještě efektivnějšímu zakódování. V kontextu představené komprese konektivity, kde rozdělení pravděpodobnosti výskytu jednotlivých indexů není rovnoměrné, ale naopak je zde očekáváno, že kódované vrcholy se budou pohybovat mezi prvními (optimálně vždy na indexu 0) je tento způsob komprese ideální.

Umělé neuronové sítě 3

Jako umělé neuronové sítě označujeme v informatice implementaci modelu strojového učení spadající do oblasti umělé inteligence. Ty jsou využívané pro řešení řady náročných úloh, pro které by bylo často obtížné nacházet ekvivalentní deterministické řešení. Myšlenkou tohoto návrhu je vytvořit síť podobnou té, která vznikala během miliónů let vývoje v lidském mozku propojováním jednotlivých neuronů do spletité struktury synapsí [Süd21]. Cílem je, aby síť po jejím naučení, na základě procházení velkého množství dat byla následně schopna predikovat nejpravěpodobnější výstup pro data, která předtím ještě neviděla.

3.1 Historie

V roce 1943 v publikaci [MP43] navrhl americký neurofyziolog a kybernetik Warren Sturgis McCulloch společně se svým spolupracovníkem Walterem Pittsem první matematickou formulaci nervové aktivity mezi mozkovými neurony, čímž se klíčovým způsobem zasadili o rozvoj mnoha vědeckých odvětví jako je například umělá inteligence či jako jsou kognitivní a generativní vědy. V jejich práci vycházeli z předpokladu, že reakce neuronu na podnět je "vše-nebo-nic"charakteru, při kterém nezáleží na síle tohoto podnětu. Je-li překročen určitý aktivační práh, neuron na podnět reaguje a to vždy s maximální intenzitou, naopak v opačném případě na něj nereaguje vůbec. Signál zde zaniká a dál se nešíří. Takto byli schopni vytvořit model s využitím logického kalkulu a popsat tyto děje pomocí výrokové logiky. Ačkoliv bylo jejich bádání jednoznačným přínosem pro nadcházející rozvoj umělých neuronových sítí, byly tyto poznatky pouze teoretického charakteru a nepřinášely žádné konkrétní nástroje pro učení těchto sítí.

I proto bylo dlouhou dobu používání neuronové sítě vzhledem k úrovni výpočetních prostředků a technologickému pokroku tehdejší doby velice obtížné. Práce s ní vyžadovala značné výpočetní kapacity, které nebyly snadno dostupné, nebo byly extrémně nákladné. Až v polovině 80. let 20. století se začaly objevovat první úspěšné aplikace neuronových sítí, které demonstrovaly jejich potenciál v praktických úlohách, jako je například rozpoznávání rukou psaného textu nebo zpracování

přirozeného jazyka. Tento pokrok pomohl vyvolat novou vlnu zájmu o neuronové sítě v akademické i průmyslové sféře.

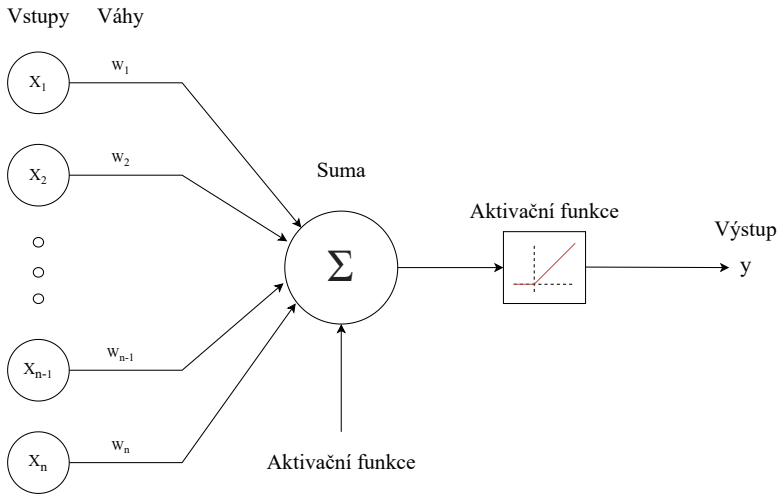
Rozhodující moment přišel s objevem a širokou adopcí algoritmu zpětné propagace chyby, který umožnil efektivní trénink vícevrstvých neuronových sítí tím, že optimalizoval jejich váhy s ohledem na minimalizaci chyby výstupu. Tato metoda se stala základním kamenem pro trénink hlubokých neuronových sítí a byla klíčová pro vývoj složitějších architektur schopných řešit obtížnější problémy. V současné chvíli zažívají neuronové sítě dobu velkého rozmachu a jejich masivní aplikace ve firmách pro řešení nejrůznějších praktických úloh. Stojí za tím hlavně rozšíření veřejného povědomí o umělé inteligenci a demonstrace jejích schopností například prostřednictvím ChatGPT [Ope20] modelu neuronové sítě určeného pro konverzaci s běžným člověkem prostřednictvím webového prohlížeče. Na práci umělé inteligence je ceněna především přesnost, se kterou ji provádí, eliminace chyby zapříčiněné lidským faktorem nebo také neúnavnost, se kterou může stroj své úkoly vykonávat.

3.2 Neuron

Neuron, neboli nervová buňka, je základní stavební složkou nervové soustavy všech živočichů. Pomocí chemických reakcí je schopen sířit eklektické signály, které mohou být pak následně interpretovány například jako chuť, zrak nebo čich. Samotný lidský mozek obsahuje desítky miliard těchto neuronů, které mezi sebou neustále komunikují a jsou tak schopni plnit i ty nejnáročnější úkoly. Ve chvíli, kdy pak McCulloch s Pittsem navrhovali jeho první matematický model, snažili se tomuto principu co nejvíce přiblížit. Jejich návrh je zachycen na obrázku 3.1 a to pouze s jednou drobnou změnou. Jelikož ve své práci model definovali pomocí logického kalkulu, který pracuje pouze se dvěma stavy, musel tomu odpovídal i tvar aktivační funkce. Ta tak nabývala pouze dvou hodnot 0 nebo 1 a to na základě toho, zda byl překročen aktivační práh neuronu. Ačkoliv se toto chování více přibližovalo skutečném fungování nervové aktivity, bylo zjištěno, že je mnohem výhodnější, aby se v případě aktivace jednalo o reálnou funkci nabývající více funkčních hodnot. V současné době tak existuje řada druhů aktivačních funkcí [DSC22], které mohou být umělými neurony využívány.

3.3 Perceptron

Po popsání dějů a funkcí v biologickém neuronu nastala otázka, jak tyto poznatky efektivně aplikovat také ve světě výpočetní techniky. Touto otázkou se zabýval i Frank Rosenblatt, který roku 1957 ve své práci [Ros57] představil svůj návrh pod názvem *perceptron*. Jednalo se o jednoduchou strukturu dopředné neuronové sítě,



Obrázek 3.1: Model umělého neuronu.

která v jeho publikaci sloužila k rozlišování jednoduchých tvarů, které jí byly představovány. Model v práci strukturuje do tří separátních částí: na část senzorickou (sensory system), část asociační (association system) a část odpovědní (responce system). Funkce každé z nich je představena níže.

Senzorická část

Senzorická část sloužila jako vstupní vrstva modelu. Jsou zde přijímány vstupní proměnné, se kterými má perceptron pracovat a podle nichž má predikovat nejpravděpodobnější výsledek. Každá z jednotek nese tedy určitou hodnotu, tou může být například frekvence, amplituda či jiná měřitelná veličina, která se dále šíří pomocí spojení mezi touto jednotkou a určitým počtem dalších jednotek v asociační vrstvě.

Asociační vrstva

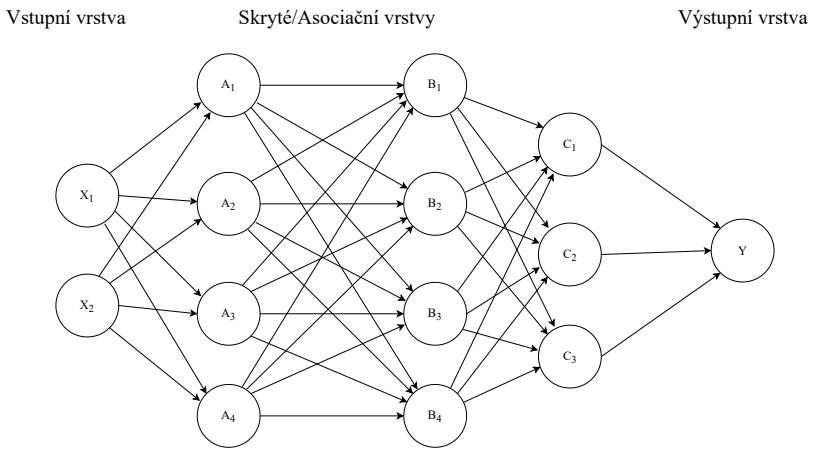
Asociační vrstva pak obsahovala předem určený počet asociačních jednotek do kterých byl přiváděn signál z jednotek senzorických. U každého spoje je zde předem jasně určeno, zda má být pozitivní či negativní, tj. jestli má hodnota jednotku stimulovat nebo její aktivitu naopak tlumit. Následně dochází k součtu velikostí všech signálů a určení zda byla překročena aktivační hranice, při které dojde k dopřednému šíření tohoto signálu do další vrstvy.

Odpovědní vrstva

Odpovědní vrstva obsahovala určitý počet odpovědních jednotek, například sadu led diod, které rozsvícením signalizovaly jednotlivé odpovědi. K zajištění jednoznačnosti výsledku byly vzájemně výlučné části propojeny *absolutním inhibičním spojením*, což bylo typem spojení mezi jednotkami, které, pokud bylo aktivováno, mělo za následek úplné potlačení jejich aktivity. Toto spojení bylo zavedeno mezi jednotkami odpovědní vrstvy, u kterých není žádoucí aby došlo k jejich současné aktivaci a také mezi jednotkami předchozí vrstvy, které její aktivitu stimulují. Jako příklad může sloužit návrh, jehož odpovědi se mají skládat z dvoupísmenných slov. Odpovědní část by byla tvořena dvěma sadami, každá o 26 jednotkách, reprezentující jednotlivá písmena. Tak aby v jedné sadě nemohlo dojít k zobrazení např. písmena N a písmena T zároveň, je každá jednotka inhibičně propojena s každou další jednotkou dané sady. Aktivace písmena N tedy automaticky blokuje zobrazení jakéhokoliv dalšího znaku v této sadě.

Schéma zapojení perceptronu

Tento návrh byl velice pokrokový a moc se nelišil od principu fungování moderních neuronových sítí. Ty vycházejí ze stejného základu předávání informací z jedné vrstvy neuronů do další. Hlavním rozdílem je, že v současné době jsou moderní sítě výrazně hlubší a obsahují mnohem více na sebe navázaných asociačních (skrytých) vrstev jak je ukázáno na obrázku 3.2.



Obrázek 3.2: Struktura vícevrstvého perceptronu.

3.4 Optimalizace

Optimalizace je při tréninku neuronových sítí jednou z nejdůležitějších částí celého procesu. Stará se o úpravu vah tak, aby se minimalizovala chyba mezi predikcemi modelu a požadovanými výsledky.

Inicializace vah

Každé spojení mezi dvěma neurony je ohodnoceno vahou, která udává jak velký vliv má aktivace jednoho neuronu na aktivaci neuronu druhého. Ta může být samozřejmě jak pozitivní, tak negativní. Správné nastavení těchto vah pak vede k optimálnímu výkonu neuronové sítě a je hlavním předmětem celého procesu učení. Na začátku jsou tedy váhy inicializovány jako náhodné hodnoty. I toto nastavení, ač náhodné, může mít ovšem značný vliv na samotné učení a to především z hlediska rychlosti konvergence a schopnosti najít globální minimum chybové funkce. Existují proto různé metody pro inicializaci těchto vah, které často využívají různé formy normálního rozdělení jako například metoda [He+15], u které je střední hodnota rovna 0 a rozptyl definován jako $2/N_p$ kde N_p je počet neuronů v dané vrstvě.

Chybová funkce

Chybová funkce slouží k výpočtu metriky, pomocí které lze určit, jak moc se predikce modelu liší od požadovaných výsledků. Její výběr záleží na druhu použitého modelu a nároků, které jsou na něj kladeny. Ty lze rozdělit do několika kategorií. Typickými zástupci jsou například regresní či klasifikační úlohy. V případe regrese je model používán pro aproximaci určitých funkčních dat. Chybová funkce zde tedy zohledňuje, jak moc se jednotlivé predikce liší od aproximovaných hodnot. V klasifikačních úlohách má zas model za úkol rozřazovat data do jednotlivých tříd klasifikace. Zde naopak chybová funkce určuje, jak moc se modelu toto rozřazování daří. Podle požadavků těchto úloh je tedy vybrána vhodná funkce pro výpočet dané chyby. Jako příklad může být uvedena cenová funkce *křížové entropie (Cross-entropy loss function)* [MMZ23], definovaná pomocí vzorce 3.1. $H(P', P)$ je zde chyba modelu, P'_i je skutečná pravděpodobnost, že prvek patří do třídy klasifikace i, P_i je tatáž pravděpodobnost pouze predikovaná pomocí modelu a n je celkový počet tříd klasifikace. Existuje však i modifikovaná verze tohoto vzorce *Binary cross-entropy function*, která uvažuje pouze dvě třídy klasifikace. Ten pak dostává tvar 3.2, kde y značí, zda daný prvek skutečně náleží do dané třídy a p je opět tatáž pravděpodobnost pouze určená pomocí modelu.

$$H(P', P) = -\sum_{i=1}^{n} P'_i \cdot \log(P_i) \tag{3.1}$$

$$H(y, p) = -[y \log(p) + (1 - y) \log(1 - p)] \tag{3.2}$$

Optimalizační algoritmus

Po zvolení chybové funkce je zapotřebí k ní vybrat vhodný optimalizační algoritmus. Jedná se zde o nástroj určený k její minimalizaci výpočtem gradientu, s jehož pomocí jsou jednotlivé váhy modelu následně upravovány tak, aby se budoucí predikce projevovaly s co nejmenší chybou. To probíhá v takzvaných *epochách*, kdy jedna epocha představuje průchod napříč celou trénovací sadou. Tato iterace je navíc běžně rozdělena do jednotlivých dávek (*batches*), tvořících podmnožinu celé sady. Při zpracování dávky se z trénovacích dat nejprve náhodně vybere konstantní počet vzorků. Ty jsou následně modelem dopředným průchodem zpracovány a určeny jejich predikce. Pomocí zpětného průchodu je pak, na základě rozdílu mezi predikcemi a skutečnými cílovými hodnotami vzorků, vypočten gradient ztrátové funkce. Podle vybraného optimalizačního algoritmu jsou potom na základě tohoto gradientu váhy modelu upraveny, tak aby se predikce více blížily požadovaným datům. Vybraný algoritmus má tímto na učení modelu zásadní vliv. Určuje strategii toho, jak má na zjištěnou chybu úpravou vah reagovat a jak razantní má tato úprava být. Během toho pracuje s hodnotami jako například moment nebo rychlost učení.

Jedním takovým algoritmem je i *adam (Adaptive Moment Estimation)* [KB14]. Ten je charakteristický svou adaptivní rychlostí učení, která mu pomáhá překonávat lokální minima chybové funkce a zároveň předchází oscilaci. Rychlost učení zde lze vnímat jako číslo udávající, jak moc jsou váhy modelu při každé úpravě pozměněny. Pokud je toto číslo příliš malé, bude ke zlepšovaní predikce modelu docházet obecně pomaleji a učení bude vyžadovat delší čas. Zároveň se tím během optimalizace zvyšuje šance na uvíznutí v lokálním minimu chybové funkce, kdy pouze malé úpravy vah nedovolí překonat krátkodobé zhoršení predikce modelu. Na druhou stranu, pokud je rychlost učení příliš velká, úprava vah není dostatečně jemná na to, aby provedla jejich optimální nastavení. Namísto toho začne docházet k oscilaci okolo hodnot vah tohoto nastavení. Daný problém *adam* řeší postupným snižováním této rychlosti pro jednotlivé váhy, jelikož je předpokládáno, že na začátku nebude predikce příliš dobrá a prováděné úpravy budou muset být razantnější. Postupem času, jak se kvalita predikce zvyšuje, je tato rychlost postupně snižována, aby mohla být úprava vah preciznější a mohlo dojít k jejich optimálnímu nastavení.

Návrh metody 4

4.1 Klíčové změny

Navržená metoda navazuje na referenční metodu (2.4) v tom smyslu, že využívá stejného principu průchodu sítí připojováním nových trojúhelníků ke zpracované části, k čemuž analogicky slouží seznam kandidátů na protější vrchol, jakož i ukládání jeho relativního indexu do výstupní sekvence. Provedené změny se pak týkají způsobu, jak je jednotlivých cílů dosaženo. Ty jsou představeny níže.

Výběr kandidátů

Výběr opět staví na úvaze, že protější vrchol se pravděpodobně nachází někde v blízkosti dané hrany. Je proto vytvořena paralelogramová predikce a k ní, pomocí kd-stromu, vybráno k nejbližších neuzavřených vrcholů. Ty budou tvořit seznam kandidátů. Jelikož velikost seznamu je hodnotou k pevně dána, její optimální nastavení se pro jednotlivé sítě může lišit a hlavní roli zde hraje poměr mezi efektivitou komprese a výpočetní složitostí.

Ohodnocení seznamu kandidátů

Pro ohodnocení kandidáta je vytvořen trojúhelník mezi jím a příslušnou hranou. Poté jsou zaznamenávány jeho geometrické vlastnosti a ty předány modelu umělé neuronové sítě, který odhadne pravděpodobnost, se kterou tato dvojice trojúhelník skutečně tvoří. Podle této pravděpodobnosti jsou následně kandidáti sestupně seřazeni a je vytvořen výsledný seznamu.

Výpočet priority hrany

Způsobů, jak postupovat při výpočtu priority, může být více. Vzhledem však k tomu, že ohodnocení jednotlivých kandidátů lze interpretovat jako pravděpodobnost, domnívali jsme se, že namísto omezení se na její výpočet pouze pomocí rozdílu kvality

dvou nejlepších kandidátů, bylo by vhodnější této skutečnosti využít a dále s ní pracovat. Přístupů tak bylo zvoleno více. Ty jsou detailněji popsány v sekci (4.3.2). Jak dále ukáží výsledky experimentů v sekci (5), měly zvolené postupy na kvalitu komprese pozitivní vliv a tento předpoklad se tedy ukázal jako správný.

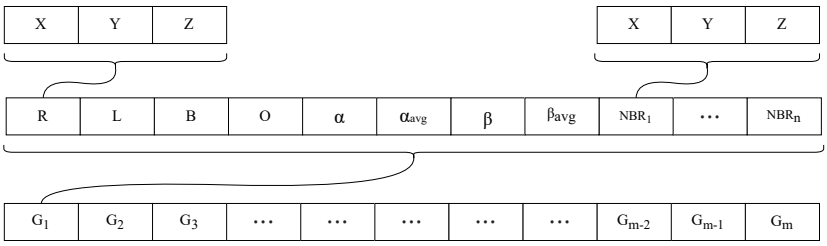
4.2 Příprava a učení modelu neuronové sítě

Tato část se zabývá procesem, kterým bylo postupováno při tvorbě neuronové sítě a na jehož konci stálo vytvoření funkčního modelu, který je schopen spolehlivě určit s jakou pravděpodobností předaná dvojice hrana/vrchol tvoří připojovaný trojúhelník. Proces se řídil modelem ETL (z anglického Extract, Transform, Load) a zahrnoval tak tři klíčové kroky: extrakce, kdy byl proveden sběr a export dat, transformace, při které byla tato data upravena tak, aby na nich bylo možné provést trénink, a načítání, při kterém došlo k převzetí těchto dat metodou, která je využila pro samotné učení modelu. Jednotlivé částí jsou představeny níže.

4.2.1 Sběr dat

Pro trénink neuronové sítě bylo nejprve zapotřebí shromáždit značné množství trénovacích dat. Toho bylo docíleno tak, že do hlavní smyčky původní metody byla vložena část kódu, starající se o jejich sběr. Ta během procesu kódování pro každý nově přidávaný trojúhelník exportovala pozice vrcholů aktivní hrany, protějšího vrcholu a vrcholu výchozího trojúhelníku, jehož byla hrana součástí. Kromě toho bylo exportováno i 20 nejbližších vrcholů od paralelogramové predikce, kteří posloužily jako negativní vzorky učební sady. Takto získaná data byla zaznamenána do externího souboru (jeho struktura je zobrazena na obrázku 4.1) a to v binární podobě, mimo jiné proto, aby se předešlo ztrátě přesnosti, neboť pozice vrcholů byly reprezentovány čísly s plovoucí desetinou čárkou. Tuto extrakci lze v kódu původní metody provést pomocí přepínače *export* ve třídě *Configuration* jeho nastavením na hodnotu *true*. Jako zdroj těchto dat byla použita sada [Koc+19], ze které byla vybrána podmnožina objektů, se kterou je metoda schopna pracovat. Došlo tak k vyloučení například nemanifoldních sítí.

V pozdější fázi vývoje bylo zjištěno, že tato data nemají dostatečnou informační hodnotu k natrénování uspokojivě výkonného modelu. Dodatečně k nim tak byla exportována ještě informace o velikosti celkového a průměrného úhlu v levém a pravém vrcholu aktivní hrany. Uhlem je zde myšlena velikost vnitřního úhlu trojúhelníku, který je daným vrcholem tvořen.

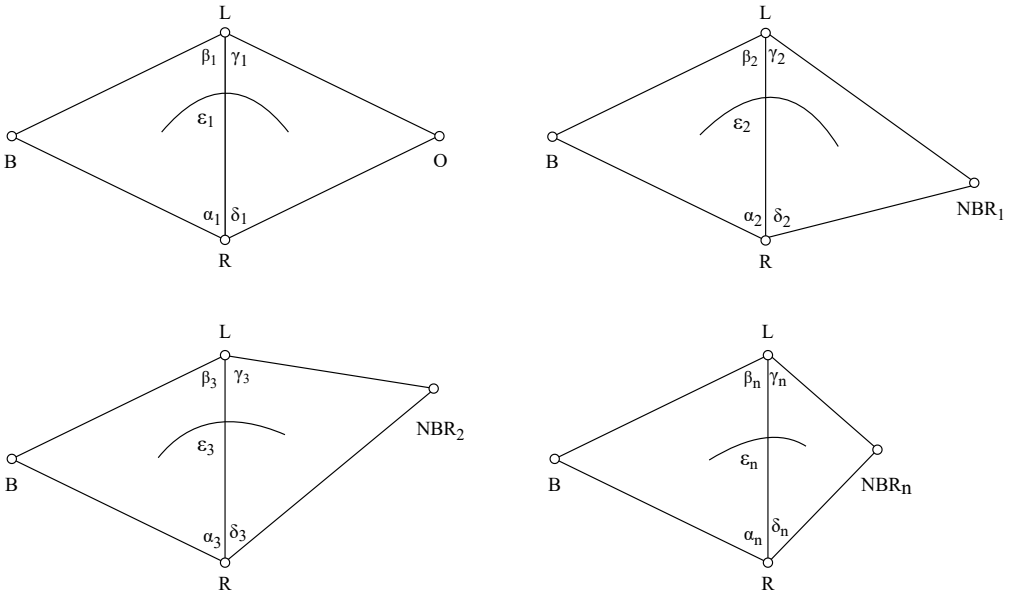


Obrázek 4.1: Schématické znázornění rozložení dat v souboru. G_1 až G_m reprezentují jednotlivé hrany, pro které jsou vždy exportována jejich data. R a L představují pravý a levý vrchol hrany, B je vrchol výchozího trojúhelníku, O je protější vrchol a NBR_1 až NBR_n představují n nejbližších vrcholů od paralelogramové predikce. Každý vrchol je pak popsán třemi souřadnicemi X, Y a Z, jejichž sekvencemi je soubor tvořen. V pozdější fázi byly také přidány hodnoty α a β udávající celkový úhel v pravém a levém vrcholu a α_{avg} a β_{avg} průměrný úhel v trojúhelnících, které tvoří.

4.2.2 Transformace extrahovaných dat

Po extrakci dat byla provedena jejich transformace tak, aby s nimi mohl model efektivně pracovat. Pro každou hranu byl nejprve rekonstituován její výchozí trojúhelník a uloženy velikosti úhlů v jejím pravém a levém vrcholu, později s tím byly připojeny i informace o celkovém a průměrném úhlu, který ve svých trojúhelnících svírají. Následně byly naproti této hraně postupně vytvářeny trojúhelníky vznikající propojováním s jednotlivými vrcholy. Jako první se skutečným protějším vrcholem a následně s ostatními (viz obrázek 4.2). I zde byly zjištěny velikosti úhlů ve vrcholech hrany a zaznamenány. Jako poslední údaj byl zjištěn dihedrální úhel mezi vzniklým a výchozím trojúhelníkem. Těchto devět hodnot tvořilo vstupní parametry, podle kterých byl model v prvních fázích vývoje trénován. Aby byl vzorek kompletní byla ještě ke každé sadě přiřazena hodnota 1 nebo 0 podle toho, zda byl připojený trojúhelník skutečně tvořen s protějším vrcholem. Takto připravená datová sada byla opět v binární podobě uložena do externího výstupního souboru.

Na konci této sekce je vhodné zdůraznit, k čemu byla tato interpretace dobrá a proč modelu nebyly představovány pouze pozice vrcholů s dodatečnou informací o tom, zda se jedná či nejedná o protější vrchol. Důvodem bylo zajištění invariance jednotlivých vzorků vůči jednoduchým transformacím jako je rotace, translace nebo škálování. Lze totiž dokázat, že žádná z těchto operací nemá na velikost získaných úhlů vliv a vzorek tak zůstává nezměněný. Opačně by šlo pak říct, že vzorek je schopný popsat všechny tyto druhy transformací, což přispívá k větší obecnosti a zvyšuje se tím počet případů, který je schopen reprezentovat.



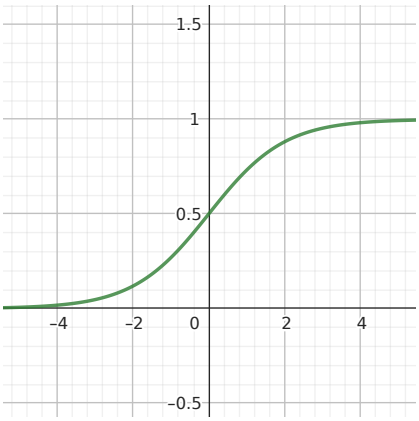
Obrázek 4.2: Reprezentace úhlů v sadě dvou trojúhelníků.

4.2.3 Učení modelu na získaných datech

Po transformaci byla data načtena do programu vytvořeného v jazyce python, kde byl sestaven a trénován model neuronové sítě. K tomu bylo využito externí knihovny *PyTorch verze 2.2.2*. Pro optimalizaci výkonu byly navíc výpočty přesunuty a prováděny na grafické kartě Nvidia GeForce RTX 4070 a to za pomoci knihovny *Cuda verze 12.1*.

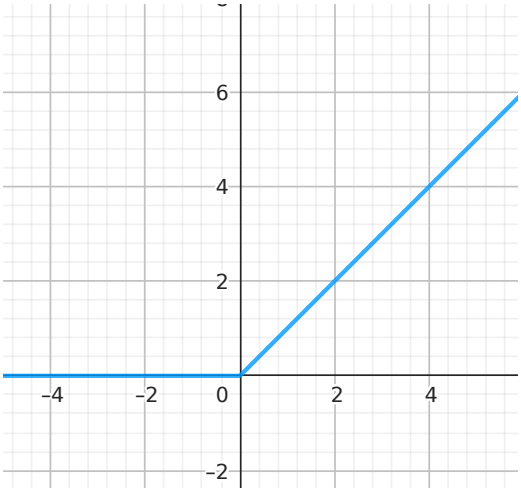
4.2.3.1 Proces učení

Při procesu učení byla použita trénovací sada o velikosti 100 mil. vzorků získaných zpracováním 80 různých trojúhelníkových sítí (viz předchozí metody). Model se skládal z jedné vstupní, jedné výstupní a jedné vnitřní vrstvy. Jednalo se o plně propojené lineární vrstvy, jejichž každý neuron byl propojen se všemi neurony vrstvy následující a mezi jednotlivými vrstvami byla jako aktivační funkce vždy použita funkce *ReLU* (její průběh je znázorněn na obrázku 4.4). Vstupní vrstva měla zprvu pět, poté devět parametrů (odpovídá počtu hodnot popisující trojúhelníkový pár), vnitřní vrstva byla rozšířena na 16 neuronů. Výstupní vrstva obsahovala jeden neuron jehož výstup byl následně transformován pomocí funkce *Sigmoid* (její průběh je znázorněn na obrázku 4.3) tak, aby se jednalo o funkci pravděpodobnosti. Model byl trénován v epochách s velikostí dávky 1024 prvků s náhodným výběrem a v každém kroku byl k optimalizaci využit algoritmus *adam*. Po každé epoše došlo



Obrázek 4.3: Průběh funkce sigmoid, definované jako $f(x) = \frac{1}{1+e^{-x}}$

.



Obrázek 4.4: Funkce ReLU, která je pro nezáporné hodnoty definovaná jako $f(x) = x$ a v ostatních případech je rovna 0.

k vyhodnocení výkonnosti modelu (více v sekci *Určení míry výkonnosti*) a pokud se tato míra oproti původní epoše zlepšila, byla výkonnost zaznamenána a váhy modelu vyexportovány do binárního souboru. Naopak pro případy, kdy dokončení epochy nevedlo ke zlepšení, byl zaveden čítač, který po čtyřech po sobě jdoucích epochách s negativním zlepšením, trénink předčasně ukončil.

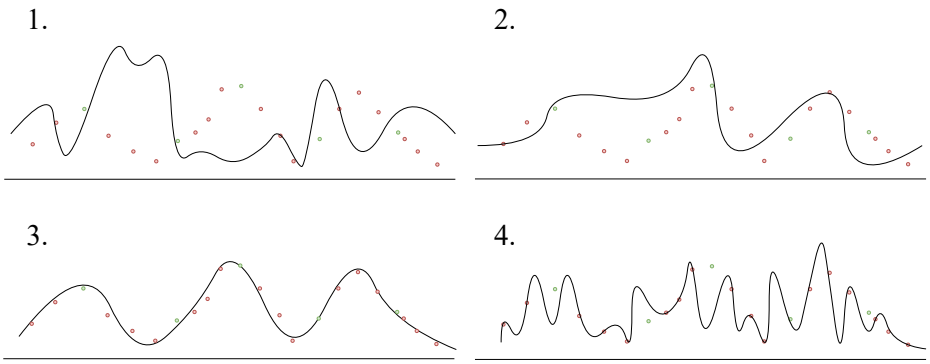
4.2.3.2 Určení míry výkonnosti

Kromě trénovací sady byla vytvořena i sada validační, ta tvořila zhruba 20% celkového objemu zpracovávaných dat a sloužila pouze k určování míry výkonnosti, ne k samotnému tréninku. Její zjištění probíhalo tak, že po každé epoše byla validační

data modelem nejprve ohodnocena. Dále, pokud pravděpodobnost přidělená vzorku přesahovala hranici 50%, byl vzorek označen jako pozitivní, v opačném případě jako negativní. Toto ohodnocení bylo porovnáno se správnými výsledky a rozloženo od dvou tříd klasifikace: *skutečně pozitivní* a *falešně pozitivní*. Míra výkonnosti byla poté určena na základě vzorce 4.1 pro výpočet přesnosti. Ten udával poměr mezi vzorky, které byly správně klasifikovány jako pozitivní, a celkovým počtem pozitivně klasifikovaných vzorků. Tento výpočet byl zvolen za nákladě nevyváženosti vzorkové sady, ve které výrazně převažovaly ty s negativním ohodnocením (kombinace hrana vrchol netvořící připojovaný trojúhelník).

$$Přesnost = \frac{Skutečně\ pozitivní}{Skutečně\ pozitivní + Falešně\ pozitivní} \tag{4.1}$$

Důvodem rozdělení na trénovací a validační sadu, byla snaha o zjištění, jak dobře je model schopný generalizovat jeho poznatky a aplikovat nabyté znalosti na nová data ležící mimo rozsah těch, na kterých byl trénován. Během učení totiž může dojít k takzvanému *přetrénování*. Při něm dochází ke stálému zlepšování výsledků pro trénovací data, to však ale na úkor obecnosti a zhoršováním výsledků pro množinu dat, kterou má trénovaný model reprezentovat. Tento problém je zjednodušeně popsán na obrázku 4.5.



Obrázek 4.5: Křivky popisují měnící se aproximaci dat v průběhu trénování. Červeně jsou vyznačena data trénovací sady a zeleně data ze stejné datové množiny, která však při učení použita nebyla. V první fázi je vidět, že model s náhodně nastavenými vahami není schopen data dobře reprezentovat. Během druhé fáze dochází díky učení ke stálému zlepšovaní jejich aproximace. Ve třetí fázi jsou data optimálně aproximována vzhledem k celé jejich množině. Ve čtvrté fázi dochází k přetrénování, kdy jsou výsledky pro trénovací data zlepšovány na úkor těch, která v trénovací sadě neleží.

4.3 Zavedení modelu do původní metody

Pro zavedení modelu byla do původní metody nejprve importována knihovna *KdTree* obsahující implementaci této struktury. Ta je schopna, na základě předané polohy, k ní vyhledat k nejbližších vrcholů, které se ve stromě nacházejí. Z důvodu toho, že se v síti může nacházet více vrcholů ležících na stejné pozici a původní implementace nebyla schopna tuto situaci dobře reflektovat, byla nad ní vytvořena obalovací třída *MultiValueKDTree*. Ta na každé pozici spravuje seznam indexů vrcholů, které zde leží.

> Z důvodu použití seznamu indexů není jejich pořadí při dekódování jednoznačné. Je tedy možné, že u sítí s překrývajícími se vrcholy dojde během dekódování k jejich záměně. Je-li stěna objektu tvořena vrcholy (4, 5, 6) a sdílí-li např. vrchol 4 svoji polohu s vrcholem 1, může být po zdekódování tato stěna tvořena vrcholy (1, 5, 6).

Následně byl v jazyce C# původní metody implementován model neuronové sítě. Ten je umístěn ve třídě *Neural*. Tato třída mimo jiné obsahuje statickou metodu *LoadBinary*, která je schopna na základě předaného souboru, do kterého bylo exportováno nastavení vah modelu během učení, daný model zrekonstruovat a s exportovanými vahami inicializovat. Cesta k souboru, ze kterého má být model načten je uložena v proměnné *modelSource* ve třídě *Configuration*.

Na počátku algoritmu byly z kódovaného objektu načteny všechny jeho vrcholy a uloženy do globálního pole *points*. To mělo v procesu kódování důležitou roli. Aby bylo možné mezi jednotlivými vrcholy jednoznačně rozlišovat, byl každému přidělen *absolutní index* rovnající se jeho pozici v tomto poli. S tímto identifikátorem pak byly vrcholy postupně vloženy do kd-stromu.

4.3.1 Vytvoření seznamu kandidátů

K vytvoření seznamu kandidátů hrany sloužila metoda *getRankByNN*. Zde bylo pomocí kd-stromu vybráno k nejbližších vrcholů k bodu paralelogramové predikce (Hodnota k je nastavena v proměnná *cndsToFind* ve třídě *Configuration* a před začátkem kódování je sdílena mezi kodérem a dekodéru tak, aby seznamy vytvořené na obou stranách měly vždy stejnou délku.). Každému kandidátovi byla poté pomocí metody *CalculateProbability* přidělena pravděpodobnost, s jakou se jedná o protější vrchol a výsledný seznam byl předán volající metodě.

Co se týče samotné hodnoty *cndsToFind*, její volba dosud představuje jednu z největších nevýhod navržené metody. Bez optimálního nastavení totiž neexistuje žádné pravidlo, které by zaručovalo, že se bude protější vrchol v tomto seznamu nacházet. Zároveň zde není zaveden žádný mechanismus, který by toto nastavení prováděl a její správná volba pro konkrétní síť může být tedy vždy jen odhadována.

Při volbě této hodnoty bude vždy hrát hlavní roli poměr mezi efektivitou kódování a časovou náročností. Z hlediska efektivity je jasné, že čím větší seznam bude, tím větší bude i pravděpodobnost, že obsahuje protější vrchol a nebude nutné tento výpadek dále řešit. Naopak co se týče Časové náročnosti, připomeňme, že výpočetní složitost vyhledávání prvku v kd-stromu je v průměrném případě rovna $O(log(n))$. To je v algoritmu provedeno $cndsToFind$-krát pro každou hranu, kterých je v sítí asymptoticky zhruba $3n/2$. Výsledná složitost pouhého vytvoření všech seznamů kandidátů tak odpovídá hodnotě $O(3n/2 \cdot log(n) \cdot cndsToFind)$. I přes to, že se ve vzorci jedná o konstantu a výpočetní složitost jako takovou zvolená velikost seznamu fakticky neovlivňuje, v praxi může představovat výrazné zpomalení.

4.3.2 Výpočet priority

Výpočet priority byl prováděn metodou *CalcualtePriority*. Ta pro získání seznamu kandidátů využívala zmíněného postupu v sekci (4.3.1). Výsledný seznam byl seřazen a následný výpočet priority pak záležel na nastavení přepínačů *normalization* a *useCertainty* nacházejících se ve třídě *Configuration*.

Přepínač *normalization*

Přepínač *normalization* je proměnná výčtového typu *Normalization*, kteru lze nastavit na hodnoty: *None, Proportional* nebo *SoftMax*. Jeho nastavením, lze pak nad seznamem kandidátů provádět normalizaci jejich pravděpodobnostního ohodnocení vzhledem k ostatním kandidátům. V případě nastavení *None* není normalizace uplatňována. V případě *Proportional* je pravděpodobnost kandidáta vypočtena podle vzorce 4.2, kde p'_i je normovaná pravděpodobnost i-tého kandidáta, p_i je jeho původní pravděpodobnost a n je počet kandidátů v seznamu. *SoftMax* je už jen varianta tohoto vzorce s využitím exponenciální funkce a má tvar 4.3.

$$p'_i = \frac{p_i}{\sum_{j=1}^{n} p_j} \tag{4.2}$$

$$p'_i = \frac{e^{p_i}}{\sum_{i=1}^{n} e^{p_j}} \tag{4.3}$$

Přepínač *useCertainty*

Tento přepínač slouží k výběru strategie toho, jak bude výsledná priorita hrany vypočtena. Pokud byl nastaven na hodnotu *false*, bylo postupováno podle návrhu původní metody a priorita bude určena jako rozdíl pravděpodobnostního ohodnocení dvou nejlepších kandidátů (tu lze však stále ovlivňovat přepínačem *normalization*). Při nastavení na hodnotu *true* je k výpočtu priority využit vzorec 4.4 pro určení

jistoty. V něm c reprezentuje výslednou prioritu, p_1 je pravděpodobnost nejlepšího kandidáta a p_i jsou jednotlivé pravděpodobnosti ostatních kandidátů. Výsledná priorita tedy určuje: "Jaká je pravděpodobnost, že protějším vrcholem je první kandidát a zároveň jím není žádný jiný".

$$c = p_1 \cdot \prod_{i=2}^{n} (1 - p_i) \tag{4.4}$$

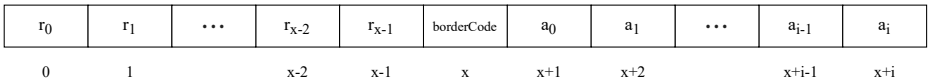
Přepínač *updateQ*

Kromě těchto dvou přepínačů pracuje metoda *CalculateProbability* také s přepínačem *updateQ*. Jedná se o pokus o další zefektivnění kódování a vychází z experimentu, kterým se zabývala již referenční metoda. Priorita hrany je totiž vypočtena pouze jednou a to ve chvíli jejího vložení do prioritní fronty. Jak již bylo ale zmíněno, během algoritmu dochází k postupnému uzavírání vrcholů a to i těch, které mohly být při výpočtu priority použity. Pokud tato situace nastane, znamená to, že momentální priorita hrany ve frontě, neodpovídá té, kterou aktuálně má. V případě referenční metody bylo však zjištěno, že k této situaci dochází v méně než 1% případů a efektivita kódování tak není příliš ovlivněna. Vysvětlení takto nízké četnosti těchto případů může být především ve způsobu, jakým je priority hrany vypočtena. Pokud je uvažována pouze jako rozdíl dvou nejlepších kandidátů, je vcelku malá pravděpodobnost, že právě uzavřeným vrcholem bude jeden z nich.

To se však v případě našeho výpočtu výrazně změní. Je-li kodér přepínačem *useCertainty* nastaven tak, že priorita hrany je vypočtena pomocí vzorce jistoty 4.4, na hodnotě její priority se podílejí všichni kandidáti a šance, že uzavřen bude jeden z nich se tedy značně zvyšuje. Během vypočtu byl tedy pro každý vrchol vytvořen seznam hran, na jejichž prioritě se podílí. Došlo-li následně v průběhu algoritmu k jeho uzavření, byla priorita všech příslušných hran v prioritní frontě aktualizována. Výsledky pro jednotlivé kombinace nastavení těchto přepínačů jsou uvedeny v sekci (5).

4.3.3 Zakódování relativního indexu protějšího vrcholu

Výpočet relativního indexu byl prováděn pomocí metody *getRank*. Při ní nejprve došlo k získání a seřazení seznamu kandidátů předané hrany a to již známou metodu, popsanou v sekci (4.3.1). Protější vrchol hrany byl při kódování samozřejmě znám. Průchodem seznamu byla tedy zjištěna jeho pozice a ta zaznamenána jako relativní index (Jak bylo již zmíněno, seznam ve skutečnosti není tvořen přímo vrcholy, ale seznamem vrcholů ležících na stejné pozici). Zaznamenán byl tedy index



Obrázek 4.6: Přehled všech použitých hodnot ve výstupní sekvenci a jejich význam. Hodnoty 0 až $x - 1$ jsou zde použity pro relativní indexaci r do seznamu kandidátů. Naopak hodnoty od $x + 1$ až $x + i$ jsou absolutní indexy a do pole *points* o velikosti i vrcholů. Hodnotou $x + 1$ je tedy reprezentován první prvek tohoto pole, atd, až po poslední prvek, který reprezentuje hodnota $x + i$. Hodnota k má speciální význam pro označení hraniční hrany a je důkladněji popsaná v sekci (4.3.4).

seznamu, ve kterém vrchol leží. K tomu byla použita pomocná metoda *getIndexPosition*. Za určitých okolností však mohlo dojí k situaci, kdy se vrchol v seznamu kandidátů nenacházel. Typicky v případech, kdy byl protější vrchol příliš vzdálený od paralelogramové predikce a neležel tak mezi k nejbližšími vrcholy.

Pokud tato situace nastala, bylo zapotřebí ji řešit. V našem případě byl tedy index protějšího vrcholu zakódován přímo pomocí svého absolutního indexu v poli *points*. I zde se však může vyskytnou problém v případě, že absolutní index kódovaného vrcholu bude menší než počet vrcholů hledaných do seznamu kandidátů. To by znamenalo, že při dekódování nebude jasné, zda přečtená hodnota má být interpretována jako relativní index v seznamu kandidátů či absolutní index v poli vrcholů. Aby došlo k rozlišení obou situací, bylo-li zapotřebí zakódovat absolutní index vrcholu, byla k němu přičtena hodnota odpovídající počtu hledaných kandidátů. To efektivně posunulo všechny absolutně kódované indexy tak, že hodnoty 0 až $x - 1$ byly rezervovány pouze pro relativní indexaci. Přehled všech použitých hodnot a jejich význam ve výstupní sekvenci je popsán na obrázku 4.6.

4.3.4 Detekce hranice

K detekci hranice byl použit stejný mechanizmus pro výpočet prahové hodnoty, pouze s tím rozdílem, že vybraný práh neodpovídá kvalitě, ale hodnotě pravděpodobnosti, pod kterou pokud ohodnocení nejlepšího kandidáta poklesne, je hrana klasifikována jako hraniční. Zde je důležité poznamenat, že nastavení přepínače *normalization* nemá v tomto případě na pravděpodobnostní ohodnocení kandidáta vliv a to je vždy nenormované.

Mnohem zásadnější rozdíl však představoval výběr symbolu *borderCode* pro identifikaci hraniční hrany. V referenční metodě bylo zapotřebí nejprve použít dočasný symbol -1 a ten následně nahradit inkrementovaným nejvyšším indexem ve výstupní sekvenci tak, aby byla zaručena jeho unikátnost. Výsledný *borderCode* byl pak opět explicitně předán dekodéru. V našem případě však toto předání nebylo nutné a mohlo být využito dvou, již dříve použitých mechanizmů. Prvním bylo infor-

mování dekodéru o počtu prvků, ze kterého má vytvořit seznam kandidátů. Ten byl uložen do proměnné *cndsToFind*. Z něj pak bylo možné určit, že příchozí hodnoty ležící v rozmezí 0 až *cndsToFind* odpovídají relativním indexům v seznamu kandidátů. Díky nulové indexaci byl zároveň tento interval shora otevřený a samotné číslo *cndsToFind* tak nebylo jako index nikdy použito. Druhým mechanizmem bylo zaručení jednoznačnosti relativní a absolutní indexace posunutím absolutních indexů na číselné ose směrem k vyšším hodnotám. Spojení těchto tvou přístupů vedlo k situaci, kdy byl vytvořen unikátní symbol, který nekolidoval jak s relativním tak ani s absolutním označováním vrcholů.

Hodnota *cndsToFind* tedy sloužila k více účelům. Pokud bylo zapotřebí vytvořit seznam kandidátů hrany, byla použita k určení délky tohoto seznamu. Ve chvíli kdy byla přečtena hodnota z kódové sekvence, bylo jejím porovnáním s touto hodnotou určeno, zda se jedná o relativní čí absolutní index dalšího kódovaného vrcholu. Pokud nastala situace, kdy se tyto dvě hodnoty rovnaly, jednalo se o indikaci toho, že právě kódovaná hrana je hranicí sítě.

4.4 Kontrola integrity dat

Pro zjištění, zda komprese probíhala korektně a z uložené konektivity je možné objekt znovu sestavit, byl k navrženému kodéru implementován i příslušný dekodér. Ten postupoval obdobným způsobem prioritizací hran a vytvářením jejich seznamu kandidátů. Její Protější vrchol byl pak vybrán pomocí informace zaslané kodérem a jejím výše uvedeným zpracováním. Vzhledem k tomu, že při hledání k-nejbližších vrcholů pomocí kd-stromu, jsou vrcholy řazeny na základě jejich vzdálenosti, může dojít k nekonzistentnímu výběrů vrcholů, jejichž vzdálenost je stejná. Během dekódování pak může dojít k desynchronizaci průchodu sítí mezi kodérem a dekodérem. Tento problém byl částečně vyřešen sekundárním výběrem kandidátů pomocí rádiusového výběr kandidátů a to podle vzdálenosti nejvzdálenějšího kandidáta. Takto by v seznamu měly být zachyceni všichni kandidáti včetně těch, ležících ve stejné vzdálenosti. Tento seznam byl následně seřazen. Primárně na základě vzdálenosti, sekundárně pak na základě absolutního indexu kandidáta. I včetně případů kde byl v metodě mylně uzavřen protější vrchol, tak bylo možné úspěšně komprimovat a zpětně dekomprimovat 98, 8% všech vzorků. Tento problém by šel pravděpodobně zcela vyřešit vlastní implementací kd-stromu, která by byla schopna výběru k nejbližších prvků stromu s jasně danou strategií.

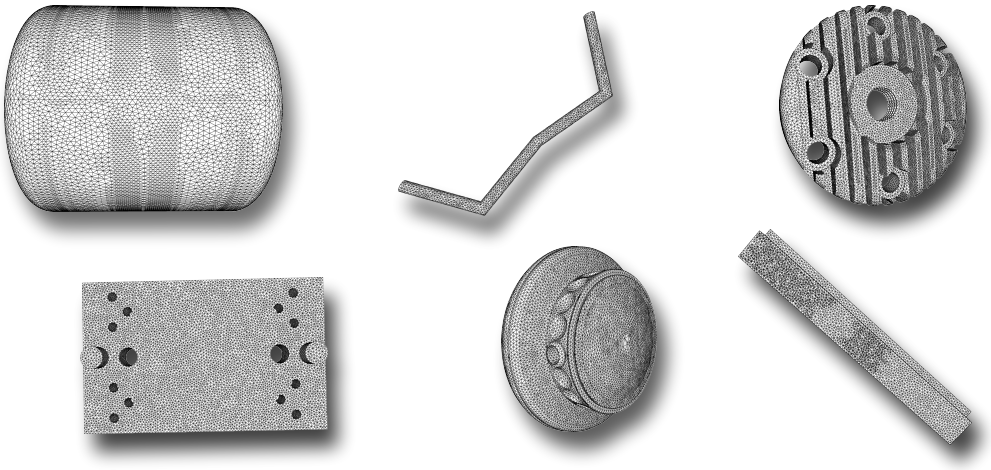
5 Porovnání navržené metody s referenční metodou

V této kapitole jsou shrnuty všechny důležité výsledky a to nejen ty získané na konci vývoje navržené metody, ale také během něj. Kromě finálních měření jsou zde tedy zdokumentovány i výsledky experimentů, které sloužily jako záchytné orientační body a významnou měrou se tak podílely na směru, kterým se další vývoj ubíral.

Pokud není v konkrétních případech stanoveno jinak, uvedená měření jsou provedena s velikostí seznamu kandidátů *cndsToFind* = 20. Konfigurace jednotlivých proměnných a přepínačů je vždy uvedena za označením modelu a má tvar:

- !C = *useCertainty* je nastaven na *false*
- C = *useCertainty* je nastaven na *true*
- None | Pro | Sofmax = Značí nastavení přepínače *normalization*
- U = Pokud je přítomno, přepínač *updateQ* je nastaven na *true*
- <*cndsToFind*> = Pokud je uvedeno, číslo značí velikost seznamu kandidátů

Měření byla prováděna na 800 testovacích vzorcích (jejich ukázka viz obrázek 5.1) ze stejné *ABC* datové sady [Koc+19], ze které byly vybrány i trénovací a validační vzorky pro učení modelů. Výsledná hodnota komprese je pak vždy váženým průměrem získaným na základě komprese jednotlivých sítí a počtu jejich vrcholů.



Obrázek 5.1: Příklady testovacích sítí.

5.1 Dosažené výsledky

Tabulka 5.1: Výsledky komprese navržené metody s různou konfigurací a její procentuální zlepšení oproti referenční metodě. bpf (*bit per face*) zde udává průměrný počet bitů pro zakódování jednoho trojúhelníku, ve sloupci *Zlepšení* je uvedeno procentuální zlepšení komprese oproti referenční metodě.

Konfigurace	bpf	Zlepšení
Referenční m.	0, 16063	
A427_!C_None	0, 13317	17, 10%
A427_!C_Pro	0, 12921	19, 56%
A427_C_None	0, 13325	17, 05%
A427_C_Pro	0, 12590	21, 62%
A427_C_Pro_U_40	**0, 10925**	**31, 99%**

Efektivita navržené metody úzce souvisela s volbou modelu, který byl při kompresi použit (více v sekci (5.2)). Jako nejvhodnější se ukázal model *A427*, se kterým jsou také hodnoty uvedené v tabulce 5.1 naměřeny. U referenční metody byla kvalita komprese naopak značně ovlivněna nastavením jednotlivých vah w_1, w_2, w_3. Vzhledem k tomu, že navržená metoda předpokládala pravidelně teselované sítě, na kterých bylo také měření prováděno, byly pro lepší srovnaní tyto váhy nastaveny na hodnoty, které

jsou v původním článku pro tento druh sítí uvedeny jako nejvhodnější.

5.2 Volba struktury modelu neuronové sítě

Během vývoje byla vytvořena řada modelů, které se lišily v počtu vstupních parametrů, vnitřních vrstev nebo počtu neuronů ve vnitřních vrstvách. Pro snažší orientaci tak bylo zavedeno jejich značení. To mělo vždy tvar *XYZZ*. *X* zde označovalo, zda model využíval informaci o topologii okolní sítě. Pokud ano, začínalo jeho označení písmenem A, pokud ne, byl označen písmenem B. *Y* bylo jednomístné číslo a rovnalo se počtu vnitřních vrstev daného modelu. S tím pak souviselo poslední dvojčíslí *ZZ*, které uvádělo počet neuronů v každé vnitřní vrstvě. Celkově byly navrženy tyto modely:

- B116
- A116
- A216
- A316
- A427
- A250
- A450

Tabulka 5.2: Efektivita komprese pro model *B116*. Sloupec *err* udává průměrný počet výpadků protějšího vrcholu ze seznamu kandidátů na tisíc vrcholů sítě. Záporné hodnoty ve sloupci *Zlepšení* naznačují zhoršení oproti referenční metodě.

Konfigurace	bpf	err	Zlepšení
B116_!C_None	0, 16334	1, 24	−1, 68%
B116_!C_Pro	**0, 14439**	1, 25	**10, 12%**
B116_!C_Softmax	0, 16564	1, 26	−3, 11%
B116_C_None	0, 16357	**1, 10**	−1, 83%
B116_C_Pro	0, 14458	1, 27	10, 00%
B116_C_Softmax	0, 18013	1, 25	−12, 13%

Jak je patrné, model *B116* byl jediným modelem, který nevyužíval informace o stavu topologie sítě v okolí vrcholů aktivní hrany. Tato informace byla pro další modely zavedena právě až na základě jeho výsledků uvedených v tabulce 5.2. Z nich bylo dále usouzeno, že provádět výpočet priority pomocí *Softmax* normalizace není vhodné a v dalších fázích vývoje s ním tak nebylo pracováno. Pro potvrzení této domněnky má v tabulce sloužit sloupec *err*, jehož cílem je demonstrovat, že výrazné rozdíly mezi efektivitou komprese pro jednotlivé konfigurace nejsou způsobeny výpadky protějšího vrcholu ze seznamu kandidátů, nýbrž distribucí jeho výskytu v něm.

Výsledky ostatní modelů v jejich nejlepší konfiguraci jsou dále uvedeny v tabulce 5.3. Kompletní výsledky modelů a jejich konfigurací jsou uvedeny v příloze B. Po-

všimněme si zde, že nezávisle na použitém modelu, se jako nejvhodnější konfigurace ukázalo nastavení přepínačů na použití jistoty a normalizace pomocí proporcionálního rozdělení.

Tabulka 5.3: Porovnání efektivity komprese jednotlivých modelů v jejich nejlepší konfiguraci. Sloupec *Zlepšení* udává procentuální zlepšení oproti referenční metodě.

Konfigurace	bpf	Zlepšení
A116_C_Pro	0, 13848	13, 79%
A216_C_Pro	0, 13127	18, 28%
A316_C_Pro	0, 13141	18, 19%
A250_C_Pro	0, 12717	20, 83%
A427_C_Pro	**0, 12590**	**21, 62%**
A450_C_Pro	0, 14008	12, 80%

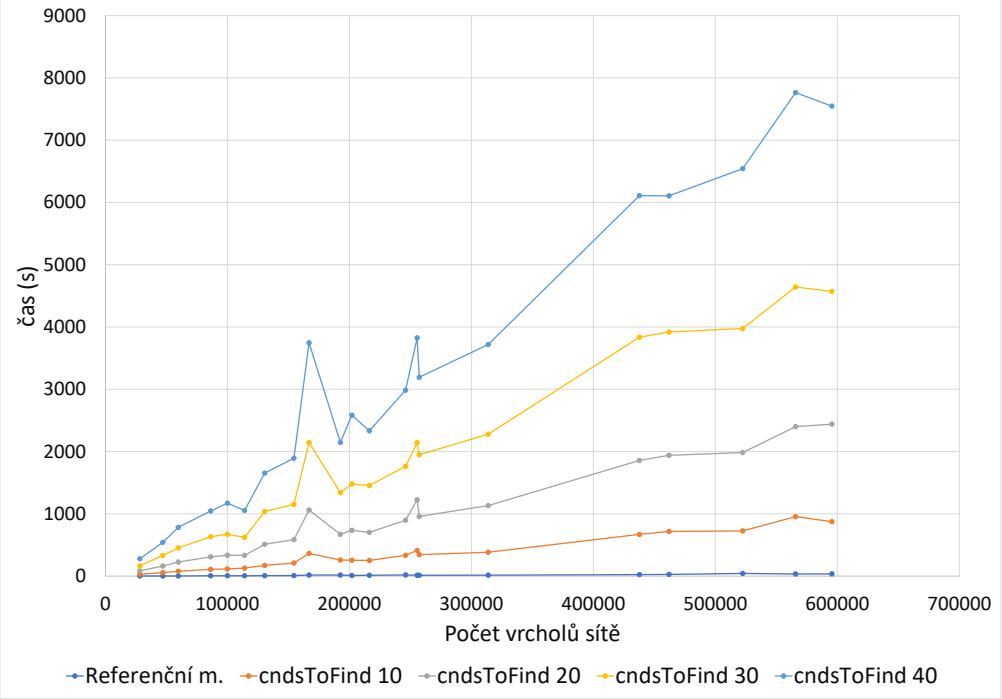
5.3 Volba velikosti seznamu kandidátů

Jak již bylo několikrát zmíněno, určení velikosti seznamu kandidátů je proces, který musí být proveden ještě před zahájením komprese a její vhodné nastavení může být pouze odhadnuto. V tabulce 5.4 jsou proto uvedeny výsledky pro různé nastavení této velikosti.

Tabulka 5.4: Porovnání efektivity komprese při různém nastavení velikosti seznamu kandidátů *cndsToFind*. Hodnota *err* udává průměrný počet výpadků protějšího vrcholu ze seznamu kandidátů na tisíc vrcholů sítě, *Zlepšení* a *Rel. Zlepšení* udává procentuální zlepšení, jak oproti referenční metodě, tak zlepšení oproti předchozí konfiguraci s nižším počtem hledaných kandidátů. Při měření byl použit model *A427* v konfiguraci *C_Pro*.

cndsToFind	bpf	err	Zlepšení	Rel. Zlepšení
10	0, 15124	3, 64	5, 85%	0%
20	0, 12590	1, 11	21, 62%	**16, 75%**
30	0, 12002	0, 50	25, 28%	4, 67%
40	**0, 11829**	**0, 29**	**26, 36%**	1, 44%

Jak je z tabulky 5.4 patrné, rostoucí velikost seznamu pozitivně ovlivňuje efektivitu komprese, a to především snižováním počtu výpadků protějšího vrcholu ze seznamu kandidátů. Ač je toto tvrzení pravdivé v rámci průměru celé testovací sady, není pravidlem, že zvětšení velikosti seznamu kandidátů by automaticky znamenalo zlepšení komprese v rámci jednotlivých souborů. Naopak, byly zjištěny i případy, kdy tato úprava vedla ke zhoršení. Důvodem může být zaprvé normalizace, která pravděpodobnost rozmělní mezi více kandidátů a zadruhé výpočet jistoty, kde více



Obrázek 5.2: Porovnaní času komprese pro různá nastavení velikosti seznamu kandidátů.

kandidátů s nenulovou pravděpodobností snižuje celkovou jistotu pro vývěr nejlepšího kandidáta.

5.4 Aktualizace prioritní fronty

Tabulka 5.5: Porovnání efektivity komprese při aktualizaci prioritní fronty s rozdílnou velikostí seznamu kandidátů. Ve sloupci *Zlepšení* jsou porovnány výsledky s referenční metodou, ve sloupci *Rel. Zlepšení*, pak s navrženou metodu využívající stejný počet hledaných kandidátů, pouze bez aktualizace prioritní fronty. Pro měření byl použit model *A427* v konfiguraci *C_Pro_U*.

cndsToFind	S aktualizací		Bez aktualizace		Rel. Zlepšení
	bpf	Zlepšení	bpf	Zlepšení	
10	0, 14622	8, 98%	0, 15124	5, 85%	3, 32%
20	0, 11821	26, 41%	0, 12590	21, 62%	6, 10%
30	0, 11162	30, 51%	0, 12002	25, 28%	7, 00%
40	**0, 10956**	**31, 80%**	**0, 11829**	**26, 36%**	7, **39%**

Z hodnot uvedených v tabulce 5.5 je zřejmé, že aktualizací prioritní fronty lze dosáhnout poměrně značného zlepšení. Nicméně jak dále ukazuje graf 5.2, toto zlepšení je podmíněno značnou časovou náročností, která činí aktualizaci prioritní fronty pro praktické užití téměř nepoužitelnou. V kontextu provedeného měření byla referenční metoda schopna zakódovat 21 souborů různé velikosti během 5 minut . Oproti tomu navržené metodě s aktualizací prioritní fronty a velikostí seznamu kandidátů 40, trvala tato komprese více než 18 hodin.

5.5 Test symetrie a funkce ohodnocení

Jako poslední experiment byl proveden test symetrie, jehož cílem bylo zjistit, jak modely ohodnocují jednotlivé kandidáty vzhledem k jejich poloze k výchozímu trojúhelníku, a zda-li toho ohodnocení nemůže být symetrické. Motivací k tomuto experimentu bylo zjištění, že u modelu *B116* téměř nezáleželo na pořadí, ve kterém byly úhly v levém a pravém vrcholu aktivní hrany modelu předávány.

Byl tak vytvořen trojrozměrný graf funkce ohodnocení, ze kterého by mělo být jasné, zda je funkce opravdu symetrická. Postup byl následující:

1. Byl zkonstruován výchozí trojúhelník a jeho úhly zafixovány (pro jednoduchost se jednalo o rovnostranný trojúhelník).

2. Byl zvolen úhel, který má připojovaný trojúhelník v protějším vrcholu svírat a ten opět zafixován.

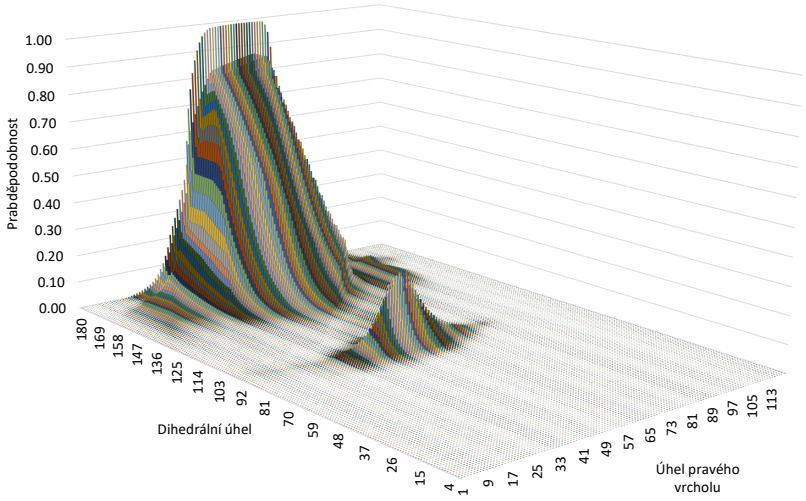
3. Následně byla měněna velikost prvního ze zbylých úhlů a druhý dopočítáván tak, aby se součet vnitřních úhlů připojovaného trojúhelníku vždy rovnal 180°.

4. Tento postup byl pak opakován s různým nastavením dihedrálního úhlu mezi výchozím a připojovaným trojúhelníkem.

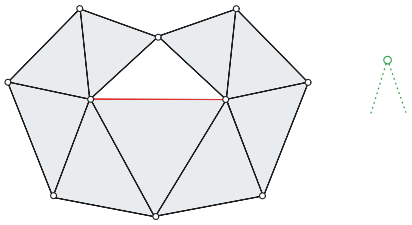
5. Pro každou takto vzniklou dvojici bylo provedeno její ohodnocení a to uloženo.

Jak je zachyceno na obrázku 5.3, výsledek tohoto experimentu jasně ukázal, že ohodnocení je opravdu symetrické. Jak je z obrázku dále možné vyčíst, model zároveň upřednostňoval připojování takových trojúhelníků, které ležely ve stejné rovině jako výchozí trojúhelník a nezavrhoval ani ty, které byly na výchozí trojúhelník kolmé.

Při hlubší analýze však tento graf poukazuje i na hlavní úskalí modelu. Tím, že neuvažuje topologii okolní sítě bude tato predikce za všech okolností vždy stejná a to i v případech, kdy takové ohodnocení nedává smysl. Jako příklad lze uvést



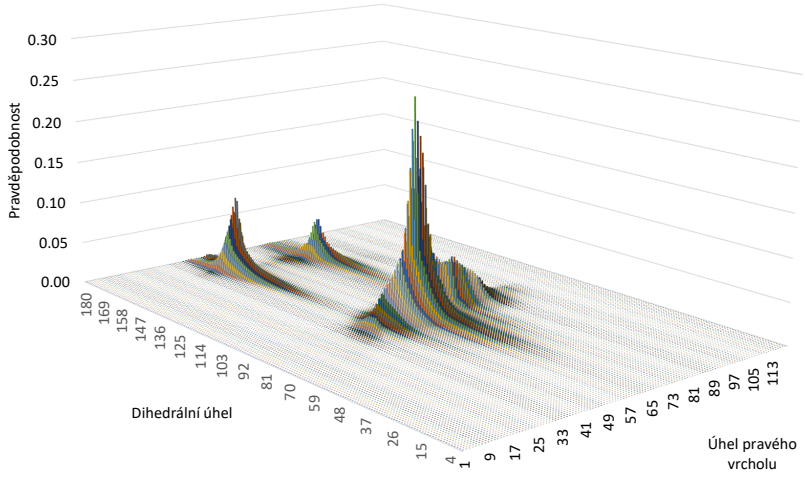
Obrázek 5.3: Graf funkce ohodnocení modelu *B116*.



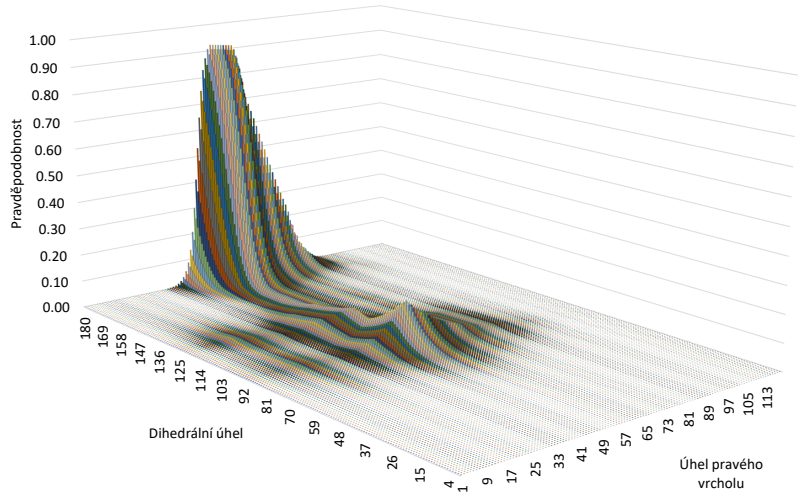
Obrázek 5.4: Případ kdy je požadováno k červeně vyznačené hraně připojit trojúhelník se zeleně znázorněným úhlem v protějším vrcholu.

topologii sítě znázorněnou na obrázku 5.4, kdy je třeba připojit trojúhelník, jehož úhel v protějším vrcholu nedovoluje, aby bez protětí okolní sítě, oba ležely ve stejné rovině. Model zde bude s velkou mírou přesvědčen, že připojovaný trojúhelník by měl stále umístit do této roviny.

Porovnejme nyní tuto modelovou situaci se dvěma modely *A247* a *A116* pracujícími už i s informacemi o okolních bodech. Provedené ohodnocení modelu *A427* je zachyceno na obrázku 5.5 a je zde vidět, že díky dodatečným informacím o topologii, lépe reaguje na vzniklou situaci a trojúhelníky se snaží připojit spíše tak, aby mezi sebou svíraly pravý úhel. Jak je ale dále vidět na grafu ohodnocení (viz obr. 5.6) modelu *A116* tato informace nemusí být každým modelem zpracována správně. Je zde vidět, že zřejmě kvůli nedostatečnému počtu skrytých vrstev nebyl tento model schopný dostatečné asociace mezi těmito informacemi. Ty tak byly použity pouze na další zpřesnění polohy ve stejné rovině nikoliv však ke kalkulaci dihedrálního úhlu. Seznam všech testovaných topologií a k nim odpovídající reakce modelů jsou uvedeny v příloze C.



Obrázek 5.5: Graf funkce ohodnocení modelu *A_427* pro situaci uvedenou na obrázku 5.4.



Obrázek 5.6: Graf funkce ohodnocení modelu *A_116* pro situaci uvedenou na obrázku 5.4.

Závěr 6

V práci byla navržena a implementována metoda pro kompresi konektivity, vycházející z návrhu původní metody založeném na principu ohodnocování kandidátních vrcholů. Bylo představeno několik klíčových změn v původní implementaci jako například: způsob výpočtu priority hran, výběr kandidátních vrcholů či samotné ohodnocení kandidátů. K němu metoda, namísto původního vzorce pro výpočet kvality, využila modelu neuronové sítě. Ten byl natrénován tak, aby byl na základě informací o výchozím trojúhelníku a poloze kandidáta schopný určit, s jakou pravděpodobností je tento vrchol součástí sousedního trojúhelníku, se kterým výchozí trojúhelník sdílí aktivní hranu.

Tyto změny měly na celkovou efektivitu komprese pozitivní vliv a metoda byla za určitých okolností schopna dosáhnout až 31% zlepšení. Nicméně práce zmiňuje i dva zásadní nedostatky. Prvním je doba komprese. Vzhledem k nutnosti provádět pro každého kandidáta jeho ohodnocení pomocí dopředného průchodu všemi vrstvami neuronové sítě, představuje tento proces v kombinaci s tím, že metoda nebyla nijak optimalizována, výrazné zhoršení co se týče časové náročnosti. Jako druhý faktor byl uveden způsob výběru kandidátních vrcholů. Ten je prováděn na základě staticky zadané hodnoty a nedokáže reflektovat situaci, kdy se protější vrchol nenachází mezi vybranými kandidáty. Nastalá situace pak musí být explicitně ošetřena.

Metoda se nicméně stále prokázala jako dostatečně efektivní a je tedy vhodným kandidátem pro další vývoj. V budoucích pracích by bylo vhodné, zaměřit se především na ohodnocování kandidátů pomocí kombinace pozitivně definitních funkcí, které by, jako v původní metodě, dokázaly vytvořit restrikci oblasti, ve které se ještě mohou nacházet vhodnější kandidáti. Byl by tím vyřešen druhý zmiňovaný problém, týkající se chybějícího protějšího vrcholu v seznamu kandidátů, což by mohlo teoreticky vést k dalšímu zefektivnění.

Bibliografie

[AS96] ALGORRI, María-Elena; SCHMITT, Francis. Mesh Simplification. *Comput. Graph. Forum*. 1996, roč. 15, s. 77–86. Dostupné z DOI: 10.1111/1467-8659.1530077.

[DSC22] DUBEY, Shiv Ram; SINGH, Satish Kumar; CHAUDHURI, Bidyut Baran. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*. 2022, roč. 503, s. 92–108. ISSN 0925-2312. Dostupné z DOI: https://doi.org/10.1016/j.neucom.2022.06.111.

[Dvo+23] DVOŘÁK, Jan; KÁČEREKOVÁ, Zuzana; VANĚČEK, Petr; VÁŠA, Libor. Priority-based encoding of triangle mesh connectivity for a known geometry. *Computer Graphics Forum*. 2023, roč. 42, č. 1, s. 60–71. Dostupné z DOI: https://doi.org/10.1111/cgf.14719.

[He+15] HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, s. 1026–1034. Dostupné z DOI: 10.1109/ICCV.2015.123.

[KB14] KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization. *CoRR*. 2014, roč. abs/1412.6980. Dostupné také z: https://api.semanticscholar.org/CorpusID:6628106.

[Koc+19] KOCH, Sebastian et al. ABC: A Big CAD Model Dataset for Geometric Deep Learning. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, s. 9593–9603. Dostupné z DOI: 10.1109/CVPR.2019.00983.

[MMZ23] MAO, Anqi; MOHRI, Mehryar; ZHONG, Yutao. *Cross-Entropy Loss Functions: Theoretical Analysis and Applications*. 2023. Dostupné z arXiv: 2304.07288 [cs.LG].

[MGS07] MARAIS, P.; GAIN, J.; SHREINER, D. Distance-Ranked Connectivity Compression of Triangle Meshes. *Computer Graphics Forum*. 2007, roč. 26, č. 4, s. 813–823. Dostupné z DOI: https://doi.org/10.1111/j.1467-8659.2007.01026.x.

[MP43] MCCULLOCH, Warren S.; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*. 1943, roč. 52, s. 99–115. Dostupné také z: https://api.semanticscholar.org/CorpusID:15619658.

[Ope20] OPENAI. *ChatGPT: Optimizing Language Models for Dialogue* [https://www.openai.com]. 2020. Datum přístupu: 27. dubna 2024.

[Ros57] ROSENBLATT, F. *The perceptron - A perceiving and recognizing automaton*. Ithaca, New York, 1957-01. Tech. zpr., 85-460-1. Cornell Aeronautical Laboratory.

[Ros99] ROSSIGNAC, J. Edgebreaker: connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*. 1999, roč. 5, č. 1, s. 47–61. Dostupné z DOI: 10.1109/2945.764870.

[Süd21] SÜDHOF, Thomas C. The cell biology of synapse formation. *Journal of Cell Biology*. 2021, roč. 220, č. 7, e202103052. ISSN 0021-9525. Dostupné z DOI: 10.1083/jcb.202103052.

[TG98] TOUMA, Costa; GOTSMAN, Craig. Triangle Mesh Compression. In: *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*. 1998, s. 26–34. Dostupné také z: http://graphicsinterface.org/wp-content/uploads/gi1998-4.pdf.

Obsah přílohy

A

Implementovaná metoda stejně tak, jako textu této práce, výsledků měření a ostatních užitých programů je uložena v příloze `A21B0124P_priloha`. Ta je strukturována do několika adresářů jejichž obsah je popsán níže.

A.1 Text práce

Tento adresář obsahuje text bakalářské práce ve formátu `pdf`. Kromě něj, se zde také nachází soubor `BP.tex` a ostatní potřebné soubory k jeho vytvoření pomocí typografického systému TEX.

A.2 Aplikace a knihovny

```
Aplikace_a_knihovny
├─ ncc ..............................Implementace navržené metody
│  ├─ ncc.csproj ............................Hlavní soubor projektu
│  ├─ weights ..........Složka s vahami jednotlivých modelů N. sítě
│  └─ requirements.txt..................Soupis potřebných knihoven
├─ python ..............Skripty pro úpravu dat a trénovaní N. sítě
│  ├─ dataTransform.py .......Vytvoření trénovací sady - popis úhly
│  ├─ neuralGPU.py ..............Trénování a export modelu N. sítě
│  └─ requirements.txt .................Soupis potřebných knihoven
├─ data_extraction ....Extrakce dat úpravou zdroj. kódu ref. metody
└─ probability_function .....Program pro vytvoření grafu ohodnocení
```

A.3 Výsledky

```
Vysledky
├─ Experiments ..................Výsledky provedených experimentů
│  ├─ Cnds_test ..........Výsledky pro různé velikosti s. kandidátů
│  ├─ PQ_update_COMPLET ....Kompletní výsledky měření akt. p. fronty
│  ├─ PQ_Update_Time...Časové měření akt. p. fonty běžící v 1 vlákně
│  ├─ Symmetry_test ....Získané grafy funkcí ohodnocení všech modelů
│  │  ├─ Flat ............................Výsledky pro otevřenou síť
│  │  ├─ Narrow ..........................Výsledky pro zúženou síť
│  │  └─ Closed ..........................Výsledky pro uzavřenou síť
├─ Models_performance ....Výsledky všech modelů a jejich konfigurací
├─ A_RESULTS.xlsx ................Shrnutí výsledků pro modely AYZZ
└─ B_RESULTS.xlsx ................Shrnutí výsledků pro model B116
```

A.4 Vstupní data

Obsahují citaci volně dostupné, datové sady, která byla v práci použita.

A.5 Readme.txt

Jedná se o textový soubor, který shrnuje obsah celé přílohy zcela identickým způsobem jako tato sekce.

Výsledky komprese pro jednotlivé konfigurace modelů

B

Výsledky jsou naměřeny s velikostí seznamu kandidátů *cndsToFind* = 20

Tabulka B.1: Výsledky komprese pro model *A116*

Konfigurace	bpf	Zlepšení
A116_!C_None	0, 17877	−11, 29%
A116_!C_Pro	0, 13955	13, 12%
A116_C_None	0, 17991	−12, 00%
A116_C_Pro	**0, 13848**	**13, 79%**

Tabulka B.2: Výsledky komprese pro model *A216*

Konfigurace	bpf	Zlepšení
A216_!C_None	0, 15316	4, 66%
A216_!C_Pro	0, 13291	17, 26%
A216_C_None	0, 15279	4, 88%
A216_C_Pro	**0, 13127**	**18, 28%**

Tabulka B.3: Výsledky komprese pro model *A316*

Konfigurace	bpf	Zlepšení
A316_!C_None	0, 15464	3, 73%
A316_!C_Pro	0, 13256	17, 48%
A316_C_None	0, 15487	3, 59%
A316_C_Pro	**0, 13141**	**18, 19%**

Tabulka B.4: Výsledky komprese pro model *A250*

Konfigurace	bpf	Zlepšení
A250_!C_None	0, 13056	18, 72%
A250_!C_Pro	0, 13006	19, 03%
A250_C_None	0, 13051	18, 75%
A250_C_Pro	**0, 12717**	**20, 83%**

Tabulka B.5: Výsledky komprese pro model *A450*

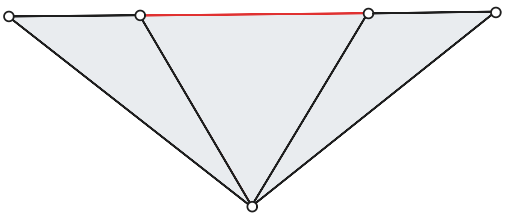
Konfigurace	bpf	Zlepšení
A450_!C_None	0, 15168	5, 58%
A450_!C_Pro	0, 14303	10, 96%
A450_C_None	0, 15188	5, 45%
A450_C_Pro	**0, 14008**	**12, 80%**

Grafy ohodnocení vzhledem k topologii sítě

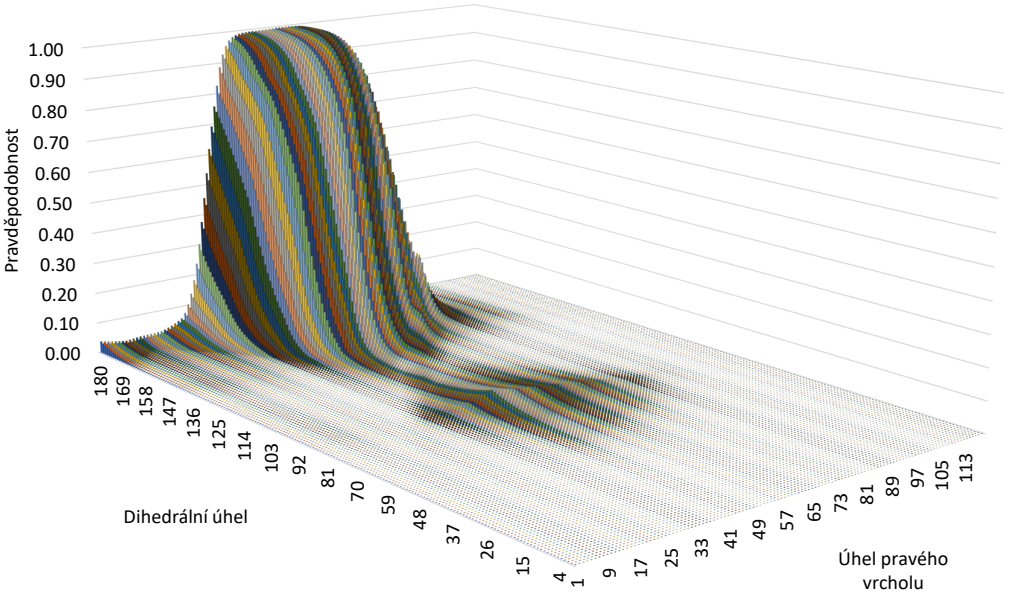
C

C.1 Otevřená síť

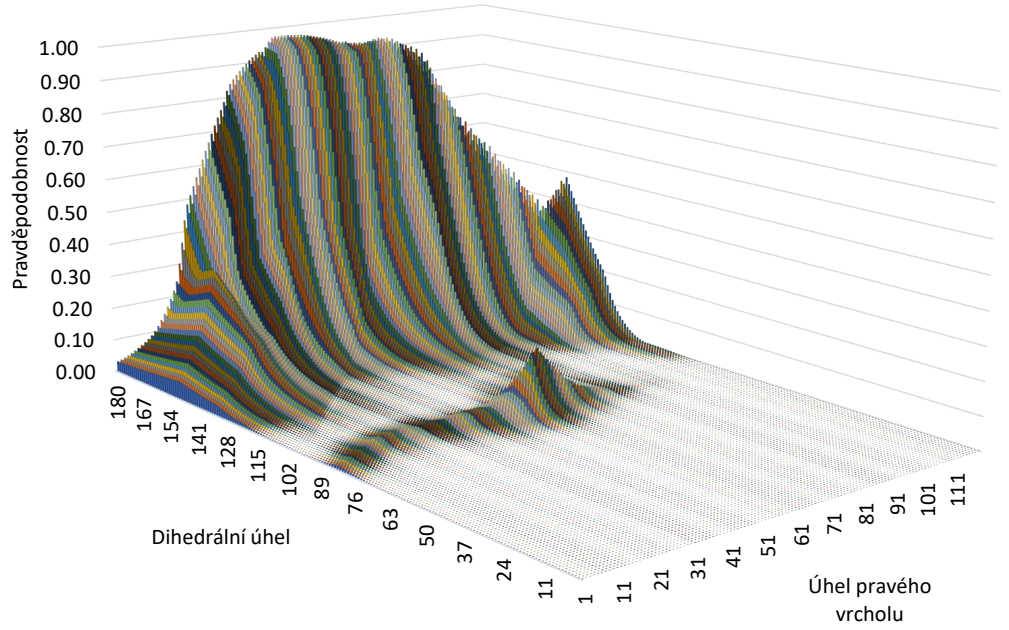
Situace, kdy je celkový úhel jak v levém tak pravém vrcholu hrany roven 180°, jeho průměrný úhel je tak 90° (180° děleno 2 trojúhelníky).



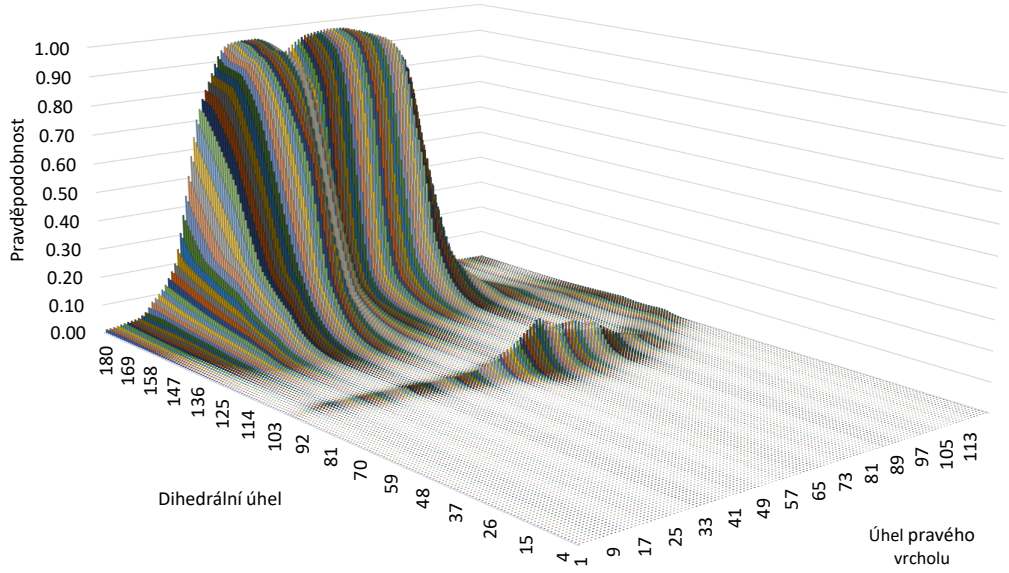
Obrázek C.1: Topologie otevřené sítě.



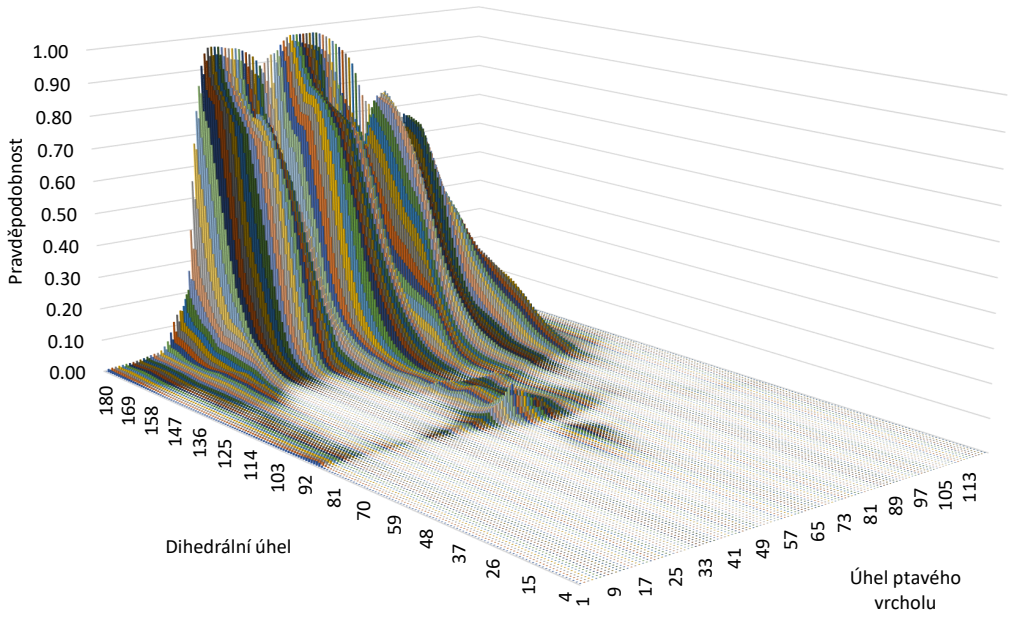
Obrázek C.2: Funkce ohodnocení pro model *A116* v otevřené síti.



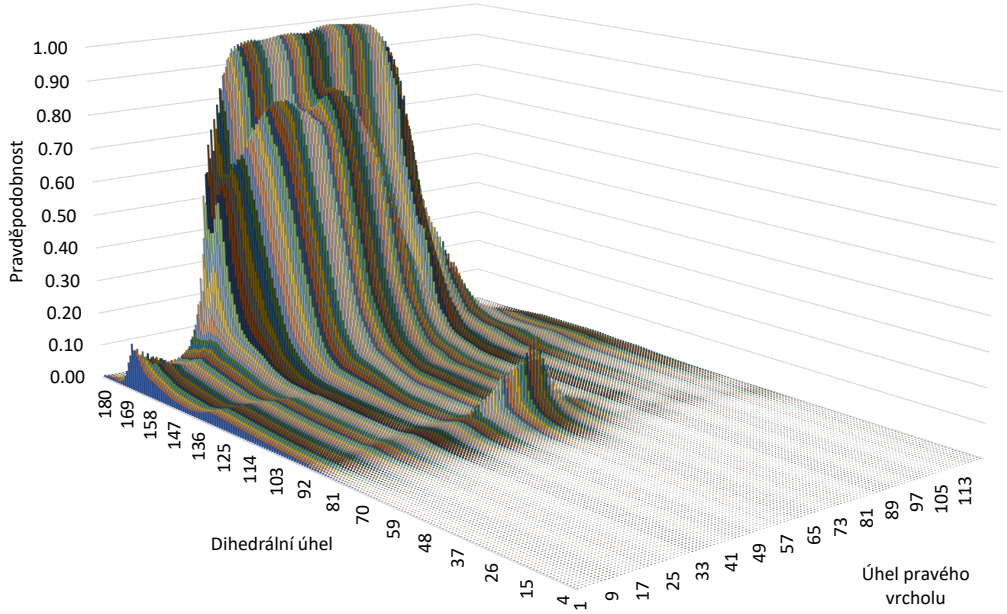
Obrázek C.3: Funkce ohodnocení pro model *A216* v otevřené síti.



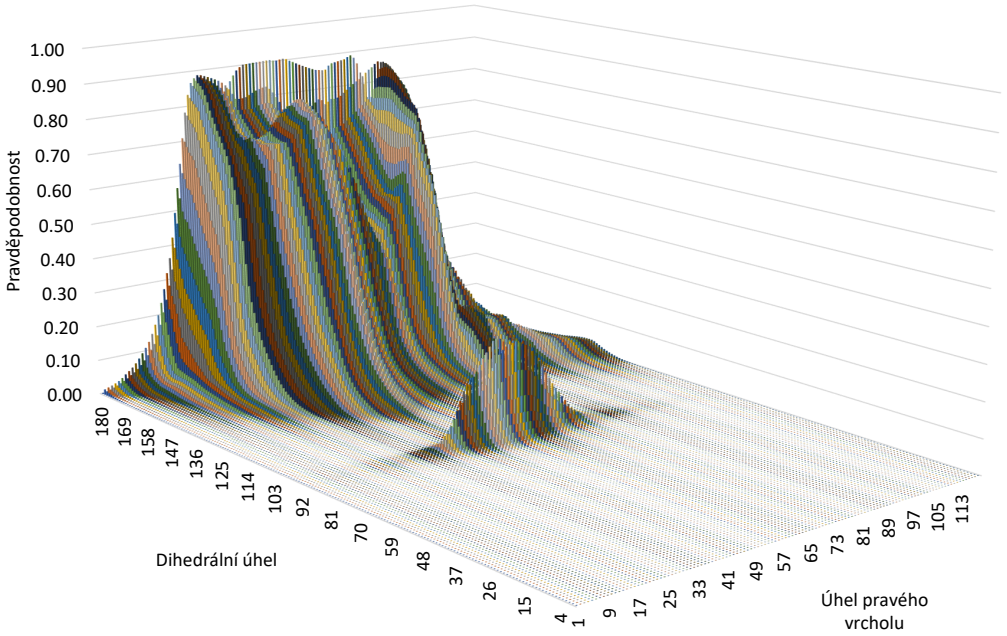
Obrázek C.4: Funkce ohodnocení pro model *A316* v otevřené síti.



Obrázek C.5: Funkce ohodnocení pro model *A250* v otevřené síti.



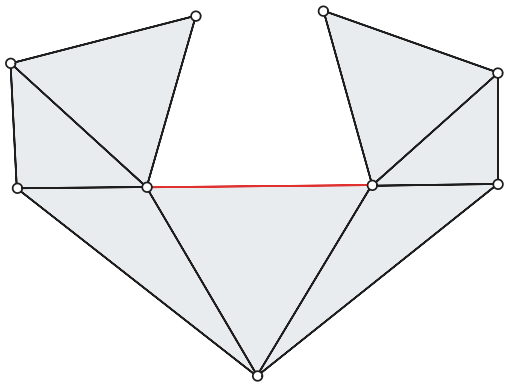
Obrázek C.6: Funkce ohodnocení pro model *A427* v otevřené síti.



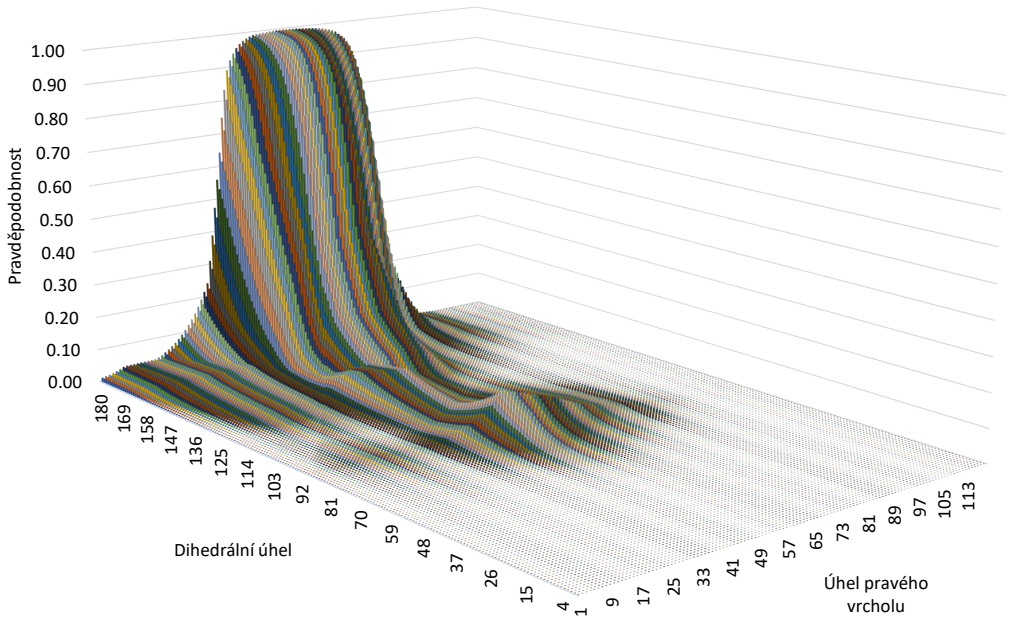
Obrázek C.7: Funkce ohodnocení pro model *A450* v otevřené síti.

C.2 Zúžená síť

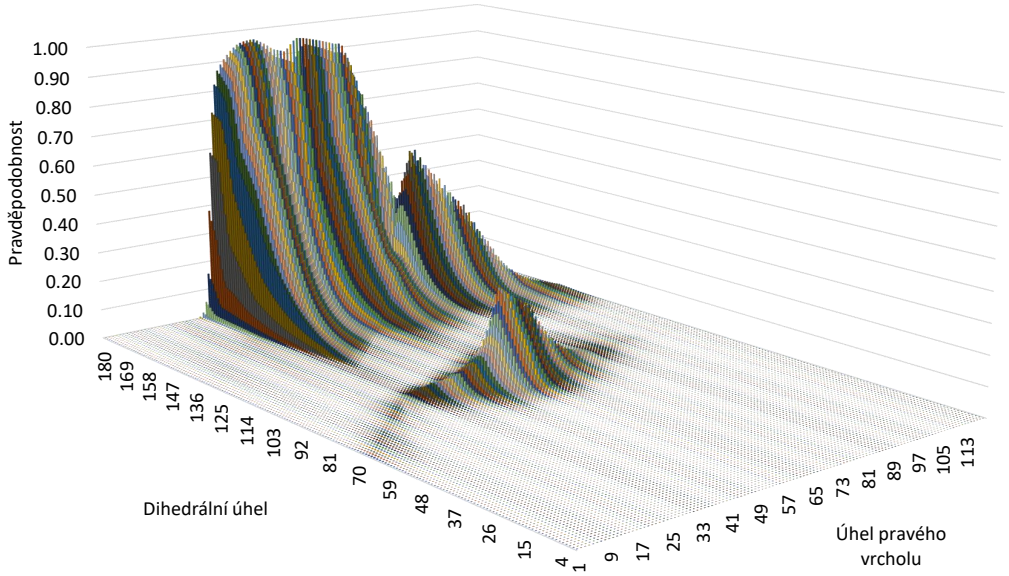
Situace, kdy je celkový úhel jak v levém tak pravém vrcholu hrany roven 280°, jeho průměrný úhel je tak 70° (280° děleno 4 trojúhelníky).



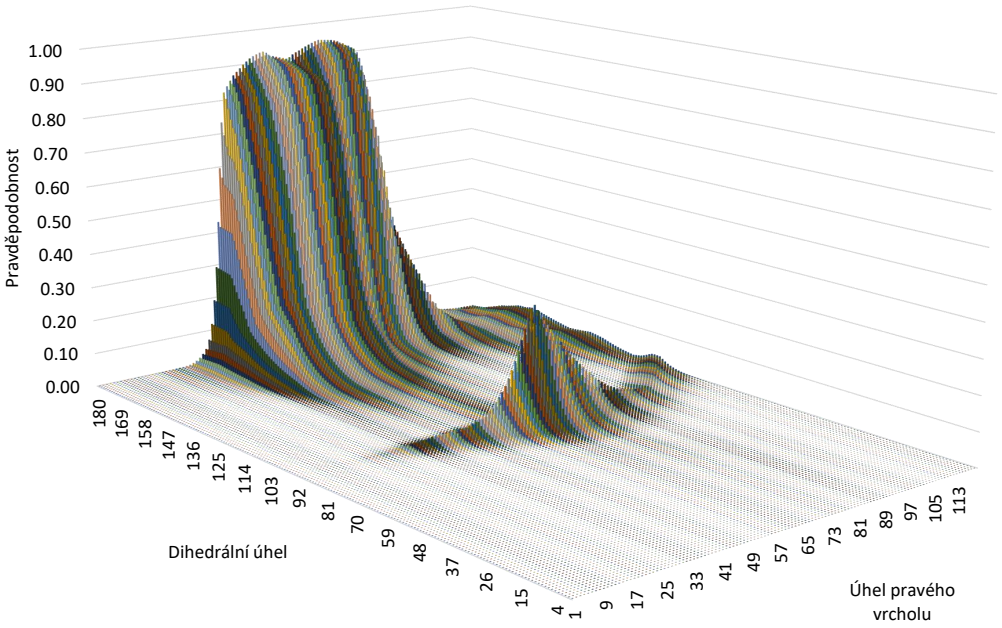
Obrázek C.8: Topologie zúžené sítě.



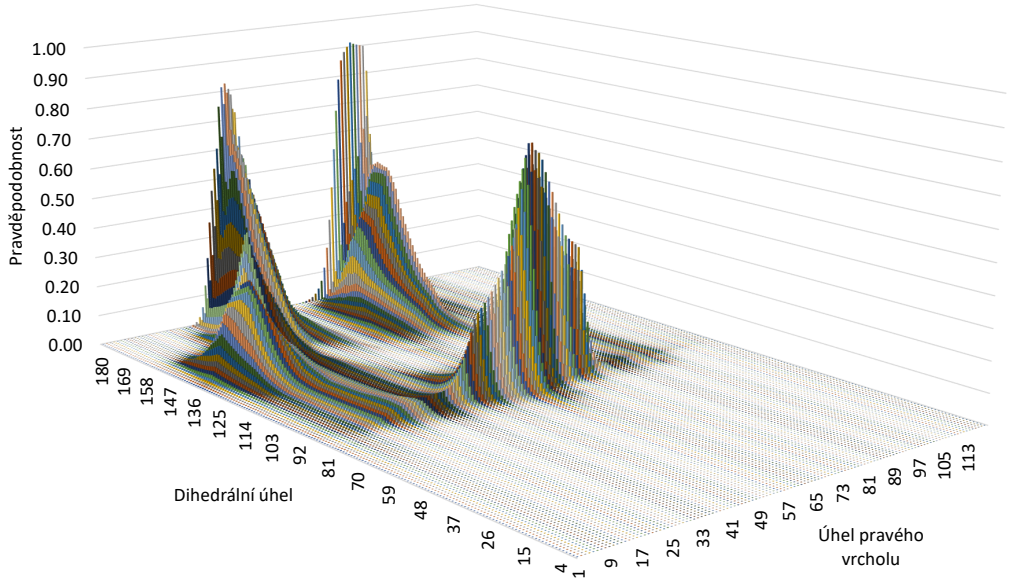
Obrázek C.9: Funkce ohodnocení pro model *A116* ve zúžené síti.



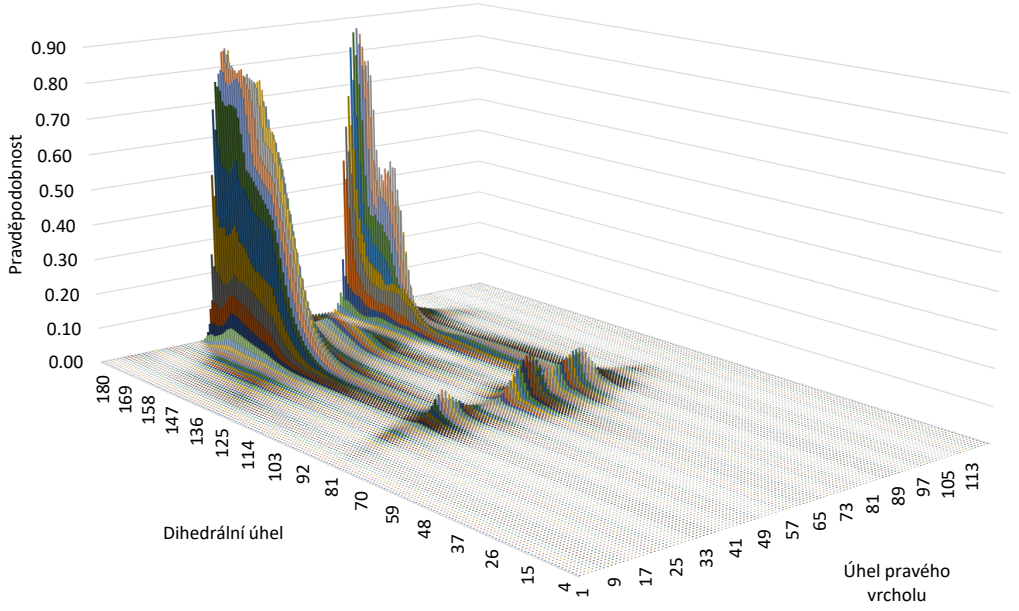
Obrázek C.10: Funkce ohodnocení pro model *A216* ve zúžené síti.



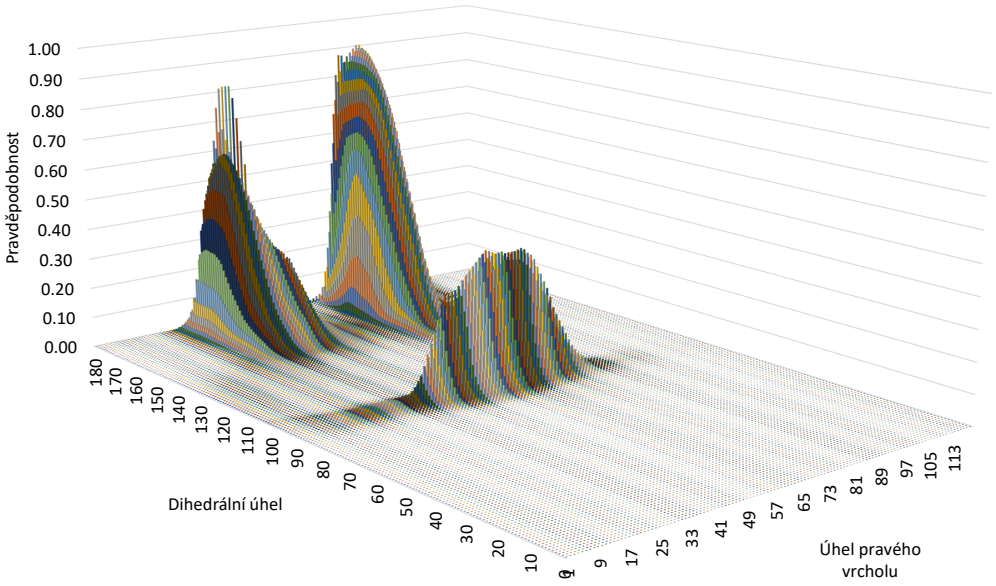
Obrázek C.11: Funkce ohodnocení pro model *A316* ve zúžené síti.



Obrázek C.12: Funkce ohodnocení pro model *A250* ve zúžené síti.



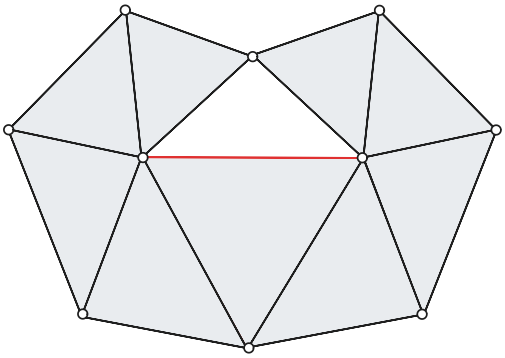
Obrázek C.13: Funkce ohodnocení pro model *A427* ve zúžené síti.



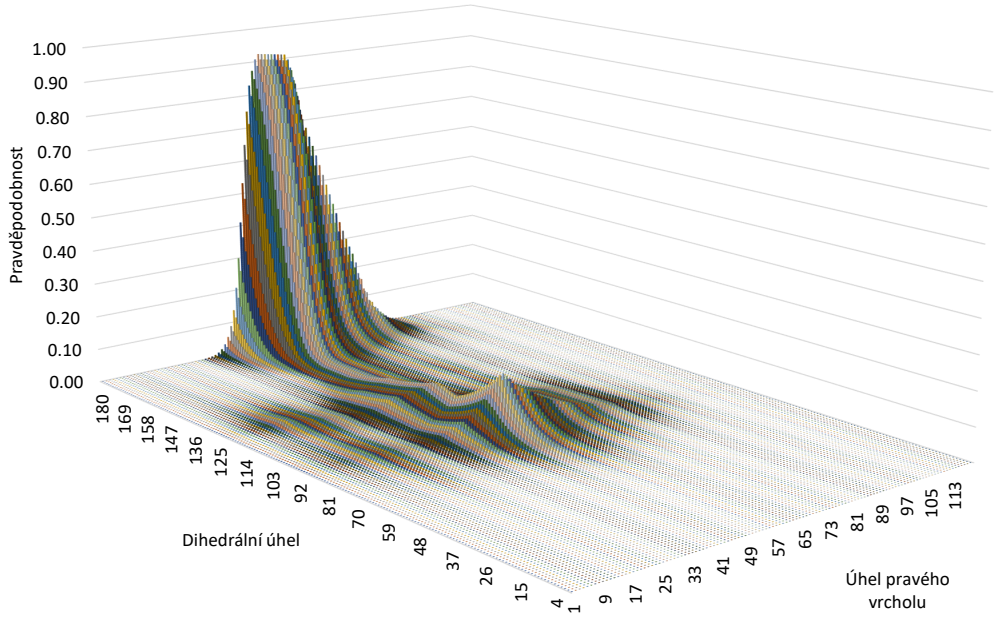
Obrázek C.14: Funkce ohodnocení pro model *A450* ve zúžené síti.

C.3 Sevřená síť

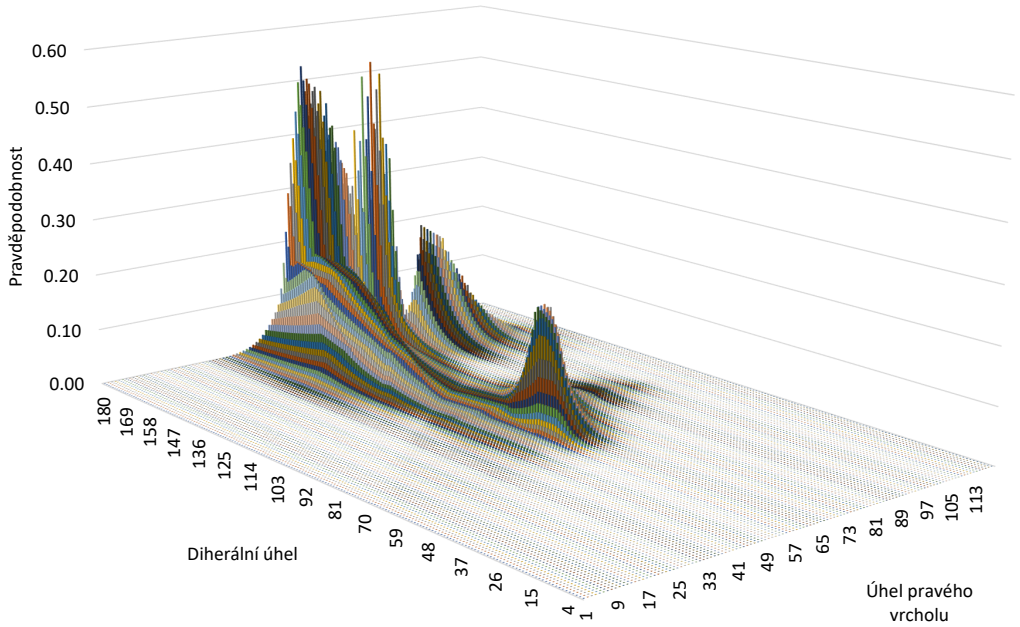
Situace, kdy je celkový úhel jak v levém tak pravém vrcholu hrany roven 315°, jeho průměrný úhel je tak 63° (315° děleno 5 trojúhelníky).



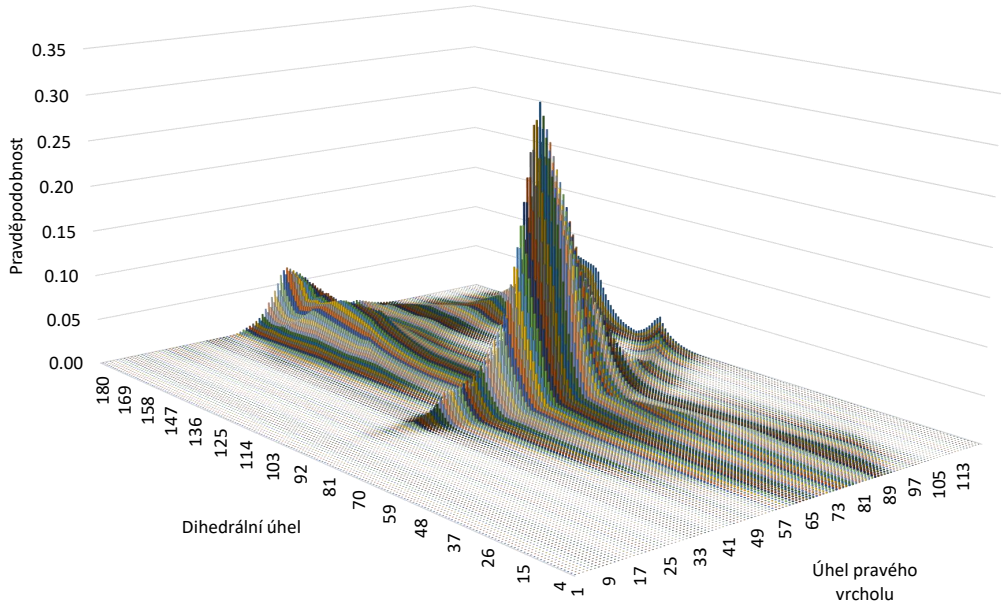
Obrázek C.15: Topologie sevřené sítě.



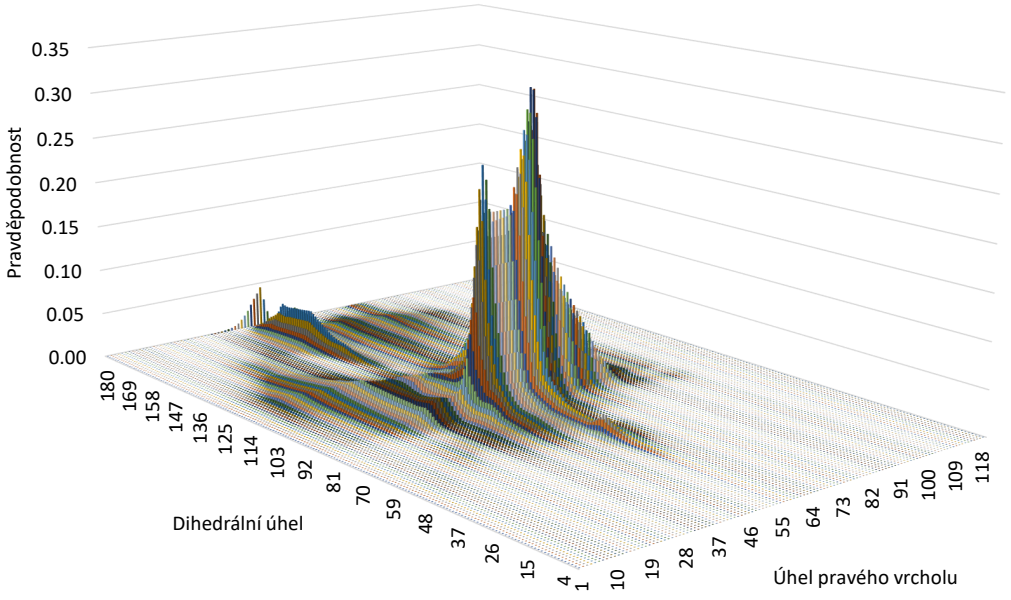
Obrázek C.16: Funkce ohodnocení pro model *A116* v sevřené síti.



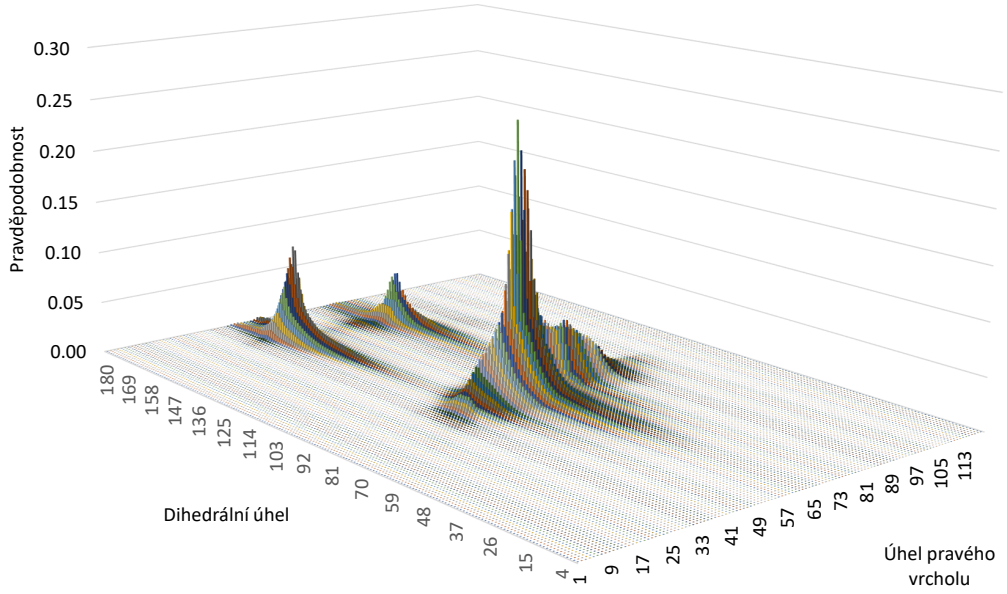
Obrázek C.17: Funkce ohodnocení pro model *A216* v sevřené síti.



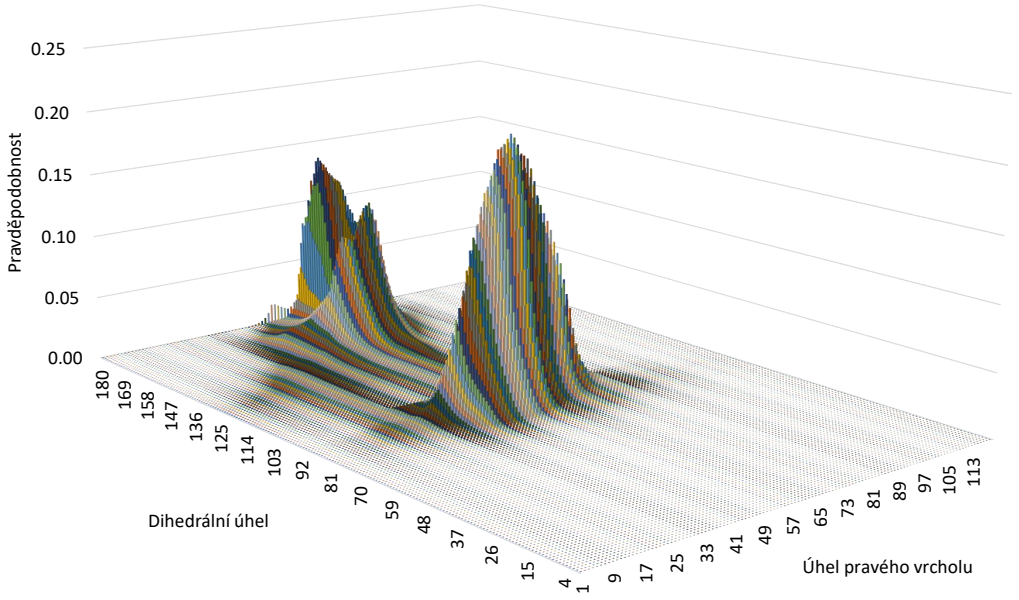
Obrázek C.18: Funkce ohodnocení pro model *A316* v sevřené síti.



Obrázek C.19: Funkce ohodnocení pro model *A250* v sevřené síti.



Obrázek C.20: Funkce ohodnocení pro model *A427* v sevřené síti.



Obrázek C.21: Funkce ohodnocení pro model *A450* v sevřené síti.