Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Implementace umělé neuronové sítě jako prvku pro expertní rozhodování o přidělení nástupištní koleje v osobních železničních stanicích

Bc. Jan Podlešák

Diplomová práce
2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra softwarových technologií
Akademický rok: **2008/2009**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jan PODLEŠÁK**

Studijní program: **N2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Implementace umělé neuronové sítě jako prvku pro expertní rozhodování o přidělení nástupištní koleje v osobních železničních stanicích**

Zásady pro vypracování:

V simulačních modelech osobních železničních stanic s uvažováním příjezdu zpožděných vlaků vzniká problém s určením náhradní nástupištní koleje pro takový vlak. Pro určení nástupištní koleje lze, kromě jiných metod, využít také neuronové sítě. Cílem diplomové práce je navrhnout, implementovat a ověřit chování vybrané umělé neuronové sítě.
Vstupy:
- vlaky osobní dopravy (vstupní kolej do stanice, čas příjezdu, čas odjezdu, výstupní kolej ze stanice, délka vlaku, datumová omezení jízdy vlaků apod.),
- uspořádání kolejiště v osobní stanici (možnosti využití jednotlivých nástupištních kolejí pro vstupní, resp. výstupní koleje do/ze stanice, délky nástupištních kolejí apod.),
- staniční intervaly,
- kritéria pro hodnocení nástupištních kolejí,
- kriteriální matice charakterizující situaci ve stanici.

Výstupy:
- grafické znázornění obsazení kolejí v osobní stanici,
- vybraná kolej pro zpožděný přijíždějící vlak na základě vyhodnocení umělé neuronové sítě.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Prohlašuji:
Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.
Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 9. 5. 2009

Jan Podlešák

PODĚKOVÁNÍ

Touto cestou bych rád vyjádřil své vřelé poděkování panu Ing. Michaelu Bažantovi, Ph.D. za poskytnuté rady a cenné připomínky, které pro mne při psaní této práce byly neocenitelnou pomocí, a za čas, ve kterém se mi věnoval při odborných konzultacích.
Dále bych rád poděkoval všem, kteří mě podpořili a pomohli při napsání této práce, zejména své rodině a nejbližším za jejich podporu během studia.

V Pardubicích 9. 5. 2009

ANOTACE

Práce se zaměřuje na naimplementování umělé neuronové sítě jako rozhodovacího prvku pro přidělení náhradní nástupištní koleje pro zpožděný vlak. Obsahuje popis kritérií a postup výběru náhradní koleje. Je zde rozebrána problematika neuronových sítí, jejich parametrizace a algoritmus učení Back-propagation. Praktické použití je ukázáno na vytvořeném programu.

KLÍČOVÁ SLOVA

náhradní nástupištní kolej; zpožděný vlak; kritéria; neuronová síť; Back-propagation

TITLE

Neural network’s implementation as an element of expert decision-making within platform track assignment problem in passenger railway stations

ANNOTATION

The work focuses on the implementation of neural network as decision-making element for the allocation of alternative track for a delayed train. It contains a description of the criteria and procedures for selecting a replacement track. There are characterization issues of neural networks, their parameterization and Back-propagation learning algorithm. Practical application is shown in the creation of a program.

KEYWORDS

alternative platform track; delayed train; criteria; neural network; Back-propagation

OBSAH

SEZNAM OBRÁZKŮ

Obr. 41 Levá rotace AVL stromu. Zdroj: [13] Obr. 42 Pravá rotace AVL stromu. Zdroj: [13].....45

ZKRATKY

BVS	binární vyhledávací strom
CSV	Comma-separated values (hodnoty oddělené čárkami)
ČD	České dráhy
GVD	grafikon vlakové dopravy
KJŘ	knižní jízdní řád
Os	osobní vlak
R	rychlík
RZG	rozkaz o zavedení GVD
Sp	spěšný vlak
UML	Unified Modeling Language (univerzální jazyk pro vizuální modelování systémů)
UNS	umělá neuronová síť
UP	Unified Process (metodika tvorby softwarového vybavení)
žst.	železniční stanice

1 Úvod

Při zpoždění vlaku je třeba především zachovat provozní výkonnost stanice, a také to aby ve stanici zbytečně nedocházelo k dalšímu zpožďování vlaků. Tyto úkoly jsou závislé na celé řadě činitelů jako např. uspořádání kolejiště, jeho technickém vybavení, systému sdělovacího a zabezpečovacího zařízení a dalších. Možností jak přispět k zajištění výše uvedeného je výběr vhodné nástupištní koleje pro zpožděný vlak.

V simulaci provozu železniční stanice vyvstává otázka jak zajistit, aby se simulační mechanismy při přidělování nástupištní koleje svým chováním co nejvíce přibližovaly realitě. Z tohoto důvodu byla vytvořena čtyři kritéria výběru koleje *A–D*, která byla odvozena ze znalosti práce řídících pracovníků (viz kap. 2.3). Kritéria však sama o sobě nestačí, je třeba stanovit jejich váhu pomocí vhodné metodiky, která určí relativní důležitost těchto kritérií a rozhodne o přidělení správné koleje. K tomuto účelu lze použít několik metod např. matematické metody vícekriteriálního hodnocení variant, metody využívající fuzzy logiku nebo umělé neuronové sítě.

Cílem této práce je implementace řešení pomocí neuronové sítě. Před samotnou aplikací neuronové sítě bude nutné rozhodnout o vhodném typu použité sítě, o vhodném trénovacím vzoru a zvolením správného učebního procesu (viz kap. 3).

Vlastní implementace bude realizována v prostředí jazyka C#. Cílem je vytvoření programu, který bude rozdělen na dvě části:

- první část bude umožňovat zachycení topologie železniční stanice a zadávání parametrů přijíždějících a odjíždějících vlaků,
- druhá část se bude zabývat realizací výpočtu hodnot kritérií výběru koleje, sestavení kriteriální matice (viz kap. 2.4) a rozhodování neuronové sítě o přidělení vhodné nástupištní koleje pro zpožděný vlak.

Navržená implementace pro určení nástupištní koleje přijíždějícímu zpožděnému vlaku bude ověřena na reálných datech v případové studii žst. Praha hlavní nádraží (viz kapitola 5).

2 Problém přidělení nástupištní koleje pro zpožděný vlak

Při simulaci provozu osobní železniční stanice vzniká problém s určením vhodné nástupištní koleje pro přijíždějící zpožděné vlaky. Bereme přitom v úvahu stanice, kde lze u nástupišť vybírat z několika kolejí. Přidělení koleje musí vycházet z rozhodovacích mechanismů standardně uplatňovaných dispečery v praxi. Výsledky simulace by tedy měly do jisté míry kopírovat výsledky rozhodnutí těchto řídících pracovníků.[1]

2.1 Vymezení pojmů

Před vlastním řešení daného problému je třeba objasnit a definovat pojmy, ze kterých budeme vycházet.

2.1.1Nástupištní kolej

Koleje ve stanicích se dělí na koleje traťové, staniční a pro zvláštní účely. Traťové koleje jsou průběžné koleje na širé trati navazující ve stanici v přímém směru na obou zhlavích koleje hlavní. Koleje pro zvláštní účely jsou koleje odvratné, záchytné a vlečkové. Staniční koleje se dělí na dopravní, hlavní, předjízdné a manipulační.[2]

Nástupištní kolej je staniční kolej, na které je cestujícím umožněn nástup nebo výstup z/do vlakové soupravy. Z toho vyplývá, že je vždy přidružena k některému nástupišti ve stanici.

2.1.2Vstupní, výstupní kolej

Vstupní resp. výstupní koleje jsou traťové koleje určené pro vjezdy resp. odjezdy vlaků ze stanice. Tyto navazují na staniční koleje dopravní a předjízdné, které slouží pro křižování a předjíždění vlaků na resp. z nástupištních nebo jiných staničních kolejí.[2]

2.1.3Přípojný vlak

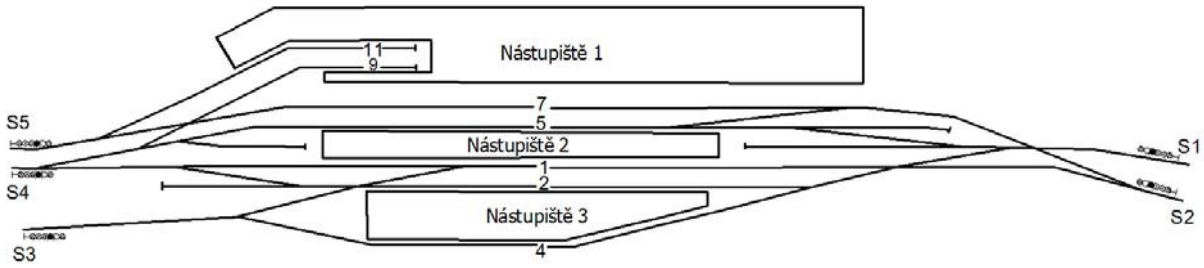
Pro účely čekacích dob je považován za první vlak ten vlak, od něhož je zajištěn přípoj. Vlak, na který je zajištěn přípoj, je považován za druhý vlak. Přípojnými vlaky jsou vlaky, u nichž interval mezi pravidelným příjezdem prvního vlaku a pravidelným odjezdem druhého vlaku je shodný nebo větší než doba potřebná na přestup mezi těmito vlaky ve stanici.

Přípojnými vlaky nejsou:

a) vlaky opačného směru téže trati (výjimky stanoví RZG[1]),
b) vlaky zastavující v různých nádražích jedné obce nebo v různých nádražích (odlišených rozdílným tarifním názvem) téže stanice, musí-li cestující při přestupu opustit její obvod,
c) vlaky, u nichž není dodržena doba potřebná na přestup podle čl.101 a) nebo b) v předpisu ČD D4[2], je-li tato skutečnost vyznačena příslušnou značkou v KJŘ[3] ČD. Chybí-li mimořádně toto označení v KJŘ, považují se tyto vlaky za přípojné a přestup musí být umožněn,
d) vlaky, u kterých je sice dodržena doba potřebná na přestup, ale u nichž je tabulkou A stanoveno, že "vlak nečeká". Tato skutečnost musí být rovněž vyznačena v KJŘ. Chybí-li toto označení v KJŘ, považují se tyto vlaky vždy za přípojné.[3]

2.2 Určení množin přípustných kolejí

Před vlastním přidělováním kolejí zpožděným vlakům je potřeba ve stanici stanovit všechny tzv. množiny přípustných kolejí. Ty jsou určeny spojením vstupních (výstupních) kolejí s nástupištními prostřednictvím předjízných kolejí. Vzniknou tak množiny $K_{Si,Sj}$, kde Si je označení vstupní a Sj označení výstupní koleje. Pro daný vlak lze tyto množiny dále redukovat o nevhodné koleje např. z důvodu nedostačující užitečné délky koleje. Na obrázku 1 je znázorněno ilustrační kolejiště železniční stanice. Množina přípustných kolejí je např. $K_{S1,S1}=\{4, 2, 1, 5, 7\}$, $K_{S4,S1}=\{7, 5, 1, 2\}$ nebo $K_{S5,S5}=\{7, 9, 11\}$.



Obr. 1 ***Příklad uspořádání kolejiště železniční stanice.*** **Zdroj: [1]**

[1] RZG – rozkaz o zavedení GVD (grafikon vlakové dopravy).
[2] ČD D4 – předpis ČD pro organizování vlakové dopravy.
[3] KJŘ – knižní jízdní řád.

2.3 Kritéria výběru koleje

Pro zachování maximální realističnosti v přidělování kolejí byla navržena čtyři kritéria výběru koleje, která byla odvozena ze znalosti práce řídících pracovníků. Jedná se o tato kritéria:

A: volnost koleje v okamžiku příjezdu vlaku,

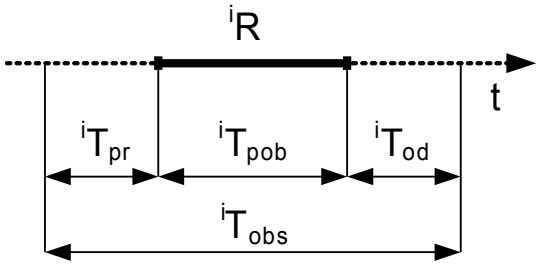
B: doba volnosti koleje vzhledem k době pobytu přijíždějícího vlaku ve stanici,

C: obsazení sousední koleje u stejného nástupiště přípojným vlakem,

D: preference koleje pro přijíždějící vlak. [1]

2.3.1 Volnost koleje v okamžiku příjezdu vlaku

Kritérium hodnotící volnost koleje by mělo logicky nabývat pouze dvou hodnot a to kolej je volná nebo je kolej obsazena. Takto postavené kritérium by ale nedokázalo činit rozdíly mezi kolejemi, které jsou v daném čase obsazené a budou obsazené na dlouhou dobu a kolejemi, které jsou v daném čase obsazené, ale v poměrně krátkém čase už by mohly být využity přijíždějícím vlakem. Z tohoto důvodu je vhodné toto kritérium rozšířit o faktor času se stanoveným výhledem do budoucna. Obrázek 2 ilustruje dobu obsazení koleje jedním vlakem. [1]



Obr. 2 ***Celková doba obsazení koleje jedním vlakem.*** **Zdroj: [1]**

^{i}R vlak, který obsazuje uvažovanou kolej

$^{i}T_{pob}$ doba pobytu vlaku ^{i}R na koleji

$^{i}T_{pr}$ doba, kterou je nutné uvažovat pro příjezd vlaku ^{i}R na kolej

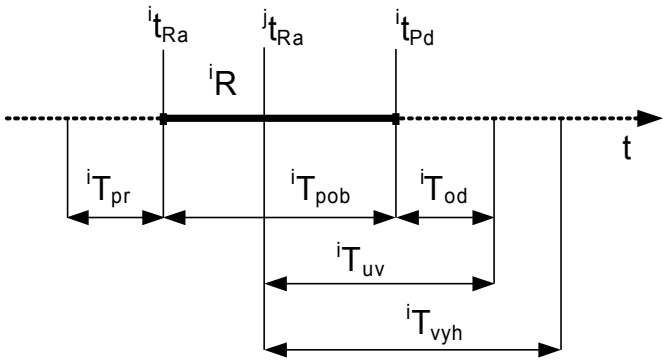
$^{i}T_{od}$ doba, kterou je nutné uvažovat pro odjezd vlaku ^{i}R z koleje

$^{i}T_{obs}$ celková doba obsazení koleje vlakem ^{i}R

Pokud zpožděný vlak ^{j}R přijíždí k uvažované koleji mimo interval $^{i}T_{obs}$, kolej je volná a hodnotu kritéria A volíme rovnu jedné ($A = 1$).

Kritérium rozšířené o faktor času počítá s parametrem T_{vyh}, což je informace o době výhledu do budoucna, kterou lze využít pro ohodnocení koleje v případě, kdy zpožděný vlak přijíždí v čase, kdy je kolej obsazena. Mohou nastat dva případy:

1) Během doby výhledu dojde k uvolnění koleje (tato situace je na obr. 3). Hodnotu kritéria *A* vypočteme podle vztahu (1).
2) Během doby výhledu nedojde k uvolnění koleje, potom podle vztahu (1) platí, že hodnota kritéria je rovna nule ($A = 0$). [1]



Obr. 3 ***Příjezd zpožděného vlaku ^{j}R v době obsazení uvažované koleje vlakem ^{i}R.*** **Zdroj: [1]**

$^{i}t_{Ra}$ čas skutečného příjezdu vlaku ^{i}R jedoucího na čas

$^{j}t_{Ra}$ čas skutečného příjezdu zpožděného vlaku ^{j}R

$^{i}t_{Pd}$ čas plánovaného odjezdu vlaku ^{i}R

$^{i}T_{uv}$ doba zbývající do uvolnění koleje vlakem ^{i}R

T_{vyh} doba, která je uvažována jako maximální pro výhled vzhledem k možné změně stavu obsazení koleje

Hodnota kritéria *A* se vyjádří vztahem:

$$A = max\left\{0{,}1 - \frac{^{i}T_{uv}}{T_{vyh}}\right\}, \qquad (1)$$

kde $^{i}T_{uv}$ je doba zbývající do uvolnění koleje vlakem ^{i}R v okamžiku příjezdu zpožděného vlaku ^{j}R. Tuto dobu lze vypočítat dle:

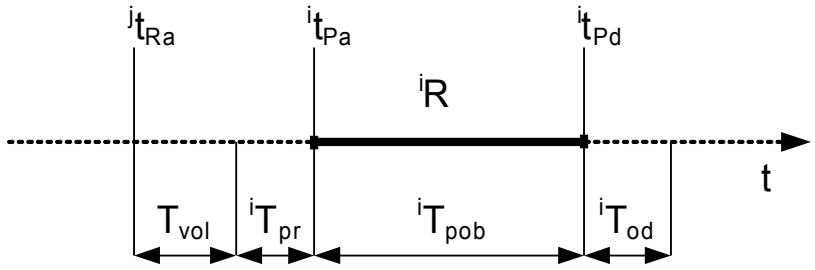
$$^{i}T_{uv} = {}^{i}t_{Pd} - {}^{j}t_{Ra} + {}^{i}T_{od} \qquad (2)$$

2.3.2Doba volnosti koleje vzhledem k době pobytu vlaku ve stanici

Druhým kritériem je doba volnosti koleje vzhledem k době pobytu přijíždějícího zpožděného vlaku ve stanici. Stanovení hodnoty kritéria *B* se vztahuje k plánovanému času pobytu vlaku ^{j}R na koleji. V první fázi stanovení hodnoty kritéria je počítáno s konstantní (plánovanou) dobou pobytu vlaku ve stanici, přičemž v další fázi výzkumu

je možné s tímto kritériem dále pracovat v tom smyslu, že je možné u vybraných vlakových spojů uvažovat o kratší než plánované době pobytu vlaku ve stanici. Tato poznámka se týká zejména vlaků, které mají ve stanicích delší plánovanou dobu pobytu a zkrácením doby pobytu by nevznikaly komplikace při vykonávání obsluhy podle stanovených postupů technologických procesů. Stejně jako u kritéria *A* bude i u kritéria *B* hodnota z intervalu $\langle 0,1 \rangle$.

Výpočet probíhá na základě vyhodnocení doby, po kterou je uvažovaná kolej volná do příjezdu dalšího vlaku. Schéma k výpočtu pro příjezd zpožděného vlaku v čase, kdy je uvažovaná kolej volná, je znázorněno na obrázku 4. [1]



Obr. 4 ***Stanovení kritéria B v případě, kdy je kolej v době příjezdu zpožděného vlaku volná.*** **Zdroj:[1]**

${}^{i}t_{Pa}$ čas plánovaného příjezdu vlaku ${}^{i}R$

T_{vol} doba, po kterou je uvažovaná kolej volná

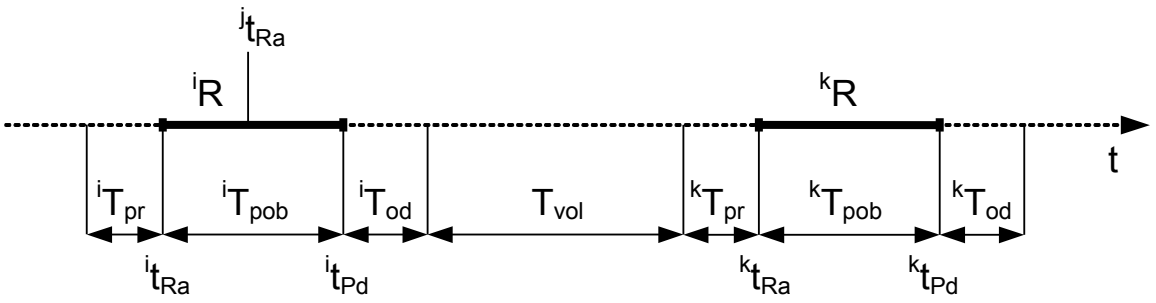
${}^{j}t_{Ra}$ čas skutečného příjezdu zpožděného vlaku ${}^{j}R$

Kritérium *B* se stanoví podle vztahu:

$$B = min\left\{1, \frac{T_{vol}}{{}^{j}T_{obs}}\right\}, \qquad (3)$$

kde ${}^{j}T_{obs}$ je plánovaná doba obsazení dané koleje vlakem ${}^{j}R$.

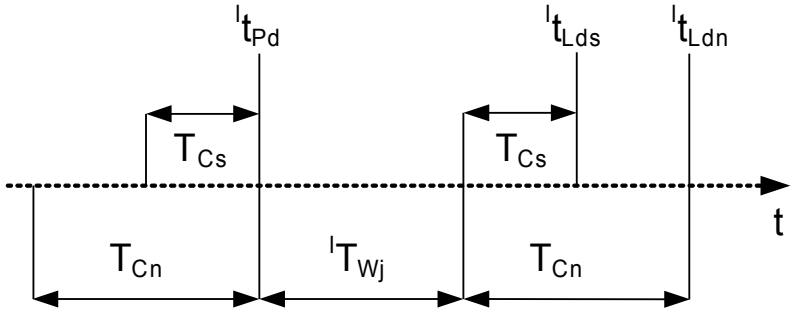
Pokud vlak ${}^{j}R$ přijede v době, kdy je uvažovaná kolej obsazena, je postup výpočtů podobný, s tím že časový interval (o délce T_{vol}), po který je kolej volná, započne až po uvolnění této koleje vlakem ${}^{i}R$. Tento případ znázorňuje obrázek 5. [1]



Obr. 5 ***Stanovení kritéria B v případě, kdy je kolej v době příjezdu zpožděného vlaku obsazena.*** **Zdroj: [1]**

2.3.3Obsazení koleje u téhož nástupiště přípojným vlakem

Kritérium *C* bylo zavedeno, protože je vhodné přijíždějící zpožděný vlak ${}^{j}R$ umístit ke stejnému nástupišti, kde čeká přípojný vlak ${}^{l}R$. Lze tak předejít vzniku nebo nárůstu zpoždění vlaku ${}^{l}R$, tím že se využije zkrácená doba potřebná na přestup. Toto kritérium nabývá pouze dvou hodnot {0,1}. Výpočet ilustruje obrázek 6.



Obr. 6 ***Určení časového intervalu pro určení hodnoty kritéria C.*** **Zdroj: [1]**

${}^{l}t_{Pd}$ plánovaný odjezd vlaku ${}^{l}R$

T_{Cn} normální doba potřebná na přestup

T_{Cs} zkrácená doba potřebná na přestup

${}^{l}T_{Wj}$ čekací doba vlaku ${}^{l}R$ na zpožděný přijíždějící vlak ${}^{j}R$

${}^{l}t_{Lds}$ nejzazší přípustný čas odjezdu vlaku ${}^{l}R$ při uplatnění zkrácené doby potřebné na přestup

${}^{l}t_{Ldn}$ nejzazší přípustný čas odjezdu vlaku ${}^{l}R$ při uplatnění normální doby potřebné na přestup

Z obrázku je patrné, že je výhodné pro příjezd zpožděného vlaku ${}^{j}R$ v časovém intervalu $\left\langle {}^{l}t_{Pd} - T_{Cn},\ {}^{l}t_{Pd} + {}^{l}T_{Wj} \right\rangle$ tento přijíždějící vlak umístit ke stejnému nástupišti, u něhož na něj čeká přípojný odjíždějící vlak z důvodu úspory času na přestup cestujících (mezi přijíždějícím zpožděným vlakem ${}^{j}R$ a vlakem přípojným ${}^{l}R$).

Hodnotu kritéria *C* pro koleje sousedící s kolejí, ze které odjíždí přípojný vlak se určí takto [1]:

$$C = 1 \quad \text{pro } {}^{i}t_{Ra} \in \left\langle {}^{l}t_{Pd} - T_{Cn},\ {}^{l}t_{Pd} + {}^{l}T_{Wj} \right\rangle \tag{4}$$

$$C = 0 \quad \text{pro } {}^{i}t_{Ra} \notin \left\langle {}^{l}t_{Pd} - T_{Cn},\ {}^{l}t_{Pd} + {}^{l}T_{Wj} \right\rangle \tag{5}$$

2.3.4Preference koleje pro přijíždějící vlak

Posledním kritériem jsou preference koleje pro přijíždějící vlak, které reflektují další technické a technologické přednosti přidělení dané koleje uvažovanému zpožděnému přijíždějícímu vlaku ${}^{j}R$. Označuje se jako kritérium *D* a nabývá hodnoty z intervalu $\langle 0,1 \rangle$. Kolej pravidelně určená přijíždějícímu vlaku může mít ohodnocení např. *D = 1*, koleje nevhodné pro přijíždějící vlak potom *D = 0*. Ostatní koleje nabývají ohodnocení z intervalu $\langle 0,1 \rangle$, přičemž je možné měnit hodnotu tohoto kritéria pro různé hodnoty zpoždění vlaku ${}^{j}R$. Hodnotu kritéria *D* je stanovíme na základě znalostí provozu zkoumané stanice (např. s využitím konzultací s provozními zaměstnanci). [1]

2.4 Sestavení kriteriální matice

Poté co jsou stanoveny množiny přípustných kolejí a vyčíslena všechna kritéria výběru koleje sestaví se z těchto hodnot tzv. kriteriální matice ve tvaru ilustrovaném na obrázku 7. Tato matice je vždy vztažená k určitému vlaku, délce jeho zpoždění a množině jeho přípustných kolejí.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} k_1 & k_2 & \dots & k_m \end{array} \\ \begin{pmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ y_{31} & y_{32} & \cdots & y_{3m} \\ y_{41} & y_{42} & \cdots & y_{4m} \end{pmatrix} \end{array}$$

Obr. 7 ***Kriteriální matice.*** **Zdroj: [1]**

k_n uvažovaná kolej

A–D kritérium výběru koleje

y_{nm} vypočtená hodnota příslušného kritéria

Aby bylo možné z kriteriální matice správně vybrat nástupištní kolej, musí se určit váha pro jednotlivá kritéria *A–D*, čímž dojde k vyjádření relativní důležitosti těchto kritérií.

3 Metoda stanovení vah kritérií

Způsobů jak stanovit váhu jednotlivých kritérií je několik např.:

- matematické metody vícekriteriálního hodnocení variant (viz [9]),
- metody využívající fuzzy logiku,
- umělé neuronové sítě.

Cílem této práce je využít k řešení umělou neuronovou síť. Použití umělé neuronové sítě má své opodstatnění v případech, kdy při řešení dané úlohy není možné matematicky popsat všechny relace a souvislosti, které ovlivňují sledovaný proces, anebo v případech, kdy je matematický model velice komplikovaný a algoritmizace problému by byla téměř nemožná.[11]

3.1 Umělé neuronové sítě

Síla neuronových sítí je hlavně ve struktuře, kterou tvoří velký počet jednoduchých výkonných elementů - neuronů. Toto uspořádání má velkou flexibilitu a spolehlivost. Umožňuje různě vytvářet a modifikovat spoje mezi vstupy a výstupy neuronů a tak zvýhodnit či potlačit některé vstupy.

Struktura neuronů je většinou vrstevnatá, neurony jsou sdružovány do vrstev. Vrstevnaté neuronové sítě lze rozčlenit jednak podle způsobů přenosu signálu mezi jejich vstupy a výstupy na:

- *sítě s dopředným šířením (feed-forward networks)*, u kterých se signál šíří z výstupu n vrstvy na vstup $(n + 1)$. vrstvy a
- *sítě zpětnovazební (feed-back networks)*, u kterých se signál šíří také z výstupu n vrstvy na vstup $(n – i)$. vrstvy pro $i = 1, 2, \ldots, n$ a jednak podle počtu vrstev a podle počtu výkonných prvků v jednotlivých vrstvách [4].

Podle polohy vrstvy v neuronové síti rozeznáváme:

- vrstvu vstupní, v ní umístěné neurony zajišťují vstup signálu z okolí,
- vrstvu nebo vrstvy skryté,
- vrstvu výstupní, jejíž neurony předávají signál do okolí.

Stejně se nazývají i neurony příslušející ke konkrétní vrstvě. Právě počet vrstev a počet neuronů ve vrstvách tvoří konfiguraci sítě.[5]

Vstupní vrstva je tvořena tzv. zdrojovými uzly (vstupními terminály) a slouží ke vstupu určitého signálu z okolí a jeho následnému rozdělení do neuronů následující

vrstvy. Lze ji chápat jako pasivní prvky s jednotkovým přenosem, které nemají vlastnosti klasického neuronu. Proto v praxi není do celkového počtu vrstev sítě tato vrstva započítávána. Potom vstupní vrstva přechází ve vektor vstupů a číslování vrstev začíná od první skryté vrstvy až po vrstvu výstupní. Poslední vrstvou uvažované umělé neuronové sítě je výstupní vrstva. Tato vrstva je určena k přenosu výstupních signálů z neuronové sítě do okolí. Tyto výstupní signály jsou pak odezvou neuronové sítě na signály vstupní. Všechny případné mezilehlé vrstvy se označují jako vrstvy skryté. Úkolem skrytých vrstev je zvýšení aproximačních vlastností neuronové sítě jako celku. [5]

Základem funkce každého paradigmatu neuronových sítí je adaptační etapa činnosti, které se říká proces učení. V průběhu tohoto procesu jsou nastavovány vhodné volné, resp. nastavitelné, parametry příslušné umělé neuronové sítě tak, aby odchylka (měřená ve vhodné metrice) mezi jejím požadovaným a skutečným výstupem při odezvě na soubor učebních (trénovacích) vzorů byla minimální.

Učební procesy je možno rozdělit do celé řady skupin podle těchto hlavních hledisek:

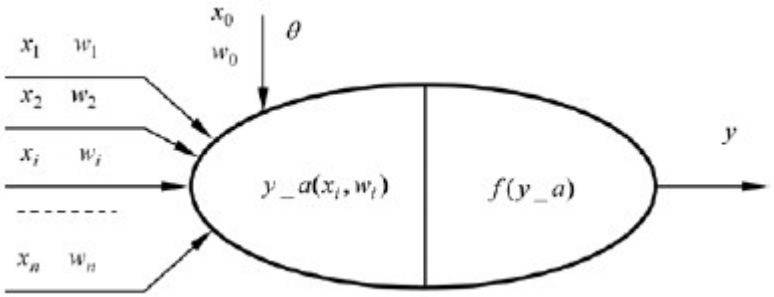
- podle toho, zda v procesu učení jsou adaptovány jen zvolené nastavitelné parametry či též struktura umělé neuronové sítě,
- podle výběru nastavitelných parametrů,
- podle toho, zda jsou pro učení použity vzory vstupních, případně též výstupních signálů (učení s učitelem) či zda jde o učení samo se organizující (bez učitele),
- podle výběru posloupnosti vstupních, resp. výstupních signálů při učení s učitelem,
- podle charakteru a matematické povahy použitého učebního algoritmu,
- podle výkonnosti, rychlosti, numerické účinnosti a přesnosti použitého učebního postupu,
- podle druhu struktury, na nějž lze daný učební proces aplikovat.[4]

Schopnosti naučit neuronovou síť rozhodovací mechanismy lze využít při určování váhy jednotlivých kritérií a to aplikací správného trénovacího vzoru, vhodně zvoleným učebním procesem a typem neuronové sítě. Dále je nutné věnovat pozornost tomu, jaký je rozměr (dimensionalita) dané sítě a posléze tomu, jaká je míra redundance již naučené sítě.

3.1.1 Neuron

V umělém neuronu dochází k náhradě funkcí biologického neuronu funkcemi matematickými. Obecně se můžeme setkat s velkým počtem různých typů modelů umělých neuronů, které se navzájem liší vlastní topologií a typy matematických funkcí, které jejich chování popisují. Dosud v praxi nejpoužívanějším modelem je tzv. formální (často označován jako základní) model neuronu. Tento typ neuronu bývá také často nazýván podle svých autorů McCulloch-Pittsův neuron.[14]

Základní schéma formálního neuronu je uvedeno na obrázku 8. Každý umělý neuron obsahuje konečný počet vstupů x_n a jediný výstup y. Tento jediný výstup je samozřejmě možno rozmnožit do potřebného počtu kopií – vstupů do následných neuronů. V každém neuronu se vstupní hodnoty transformují na výstup pomocí minimálně dvou výpočetních procedur. Konkrétně se jedná o výpočet vstupního potenciálu y_a a o tzv. aktivační funkci f.



Obr. 8 ***Schéma umělého neuronu.*** **Zdroj: [14]**

x_i vstupy neuronu (výstupy z předcházející vrstvy), i = 1, 2, ..., n

n počet vstupů (počet neuronů v předcházející vrstvě)

w_j synaptické váhy

y_a vstupní potenciál neuronu

f aktivační funkce neuronu

θ práh neuronu

y výstup neuronu

3.2 Zvolený typ neuronové sítě

Každá neuronová síť jako celek realizuje určitou, pro ni typickou transformační funkci – převádí (transformuje) hodnoty vstupních veličin na hodnoty veličin výstupních. V tomto případě nám UNS[4] poslouží jako univerzální funkční aproximátor. To znamená, že navržená a naučená UNS by měla být následně schopná činnost výchozího procesu s určitou mírou přesnosti reprodukovat pro různé vstupní signály.[5]

[4] UNS – umělá neuronová síť

Návrh topologie neuronové sítě je poměrně složitý a bohužel neexistuje jednoznačný návod. Podle [5] však vyplývá, že v řadě případů dosáhneme vyhovující aproximace chování modelované soustavy za použití dvouvrstvé neuronové sítě. Pro výpočet počtu neuronů v této skryté vrstvě se uvádí vztah:

$$p = \sqrt{n\,m} \tag{6}$$

nebo

$$p \geq n + m, \tag{7}$$

kde n je počet vstupů a m je počet výstupních neuronů. Ze zkušeností také vychází, že větší počet skrytých vrstev a velký počet neuronů v těchto vrstvách může vést k tzv. „přeučení“ sítě, a nevede k lepším výsledkům a k zkvalitnění modelu analyzovaného systému.

Ze zadání úlohy plyne, že po neuronové síti budeme požadovat, aby ve vytvořených kriteriálních maticích (vstupy) ohodnotila na základě hodnot jednotlivých kritérií všechny přípustné nástupištní koleje (výstupy). Signál se tedy bude šířit od vstupu k výstupu, a proto použijeme síť s dopředným šířením. Jelikož budeme mít k dispozici také trénovací a testovací množiny je vhodné zvolit síť využívající metodu učení s učitelem (viz kap. 3).

Jedním ze zástupců výše zmíněných dopředných sítí je perceptronová síť. Perceptron sestává z jediného výkonného prvku modelovaného obvykle McCullochovým a Pittsovým modelem neuronu, který má nastavitelné váhové koeficienty a nastavitelný práh přičemž platí Rosenblattova věta:

Máme-li v n-rozměrném prostoru lineárně separabilní třídy objektů, pak lze v konečném počtu kroků učení (iterací optimalizačního algoritmu) nalézt vektor vah W perceptronu, který oddělí jednotlivé třídy bez ohledu na počáteční hodnoty těchto vah.

Perceptron s jedním výkonným prvkem umožňuje ovšem nanejvýše klasifikaci do dvou tříd. Zvětšíme-li však počet výkonných prvků pracujících v perceptronu a zvětšíme-li i počet jeho vrstev, je možno jím klasifikovat do více tříd. Tyto třídy již nemusí být lineárně separabilní, musí však být separabilní. Jednotlivé položky v kriteriální matici jsou lineárně nezávislé, takže tato podmínka bude splněna.[4]

3.3 Výstavba neuronové sítě

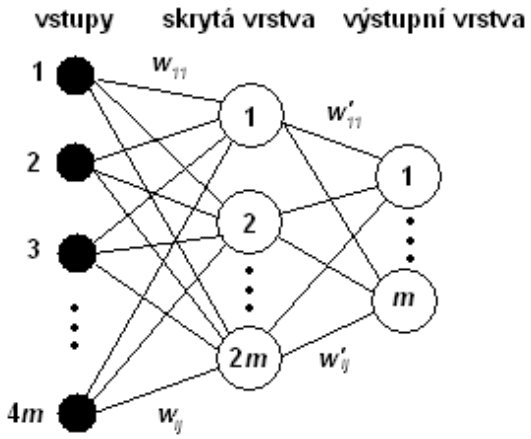
Z výše uvedeného plyne, že s největší pravděpodobností bude nejlépe aproximovat chování zadaného systému dvouvrstvá perceptronová síť. Aby síť poskytovala korektní výsledky, je třeba ji vhodně parametrizovat.

3.3.1Počet neuronů ve skryté a výstupní vrstvě

Počet vstupů n bude podmíněn počtu přípustných kolejí v kriteriální matici a počtu kritérií u jednotlivé koleje (u všech kolejí se bude počítat se 4 kritérii *A–D*). Množství výstupních neuronů m bude odpovídat počtu přípustných kolejí v kriteriální matici. Z tohoto vyplývá vztah $n = 4m$. Počet neuronů p ve skryté vrstvě se pak dosazením do (6) určí jako

$$p = \sqrt{4m^2} = 2m \tag{8}$$

Na obrázku 9 je zobrazeno ilustrační schéma navržené neuronové sítě. Koeficient w_{ij} symbolizuje váhu spojení mezi jednotlivými položkami vrstev. Příslušné indexy charakterizují spojení od *i* zdroje k *j* cíli.



Obr. 9 ***Ilustrační schéma použité neuronové sítě***

3.3.2Aktivační funkce

Dále je třeba rozhodnout o tvaru aktivační (přenosové) funkce neuronů ve skryté a výstupní vrstvě. Tato funkce transformuje vstupy na výstupní signál neuronu. Má za úkol rozeznat, zda kombinace úrovně vstupních signálů je dostatečně významná (zda není podprahová, či příliš vysoká), resp. kvantifikovat v určitém rozmezí jejich významnost.[7] Funkce vychází ze vstupně-výstupní rovnice neuronu:

$$y = f(síť) = f\left(\sum_{i=1}^{D} w_i x_i + \theta \right), \tag{9}$$

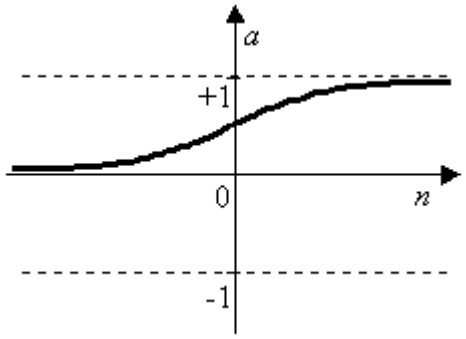
kde D je počet vstupů, x_i jsou vstupy neuronu, w_i jsou váhové hodnoty a θ je prahová hodnota. Aktivační funkce f je pak logická funkce definovaná vztahem:

$$f(síť) = \begin{cases} 1 & \text{pro } síť \geq 0 \\ -1 & \text{pro } síť < 0 \end{cases} \tag{10}$$

Aktivační funkce mohou mít různý charakter průběhu funkce (lineární nebo nelineární). Pro tento případ byla vybrána funkce typu logsigmoid

$$y = \frac{1}{1 + e^{-x}}, \tag{11}$$

jejíž průběh ilustruje obrázek 10. Tato funkce je velmi často využívána v mnohovrstvých perceptronových sítích, protože je diferencovatelná a výstup nabývá hodnot z intervalu $\langle 0,1 \rangle$. [6]



Obr. 10 ***Průběh logsigmoidální aktivační funkce.*** **Zdroj: [6]**

3.3.3Algoritmus učení back-propagation

Důležitým krokem při výstavbě neuronové sítě je volba algoritmu učení (viz kap. 3.1). K naučení navržené sítě byla použita metoda back-propagation (zpětné šíření), která byla vypracována v polovině osmdesátých let jako obecně použitelná metoda učení vícevrstvých umělých neuronových sítí. Je to v podstatě gradientní metoda minimalizující součet čtverců chyby výstupních hodnot uvažované sítě.[4] Změna vektoru parametrů γ neuronové sítě je úměrná gradientu chyby $\Delta E(p_e)$ kriteriální funkce:

$$\Delta \gamma (p_e) = -\alpha \, \Delta E(p_e), \tag{12}$$

kde α je velikost iteračního kroku, kterému v daném případě říkáme koeficient učení UNS a p_e je pořadové číslo epochy trénování. Vektor γ obsahuje váhy spojení *w* a prahové hodnoty aktivačních funkcí. Jednotlivé složky gradientu $\Delta\gamma_j(p_e)$ se pak určí jako parciální derivace kriteriální funkce podle jednotlivých proměnných. Kriteriální funkce se určí dle vztahu:

$$E = \frac{1}{2} \sum_{i=1}^{n} \left(y_{Si} - y_{Mi} \right)^2, \tag{13}$$

kde *n* je počet výstupů sítě, y_{Si} je *i*. výstup a y_{Mi} je požadovaný výstup.[8]

Podle [8] je určení koeficientu učení α zcela experimentální záležitostí. Podobně neexistuje jednotné doporučení pro volbu počátečních hodnot parametrů (vah spojení, prahů, případně strmostí aktivačních funkcí). Většinou je doporučen náhodný výběr

z intervalu $\langle -0{,}5;\, 0{,}5\rangle$, ale i z intervalu $\langle -1;1\rangle$, nebo při dvouvrstvé neuronové sítě zadávat počáteční hodnoty z intervalu $\langle -\gamma\,;\gamma\rangle$, kde

$$\gamma = 0{,}7\sqrt[n]{p}, \tag{14}$$

p je počet neuronů ve skryté vrstvě a *n* počet prvků vstupní vrstvy.

Algoritmus učení metodou zpětného šíření chyby má dvě etapy: *dopředný krok*, spočívající v přenosu signálu sítí, v aktivaci jednotlivých neuronů a ve výpočtu chyby *E* a *zpětný krok*, spojený s výpočtem nových hodnot parametrů sítě γ.[8]

Přírůstek vah bude záporný α násobek hodnoty gradientu, tedy:

$$\Delta w = -\alpha\, \nabla E, \tag{15}$$

Nové přírůstky vah a prahů se počítají postupně od výstupní ke vstupní vrstvě podle vztahů:

$$\Delta w_{ij}^{o}(t) = \alpha\, \delta_i^{o}(t)\, x_j^{h}(t) + \lambda\, \Delta w_{ij}^{o}(t-1), \tag{16}$$

$$\Delta \theta_i^{o}(t) = \alpha\, \delta_i^{o}(t) + \lambda\, \Delta \theta_i^{o}(t-1), \tag{17}$$

x_j^h je výstup *i* neuronu skryté vrstvy a w_{ij}^o je váha spojující *i* neuron výstupní vrstvy s *j* neuronem předcházející vrstvy. Koeficient λ označuje tzv. setrvačnost, která se může zavést pro „přibrzdění“ rychlých změn v případě, že by kriteriální funkce vykazovala lokální minima. Index *o* označuje výstupní vrstvu a *h* skrytou vrstvu. Koeficient δ_i představuje chybu aktuální vrstvy. Chyba předcházející vrstvy se vypočte vynásobením chyby δ_i s hodnotou příslušného váhového koeficientu. Tímto způsobem se přenese hodnota chyby na předchozí skrytou vrstvu. Obecný vztah pro výpočet chyb v celé síti lze zapsat ve tvaru:

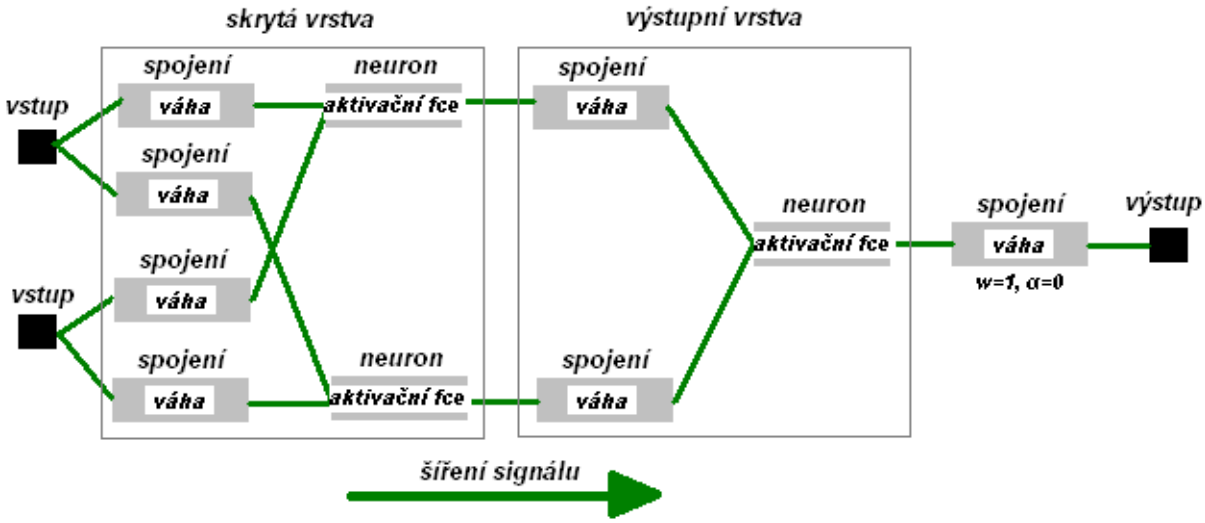
$$\delta_i^{h-1} = x_i^{h-1}\left(1 - x_i^{h-1}\right)\sum_{k=1}^{n} w_{ki}^{h}\delta_k^{h}, \tag{18}$$

Pro výstupní vrstvu lze parametr δ_i vyjádřit jako:

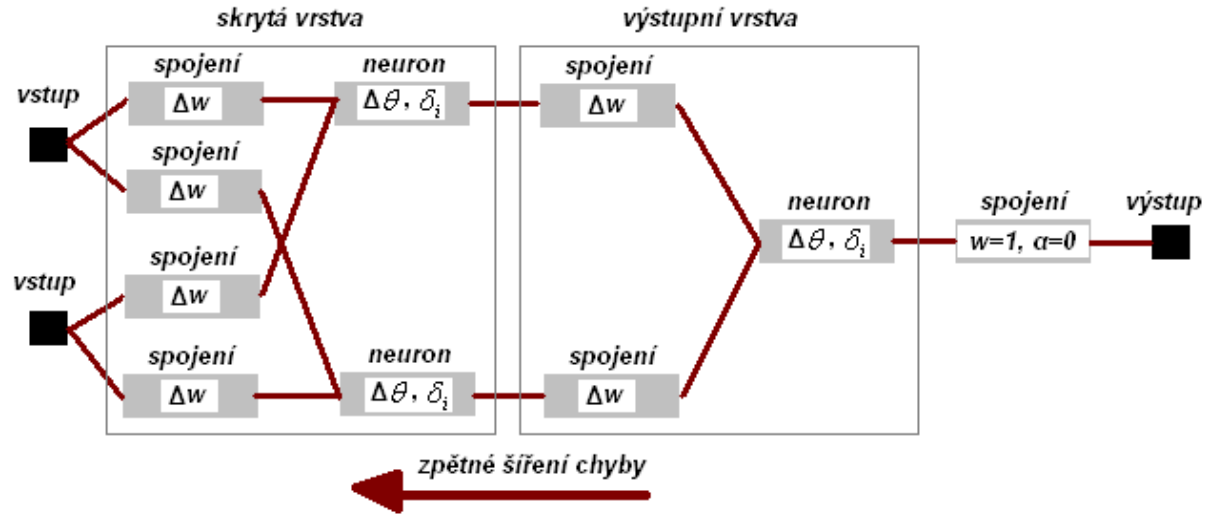
$$\delta_i = y_i(1-y_i)(\hat{y}_i - y_i), \tag{19}$$

kde y_i je skutečný výstup a $\hat{y}_i$ je očekávaný výstup.[4]

Obrázky 11 a 12 ukazují šíření signálu při dopředném kroku resp. šíření chyby ve zpětném kroku.



Obr. 11 ***Dopředný krok alg. back-propagation***



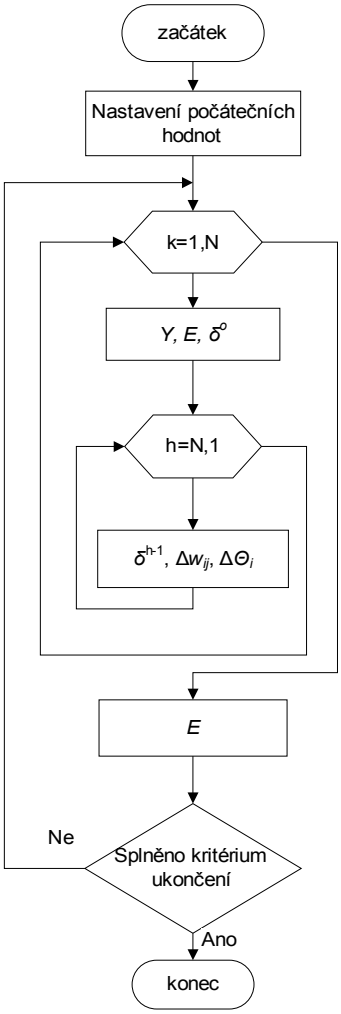
Obr. 12 ***Zpětný krok alg. back-propagation***

Popis algoritmu:

1. Nastavení počátečních hodnot parametrů podle (14).
2. Výběr trénovacího vzoru z trénovací množiny a výpočet výstupů jednotlivých neuronů dle vztahu (9).
3. Vypočtení hodnoty kriteriální funkce (13) a chyby pro výstupní vrstvu δ_i^o z (19).
4. Zpětné šíření chyby o jednu vrstvu blíž vstupům pomocí vztahu (18). Upravení vah a prahů výpočtem jejich přírůstků (16), (17). Tento krok se opakuje pro všechny vrstvy sítě od výstupní ke vstupní.
5. Pokud byly do sítě vloženy všechny vzory z trénovací množiny přechází se na krok 6, jinak se vrací na krok 2.

6. Je-li po poslední iteraci dosažena hodnota kriteriální funkce menší než námi zvolená hodnota, pak se učení ukončí, jinak se pokračuje bodem 2.

Na obrázku 13 je pak vidět znázornění tohoto algoritmu ve vývojovém digramu.



Obr. 13 ***Vývojový diagram alg. back-propagation***

4 Implementace řešení

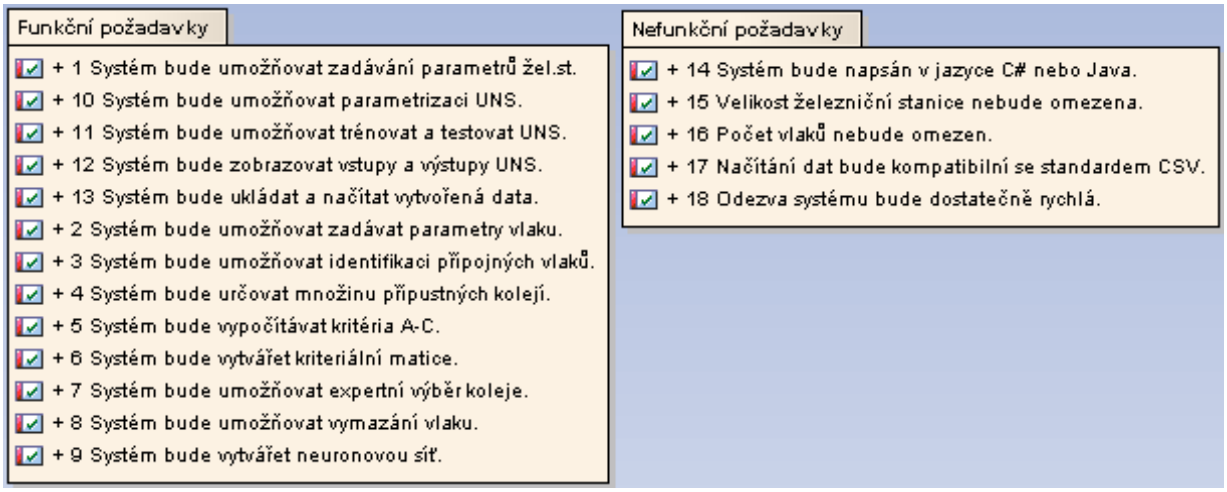
Tato kapitola se bude zabývat vytvořením programu, který by měl realizovat přidělování nástupištních kolejí zpožděným vlakům za využití navržené umělé neuronové sítě. Požadavky, analýza a návrh programu bude provedena pomocí jazyka UML[5] a metodiky Unified Process[6]. Implementace proběhne v prostředí jazyka C# ve vývojovém softwaru Microsoft Visual Studio 2005 .

4.1 Požadavky

Zachycení požadavků na systém probíhá ve dvou modelech.

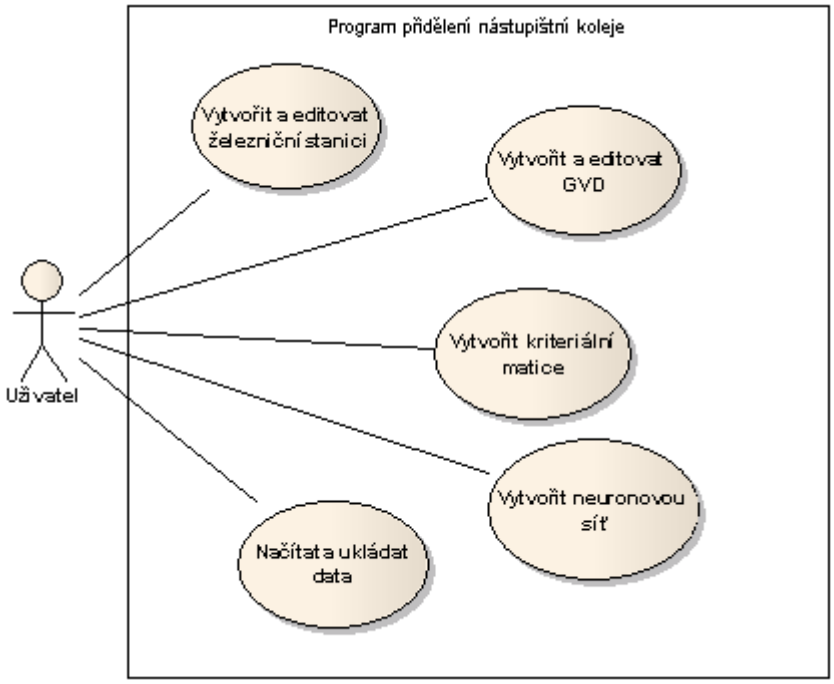
- *Model požadavků* (requirements specification) – obsahuje funkční (chování jaké bude systém nabízet) a nefunkční (vlastnosti nebo omezující podmínky systému) požadavky.
- *Model případů užití* (use case model) – vyjadřuje činnosti systému, aktéry a jejich relace.[10]

Na obrázku 14 je zobrazen model požadavků na navrhovaný systém zachycený v programu Enterprise Architect 4.01. Před každým požadavkem je jeho ID, které bude sloužit k následné kontrole jeho splnění. Na pořadí jednotlivých položek nezáleží.



Obr. 14 ***Model funkčních a nefunkčních požadavků***

[5] UML – Unified Modeling Language (univerzální jazyk pro vizuální modelování systémů) [10]
[6] Unified Process – metodika tvorby softwarového vybavení [10]



Obr. 15 ***Model případů užití***

Na obrázku 15 je znázorněn aktér *Uživatel* a jeho případy užití systému. Ty lze ještě více specifikovat:

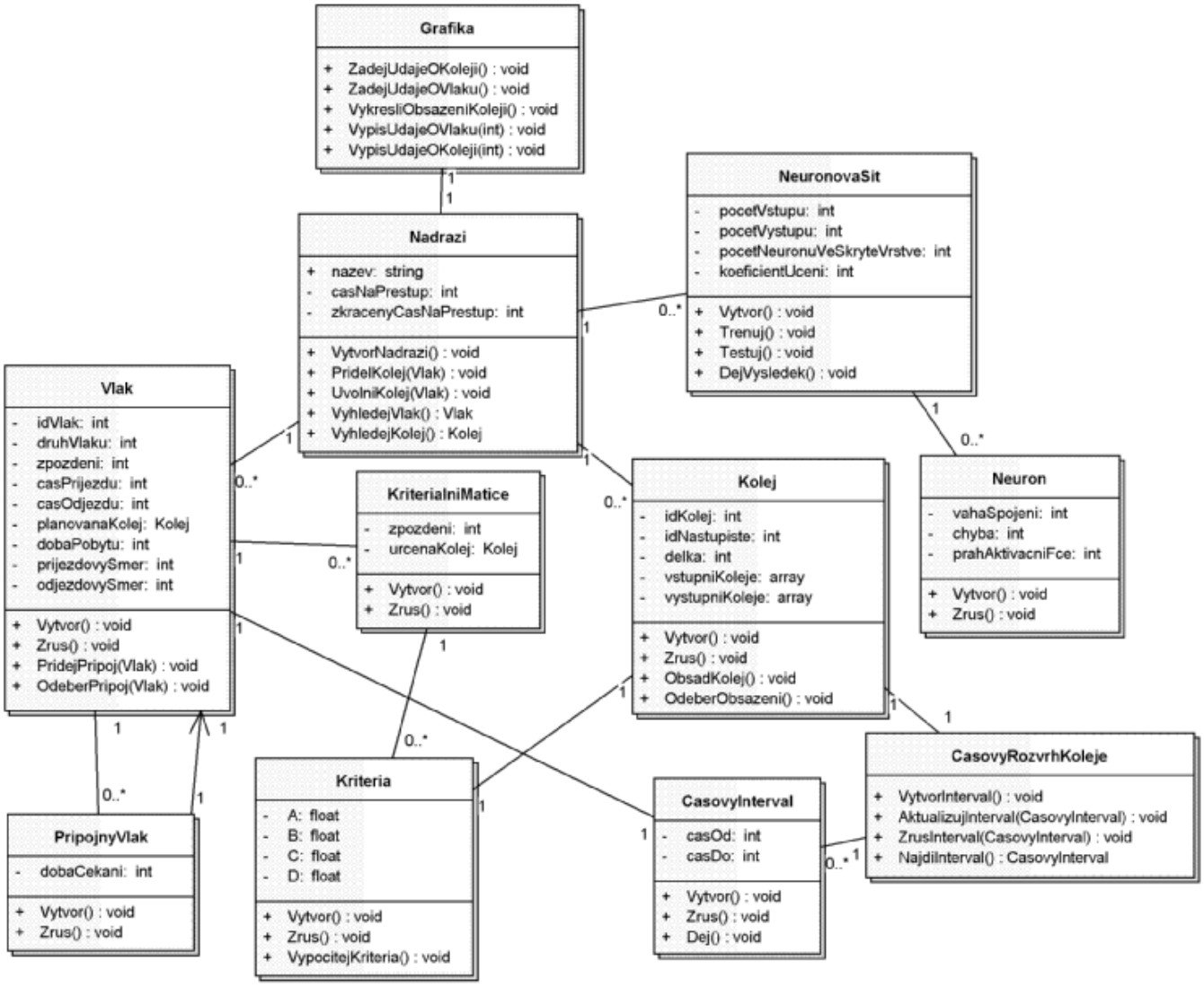
- *Vytvořit a editovat žel. stanici* – vytvořit a editovat nástupištní, vstupní a výstupní kolej, připojit vstupní a výstupní koleje k nástupištním, zadat časy potřebné na přestup.
- *Vytvořit a editovat GVD* – vytvořit a editovat vlak, označit přípojné vlaky.
- *Vytvořit kriteriální matice* – nastavit hodnotu kritéria *D*, nastavit zpoždění vlaku, označit expertem vybranou kolej.
- *Vytvořit neuronovou síť* – parametrizovat UNS, trénovat a testovat UNS.

4.2 Analýza

Analýza spočívá v tvorbě modelů, jež zachycují podstatné požadavky a charakteristické rysy vytvářeného systému. Artefakty analýzy jsou analytické třídy, které tvoří klíčové pojmy, a realizace případů užití, jež názorně ukazují, jak mohou instance analytických tříd vzájemně komunikovat s cílem realizovat chování systému specifikované případem užití.[10]

Model analytických tříd a jejich relací je zobrazen na obrázku 16. Z něj jsou patrné základní metody a atributy tříd. Ústřední třídou je třída *Nadrazi*, která agreguje třídy *Vlak*, *Kolej* a *NeuronovaSit*. Jednotlivé třídy *Vlak* zahrnují třídy *KriterialniMatice*

obsahující *Kriteria* pro koleje. Každá třída *Vlak* může také agregovat třídy *PripojnyVlak*, které ilustrují návaznosti a čekání vlaků. Třída *Kolej* sdružuje třídu C*asovyRozvrhKoleje*, která obsluhuje třídy C*asovyInterval* a slouží pro zachycení obsazení koleje. Stanovení optimální nástupištní koleje z vybraných kriteriálních matic zajišťuje třída *NeuronovaSit*, která spolu s třídami *Neuron* obsahuje metody a atributy potřebné k výstavbě, natrénování a testování UNS. Interakci s uživatelem zajišťuje třída *Grafika*, která využívá pro přístup do systému třídu *Nadrazi*.

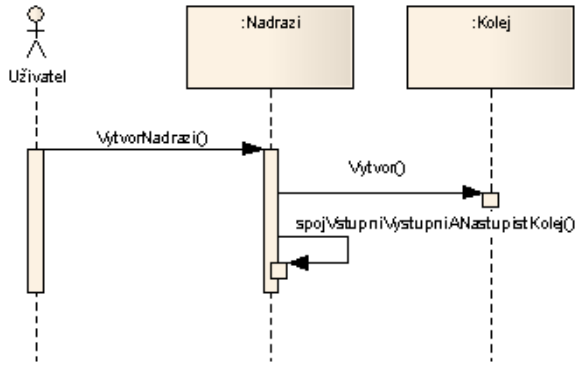


Obr. 16 ***Model analytických tříd***

Model analytických tříd však neposkytuje informaci o interakcích v systému. Tu lze zachytit v realizaci případů užití. Jednou z možností jak vyjádřit realizaci případů užití jsou sekvenční diagramy znázorňující časově orientovanou posloupnost zpráv předávaných mezi objekty. Modelování vztahů v diagramech je důležité pro jasné a

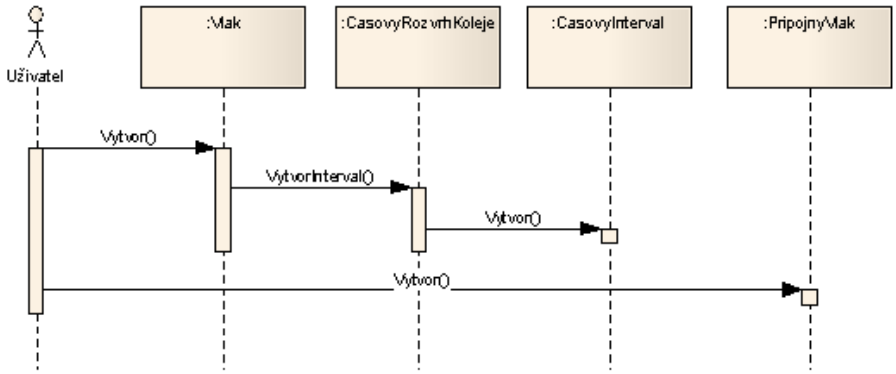
rychlé pochopení činností v systému. Další variantou zobrazení interakcí jsou např. komunikační diagramy, diagramy zjednodušené interakce a diagramy časování.

Na obrázku 17 je vidět proces vytvoření železniční stanice. Obrázek zdůrazňuje nutnost spojení vstupních a výstupních kolejí s nástupištními, které zajišťuje třída *Nadrazi*.



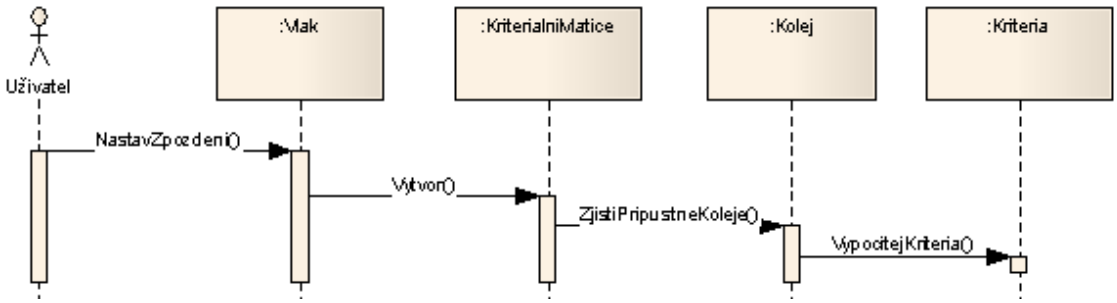
Obr. 17 ***Sekvenční diagram vytvoření železniční stanice***

Případ užití Vytvoření GVD je zachycen na obrázku 18. Zde je ukázána posloupnost činností při vytváření jednotlivých vlaků s následným obsazováním kolejí v podobě tvorby časového rozvrhu koleje.



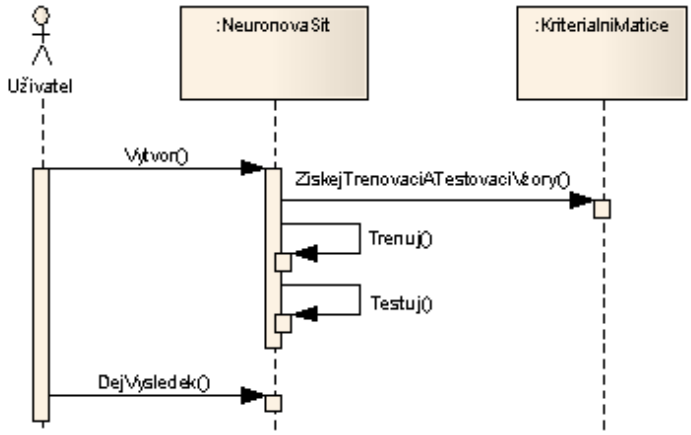
Obr. 18 ***Sekvenční diagram vytvoření GVD***

Předávání zpráv mezi třídami v případu užití Vytvoření kriteriálních matic ilustruje obrázek 19.



Obr. 19 ***Sekvenční diagram vytvoření kriteriální matice***

Proces výstavby neuronové sítě naznačuje obrázek 20, na kterém je pro zjednodušení vidět přímé předání trénovacích a testovacích vzorů z třídy *KriteriálníMatice* do třídy *NeuronovaSit*, i když tyto jsou v analytickém modelu spojeny nepřímo.

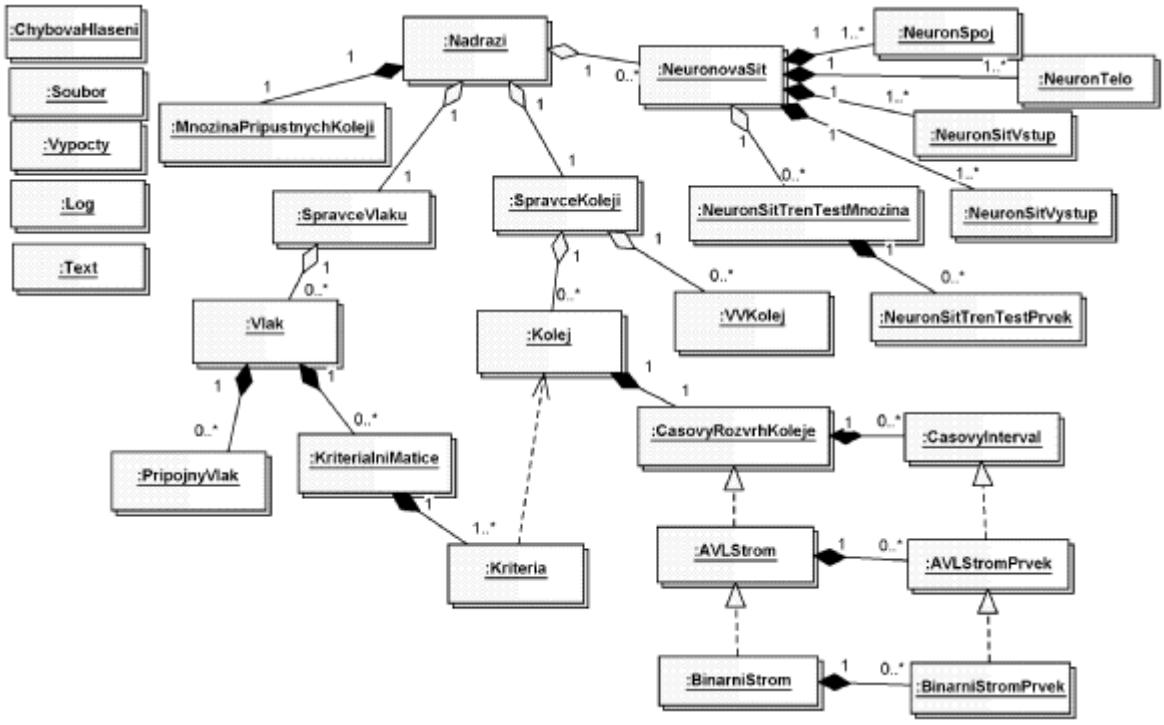


Obr. 20 ***Sekvenční diagram vytvoření neuronové sítě***

4.3 Návrh

Zatímco analýza je zaměřena hlavně na tvorbu logického modelu vyvíjeného systému, který zachycuje funkce, jež tento systém musí poskytovat, aby uspokojil požadavky uživatelů, smyslem návrhu je přesná specifikace způsobu, jak takové funkce implementovat. Návrhový model je založen na modelu analytickém, který upřesňuje a rozpracovává. Návrhový model obsahuje přesně jeden návrhový systém, jenž může obsahovat mnoho návrhových podsystémů.[10]

Pro zjednodušení je návrhový model na obrázku 21 bez grafických tříd. Jelikož na systém nejsou kladeny žádné zvláštní grafické požadavky, budou vstupy a výstupy systému řešeny až v implementační fázi. Na následujícím obrázku také chybí metody jednotlivých návrhových tříd. Ty budou specifikovány v dalších podkapitolách.



Obr. 21 ***Návrhový model tříd***

4.3.1Statické třídy

Z obrázku 21 lze vyčíst spojení a vztahy mezi třídami. Bez přímého spojení na ostatní jsou statické třídy:

- *ChybovaHlaseni* – slouží k obsloužení, výpisu a ošetření chyb.
- *Vypocty* – poskytuje různé výpočty jako např. přepočet. celého čísla na čas a obráceně.
- *Text* – obsahuje metody, které formátují text podle zadaných parametrů.
- *Log* – zapisuje průběh trénování a testování neuronové sítě do souboru.
- *Soubor* – obsahuje metody pro import a export z CSV[7] souboru.

[7] CSV – souborový formát pro tabulková data (Comma-separated values, hodnoty oddělené čárkami)

4.3.2Třída Nadrazi

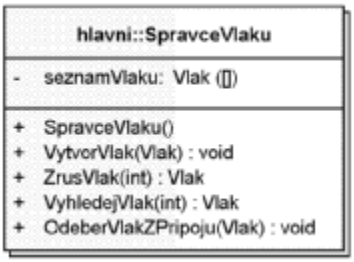
Obr. 22 ***Návrhová třída Nadrazi***

Třída *Nadrazi* je nejdůležitější třídou v programu. Obsahuje metody, které umožňují editaci GVD a kolejiště, a atributy potřebné pro výpočet kritérií (viz kapitola 2.3). Podrobnější popis vybraných metod:

- *vytvorNastupKolej(Kolej)* – volá metodu *vytvorNastupKolej* třídy *SpravceKoleji*, analogicky se chovají metody *vytvorVVKolej* a *Zrus-Kolej*,
- *VytvorVlak(Vlak)* – zavolá metodu *VytvorVlak* třídy *SpravceVlaku* a následně volá metodu *PridelKolej*,
- *ZrusVlak(Vlak)* – zavolá metodu *ZrusVlak* třídy *SpravceVlaku* a následně volá metody *UvolniKolej* a *OdeberVlakZPripoju*,
- *PridelKolej(Vlak)* – volá metodu *ObsadKolej* třídy *SpravceKoleji*,
- *UvolniKolej(Vlak)* – volá metodu *ZrusObsazeniKoleje* třídy *Spravce-Koleji,*
- *VyhledejNastupKolej(string)* – volá metodu *VyhledejNastupKolej* třídy *SpravceKoleji*,
- *DejPoleNastupKoleji* – vrací atribut *seznamNastupKoleji* třídy *Spravce-Koleji* obsahující pole nástupištních kolejí,

- *KontrolaNavaznostiKoleji(VVKolej, Kolej)* – zkontroluje zda vstupně-výstupní kolej navazuje na nástupištní voláním metody *JeVNavazujicich* třídy *VVKolej*,
- *VytvorMnozinyPripustnychKoleji* – do atributu *mnozinyPripustnych-Koleji* ukládá množiny přípustných kolejí (viz kapitola 2.2), příslušná množina je pak identifikována jako množina[kolej1,kolej2], kde jednotlivé její prvky jsou typu *MnozinaPripustnychKoleji*.

4.3.3Třída SpravceVlaku



Obr. 23 ***Návrhová třída SpravceVlaku***

SpravceVlaku má atribut *seznamVlaku*, jenž je datovou strukturou tříd *Vlak* typu pole. Práci s touto strukturou zajišťují metody:

- *VytvorVlak(Vlak)* – vloží zadaný vlak do pole,
- *ZrusVlak(int)* – odebere vlak na zadaném indexu pole,
- *VyhledejVlak(int)* – vyhledá vlak se zadaným id vlaku,
- *OdeberVlakZPripoju(Vlak)* – projde vlaky a odebere jim z přípojů zadaný vlak.

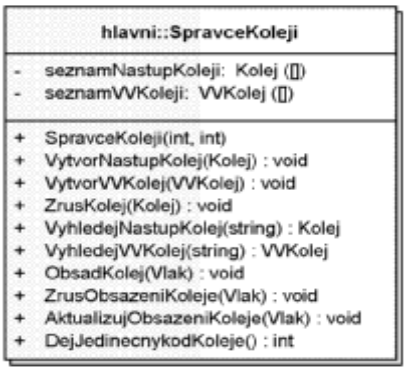
4.3.4Třída Vlak

Obr. 24 ***Návrhová třída Vlak***

Tato třída obsahuje atributy pro uchování všech důležitých informací o vlaku, jako jsou např. čas příjezdu a odjezdu, plánovanou nástupištní kolej, vstupní a výstupní kolej, přípojné vlaky a seznam kriteriálních matic pro jednotlivá zpoždění vlaku. Podrobnější popis vybraných metod:

- *PridejPripoj(PripojnyVlak)* – přidá instanci třídy *PripojnyVlak* do struktury *seznamPripojVlaku*, třída *PripojnyVlak* obsahuje pouze konstruktor, ukazatel na třídu Vlak, která má být přípojem, a dobu čekání. Cílem této konstrukce je umožnit zadání více různých čekacích dob pro jeden vlak.
- *JeVPripojich(Vlak)* – zjistí zda je ukazatel na předanou instanci třídy *Vlak* ve struktuře *seznamPripojnychVlaku*.
- *DejPripustneKoleje(Nadrazi)* – poskytuje pole kolejí obsažené v matici *MnozinaPripustnychKoleji*[*prijezdovaKolej*, *odjezdovaKolej*]

4.3.5Třída SpravceKoleji



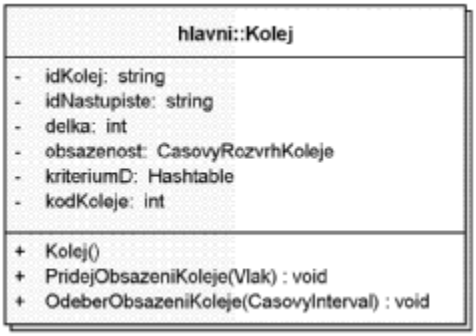
Obr. 25 ***Návrhová třída SpravceKoleji***

Třída *SpravceKoleji* spravuje struktury nástupištních (*seznamNastupKoleji*) a vstupně-výstupních kolejí (*seznamVVKoleji*) typu pole. K tomu jí slouží metody:

- *VytvorNastupKolej(Kolej)* – uloží zadanou kolej do pole *seznamNastup-Koleji*, obdobně *VytvorVVKolej*,
- *ZrusKolej(Kolej)* – odebere z pole zadanou kolej,
- *VyhledejNastupKolej(string)* – projde *seznamNastupKoleji* a vrátí kolej se zadaným id, analogicky pracuje metoda *VyhledejVVKolej*,
- *DejJedinecnyKodKoleje* – vytvoří unikátní kód, jenž má za úkol jednoznačně identifikovat kolej,
- *ObsadKolej(Vlak)* – volá metodu *PridelObsazeniKoleje* třídy *Kolej*,

- *ZrusObsazeni(Vlak)* – volá metodu *OdeberObsazeniKoleje* třídy *Kolej*,
- *AktualizujObsazeniKoleje(Vlak)* – v případě změny obsazení koleje, provede jeho zrušení a následně znovu obsadí kolej daným vlakem z důvodu správného zařazení časového intervalu do struktury časového rozvrhu koleje (viz 4.3.8) .

4.3.6Třída Kolej

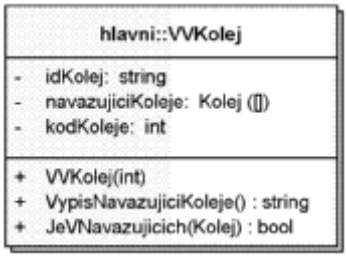


Obr. 26 ***Návrhová třída Kolej***

Třída *Kolej* obsahuje atributy důležité pro uchování informací o nástupištní koleji např. číslo koleje (*idKolej*), číslo nástupiště a délka. Dále udržuje záznam o vytíženosti koleje (*obsazenost*) a hodnotu kritéria *D* (viz kapitola 2.3.4) pro přijíždějící vlak a to v hashovaní tabulce, kde klíčem je číslo vlaku a hodnotou kritérium *D*. Využívá metod:

- *PridejObsazeniKoleje(Vlak)* – voláním metody *VytvorInterval* třídy *CasovyRozvrhKoleje* obsadí kolej daným vlakem,
- *OdeberObsazeniKoleje(Vlak)* – voláním metody *OdeberInterval* třídy *CasovyRozvrhKoleje* zruší obsazení koleje daným vlakem.

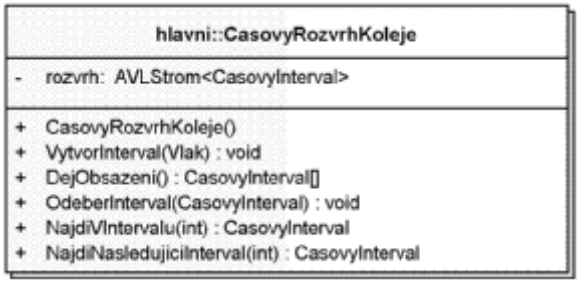
4.3.7Třída VVKolej



Obr. 27 ***Návrhová třída VVKolej***

Vstupně-výstupní koleje jsou uloženy v instancích třídy *VVKolej*. Ta obsahuje pole *navazujiciKoleje* zahrnující ukazatele na třídy *Kolej*, což umožňuje identifikovat spojení mezi kolejemi. Metoda *JeVNavazujicich(Kolej)* zjistí zda je v tomto poli ukazatel na předanou instanci třídy *Kolej*.

4.3.8Třída CasovyRozvrhKoleje

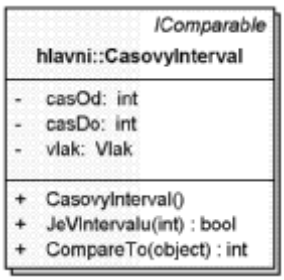


Obr. 28 ***Návrhová třída CasovyRozvrhKoleje***

Třída *CasovyRozvrhKoleje* má na starost udržovat informaci o obsazení koleje. Obsahuje atribut *rozvrh* typu *AVLStrom* (viz 4.3.10), v němž jsou uspořádány instance tříd *CasovyInterval*, kde klíčem je celočíselná hodnota příjezdu vlaku. Podrobnější popis vybraných tříd:

- *VytvorInterval(Vlak)* – vytvoří novou instanci třídy *CasovyInterval* a vloží ji do rozvrhu.
- *DejObsazeni* – vrátí pole tříd *CasovyInterval* získané metodou *Prohlidka* (viz 4.3.12).
- *NajdiVIntervalu(int)* – vrátí *CasovyInterval*, ve kterém se vyskytuje zadaný čas vyjádřený celočíselnou hodnotou.
- *NajdiNasledujiciInterval(int)* – vrátí *CasovyInterval*, který následuje po zadaném čase vyjádřeném celočíselnou hodnotou.

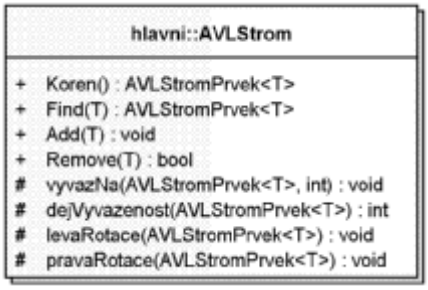
4.3.9Třída CasovyInterval



Obr. 29 ***Návrhová třída CasovyInterval***

Třída *CasovyInterval* vyjadřuje obsazení koleje daným vlakem. Tato třída implementuje rozhraní *IComparable*, a proto musí povinně obsahovat metodu *CompareTo(object)*, která porovnává třídy *CasovyInterval* podle atributu *casOd*. Metoda *JeVIntervalu(int)* zjišťuje zda zadaný čas v celočíselné hodnotě patří do intervalu $\langle casOd, casDo \rangle$.

4.3.10Třída AVLStrom



Obr. 30 ***Návrhová třída AVLStrom***

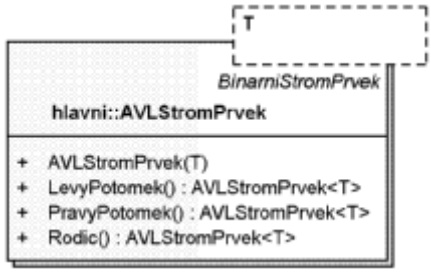
Třída *AVLStrom* implementuje datovou strukturu AVL strom (viz podkapitola 4.3.21) a dědí z třídy *BinarniStrom*. Jelikož její předek implementuje rozhraní *ICollection,* musí povinně obsahovat metody:

- *Add(T)* – přidá prvek *T* do struktury zavoláním metody předka *Add(T)* a je-li potřeba vyváží strom pomocí metody *vyvazNa*,
- *Remove(T)* – odebere prvek *T* ze struktury voláním metody předka *Remove(T)* a je-li potřeba vyváží strom pomocí metody *vyvazNa*,
- *Find(T)* – nalezne prvek *T* zavoláním metody předka *Find(T)*.

Dále poskytuje metody pro manipulaci se zmíněnou strukturou:

- *vyvazNa(AVLStromPrvek<T>, int)* – pomocí metod *levaRotace* a *pravaRotace* vyváží strom.
- *dejVyvazenost(AVLStromPrvek<T>)* – vrací rozdíl výšek podstromů pravého a levého syna zadaného prvku.
- *levaRotace(AVLStromPrvek<T>)* – provede levou rotaci podstromu, kde pivotem je pravý syn zadaného prvku.
- *pravaRotace(AVLStromPrvek<T>)* – provede pravou rotaci podstromu, kde pivotem je levý syn zadaného prvku.

4.3.11Třída AVLStromPrvek

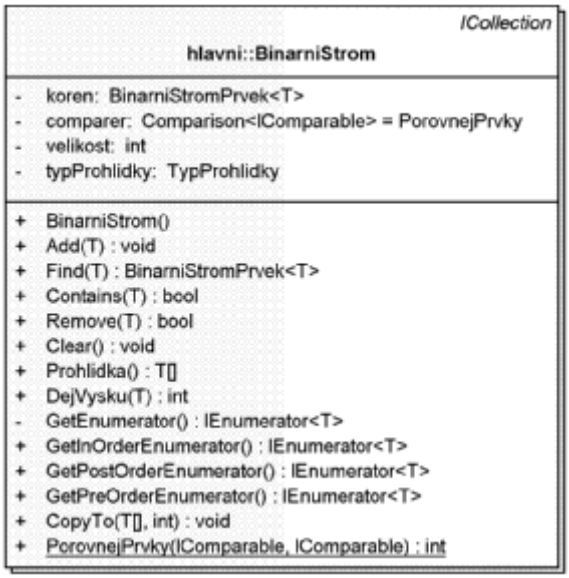


Obr. 31 ***Návrhová třída AVLStromPrvek***

AVLStromPrvek je šablona třídy, což umožňuje do ní později vložit jakoukoliv proměnnou, v tomto případě třídu *CasovyInterval*. Tato třída dědí metody a atributy

z třídy *BinarniStromPrvek* a navíc poskytuje pouze metody pro získání potomků a rodiče.

4.3.12Třída BinarniStrom



Obr. 32 ***Návrhová třída BinarniStrom***

Třída BinarniStrom implementuje datovou strukturu binární vyhledávací strom (viz podkapitola 4.3.21). Atributy jsou:

- *koren* – prvek, který je kořenem stromu,
- *comparer* – poskytuje návratovou hodnotu statické metody *Porovnej-Prvky*,
- *velikost* – počet prvků ve struktuře,
- *typProhlidky* – přednastavený typ prohlídky, která se provede po zavolání metody *Prohlidka*

Třída implementuje rozhraní *ICollection*, proto musí povinně obsahovat tyto metody:

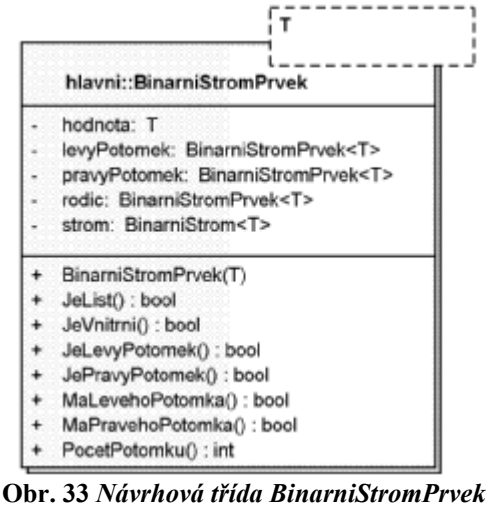
- *Add(T)* – za pomoci atributu *comparer* vloží prvek *T* na správné místo stromu,
- *Remove(T)* – odebere prvek *T* ze stromu a v případě, že prvek není listem, zajistí, aby se zachovala integrita a správné řazení stromu,
- *Find(T)* – za pomoci atributu *comparer* vyhledá ve stromě prvek *T*,
- *Contains(T)* – voláním metody *Find* zjistí zda strom obsahuje prvek *T*,
- *Clear* – provede metodu *Remove* nad všemi prvky stromu,
- *CopyTo(T [], int)* – zkopíruje podstrom prvku se zadaným indexem do pole,

- *GetEnumerator* – vrací objekt enumerátor, který obsahuje referenci na aktuální prvek. Ten se pomocí metody *MoveNext* dokáže posouvat na další prvky. Podle hodnoty atributu *typProhlidky* využívá metod *GetInOrderEnumerator*, *GetPostOrderEnumerator* nebo *GetPreOrderEnumerator*.

Další metody pro práci s touto strukturou:

- *Prohlídka* – projde všechny prvky stromu využitím metody *GetEnumerator*,
- *DejVysku(T)* – vrací výšku podstromu jehož kořenem je prvek T,
- *GetInOrderEnumerator* – vrací enumerátor, jehož metoda *MoveNext* postupuje ve smyslu inorder prohlídky,
- *GetPostOrderEnumerator* – vrací enumerátor, jehož metoda *MoveNext* postupuje ve smyslu postorder prohlídky,
- *GetPreOrderEnumerator* – vrací enumerátor, jehož metoda *MoveNext* postupuje ve smyslu preorder prohlídky,
- *PorovnejPrvky(IComparable, IComparable)* – statická metoda, která vrací kladnou hodnotu je-li hodnota prvního prvku větší než hodnota druhého, zápornou hodnotu je-li to naopak a nulu jsou-li hodnoty prvků stejné. Využívá metodu prvku *CompareTo*.

4.3.13Třída BinarniStromPrvek

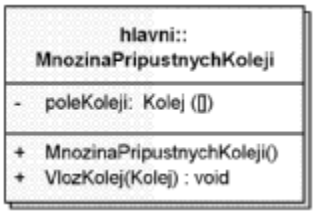


Obr. 33 ***Návrhová třída BinarniStromPrvek***

BinarniStromPrvek je šablona třídy. Obsahuje atribut *hodnota*, který představuje prvek vložený do této šablony. Dalšími atributy jsou levý a pravý potomek prvku, rodič a strom, ve kterém je prvek uložen. Podrobnější popis vybraných metod:

- *JeList* – vrací hodnotu *true* pokud prvek nemá žádné potomky,
- *JeVnitrni* – vrací hodnotu *true* pokud má prvek potomky,
- *JeLevyPotomek* – vrací hodnotu *true* pokud je prvek levým potomkem, analogicky pracuje metoda *JePravyPotomek*,
- *MaLevehoPotomka* – vrací hodnotu *true* pokud má prvek levého potomka, analogicky pracuje metoda *MaPravehoPotomka*,
- *PocetPotomku* – projde podstrom daného prvku a spočítá počet jeho prvků.

4.3.14Třída MnozinaPripustnychKoleji



Obr. 34 ***Návrhová třída MnozinapripustnychKoleji***

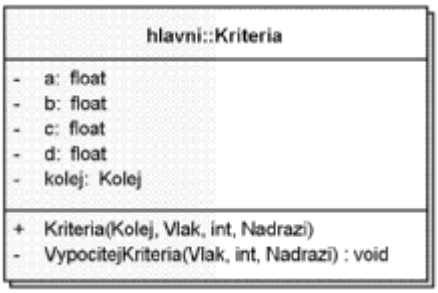
Třída *MnozinaPripustnychKoleji* obsahuje atribut *poleKoleji*, do kterého se ukládají koleje patřící do jedné množiny přípustných kolejí. Metoda *VlozKolej* přidává reference na instance tříd *Kolej* do atributu *poleKoleji*.

4.3.15Třída KriterialniMatice

Obr. 35 ***Návrhová třída KriterialniMatice***

Tato třída obsahuje pole *koleje*, jehož prvky jsou třídy typu *Kriteria*. Dále zahrnuje atributy *vlak* (k němuž je kriteriální matice přiřazena), *zpozdeni* (zpoždění daného vlaku) a *expertUrcKolej* (kolej, kterou určil expert jako optimální). Metoda *NaplnMatici(Nadrazi)* projde přípustné koleje daného vlaku a vloží je do pole.

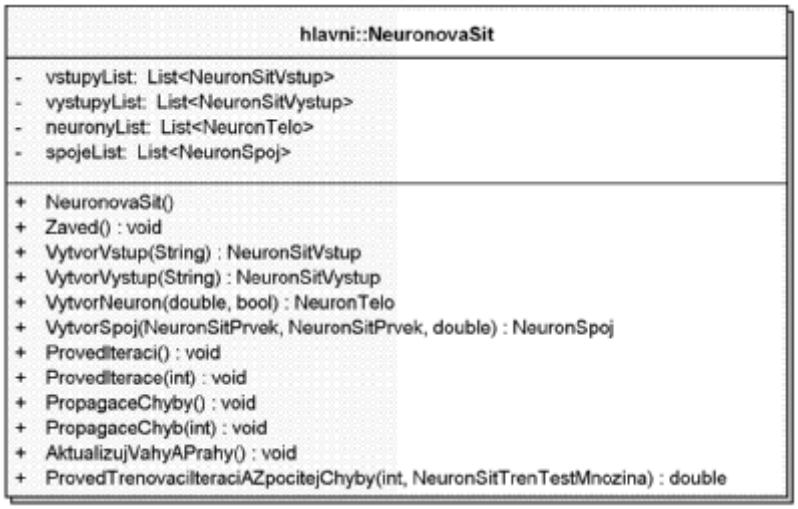
4.3.16Třída Kriteria



Obr. 36 ***Návrhová třída Kriteria***

Třída *Kriteria* má atributy *a*, *b*, *c*, *d*, které představují jednotlivá kritéria podle 2.3 a kolej, ke které jsou kritéria vztažena. Metoda *VypocitejKriteria(Vlak, int, Nadrazi)* vyčíslí hodnotu kritérií *A*–*C* dle vzorců (1)–(5) a hodnotu kritéria *D* převezme z hashovaní tabulky *kriteriumD* třídy *Kolej*.

4.3.17Třída NeuronovaSit



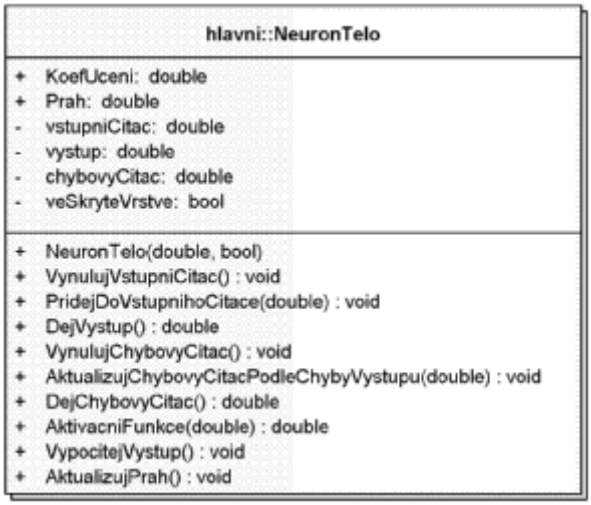
Obr. 37 ***Návrhová třída NeuronovaSit***

Třída *NeurovovaSit* implementuje struktury a metody potřebné k výstavbě UNS. Obsahuje pole vstupů, výstupů, neuronů a spojení. Třídy *NeuronSitVstup* a *NeuronSitVystup* obsahují pouze id a hodnotu vstupu resp. výstupu. Podrobnější popis vybraných metod:

- *ProvedIteraci* – pokud jsou naplněny všechny vstupy, přiřadí každému spoji vstupní hodnotu, pak vypočítá výstupy ze spojů, tyto hodnoty zapíše do vstupních čítačů neuronů a vyčíslí výstupy z neuronů,
- *ProvedIterace(int)* – volá nkrát metodu *ProvedIteraci*,
- *PropagaceChyby* – každému spoji přiřadí chybu, pak v každém neuronu vynuluje chybový čítač, vypočítá váženou chybu a chybový čítač aktualizuje,

- *PropagaceChyb(int)* – volá nkrát metodu *PropagaceChyby*,
- *AktualizujVahyAPrahy* – pro každý spoj volá metodu *AktualizujVahu* a pro každý neuron metodu *AktualizujPrah*,
- *ProvedTrenovaciIteraciAZpocitejChyby(int, NeuronSitTrenTestMnozina)* – nastaví hodnoty vstupních prvků, pak zavolá metodu *ProvedIterace*, poté nastaví hodnoty chyb výstupů, zavolá metodu *PropagaceChyb* a *AktualizujVahyAPrahy* a nakonec vypočítá střední kvadratickou chybu trénování.

4.3.18Třída NeuronTelo



Obr. 38 ***Návrhová třída NeuronTelo***

Třída *NeuronTelo* reprezentující neuron v UNS obsahuje tyto atributy:

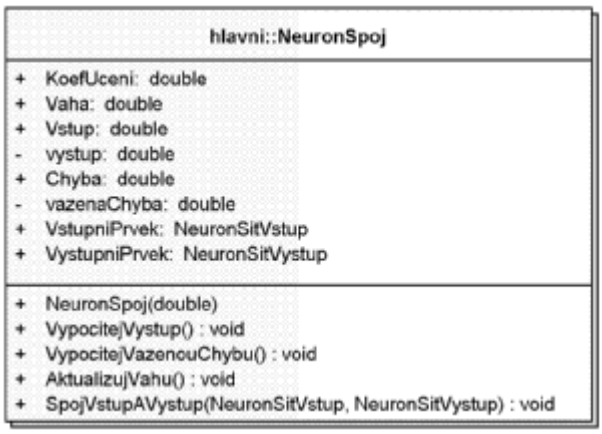
- *KoefUceni* – koeficient učení neuronové sítě α,
- *Prah* – práh aktivační funkce neuronu θ ,
- *vstupniCitac* – představuje kumulovanou hodnotu z výstupů spojů, které jsou připojeny na vstup neuronu,
- *vystup* – číselná hodnota představující výstupní hodnotu z neuronu,
- *chybovyCitac* – je číselná hodnota představující nahromaděnou chybu danou výstupem neuronu,
- *veSkryteVrstve* – určuje zda je neuron ve skryté nebo výstupní vrstvě.

Podrobnější popis vybraných metod:

- *AktualizujChybovyCitacPodleChybyVystupu(double)* – upraví hodnotu atributu *chybovyCitac* vypočtením chyby dle vztahu (18) resp. (19),
- *AktivacniFunkce(double)* – spočítá hodnotu funkce dle vztahu (11),

- *VypocitejVystup* – vypočítá výstupní hodnotu neuronu použitím vzorce (9),
- *AktualizujPrah* – aktualizuje hodnotu prahu aktivační funkce přičtením jeho přírůstku, který se vyčíslí dle (17).

4.3.19Třída NeuronSpoj



Obr. 39 ***Návrhová třída NeuronSpoj***

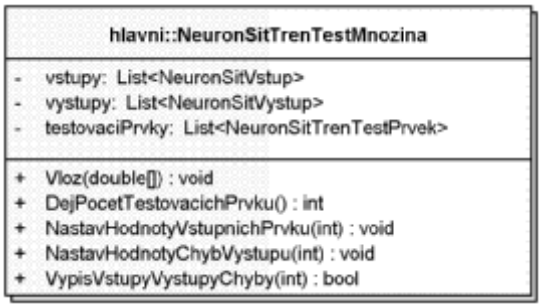
Třída *NeuronSpoj* představuje spojení mezi prvky neuronové sítě patrné z obrázků 11 a 12. Atributy této třídy:

- *KoefUceni* – koeficient učení neuronové sítě α,
- *Vaha* – váha spojení *w*,
- *Vstup* – číselná hodnota přicházející na vstup spojení,
- *Vystup* – číselná hodnota odcházející z výstupu spojení,
- *VstupniPrvek* – prvek neuronové sítě, který je na vstupu spojení,
- *VystupniPrvek* – prvek neuronové sítě, který je na výstupu spojení,
- *Chyba* – číselná hodnota chyby, která přijde z výstupu při zpětném šíření,
- *VazenaChyba* – chyba násobená váhou spojení.

Podrobnější popis vybraných metod:

- *VypocitejVystup* – spočítá hodnotu výstupu tak, že vynásobí váhu spojení a hodnotu vstupu,
- *AktualizujVahu* – upraví hodnotu váhy spojení přičtením jejího přírůstku vyjádřeného vztahem (16),
- *SpojVstupAVystup(NeuronSitVstup, NeuronSitVystup)* – přiřadí zadaný vstup a výstup atributům *VstupniPrvek* a *VystupniPrvek*.

4.3.20Třída NeuronSitTrenTestMnozina



Obr. 40 ***Návrhová třída NeuronSitTrenTestMnozina***

Třída *NeuronSitTrenTestMnozina* slouží pro uložení trénovacích nebo testovacích vzorů. Obsahuje atributy:

- *Vstupy* – pole vstupů neuronové sítě,
- *Vystupy* – pole výstupů neuronové sítě,
- *testovaciPrvky* – trénovací nebo testovací vzory (vstupy a výstupy).

Metody pro práci s třídou:

- *Vloz(double[])* – vytvoří trénovací nebo testovací vzory,
- *DejPocetTestovacichPrvku* – počet trénovacích nebo testovacích vzorů,
- *NastavHodnotyVstupnichPrvku* – odešle hodnoty vstupů z trénovacího nebo testovacího vzoru na vstupy neuronové sítě,
- *NastavHodnotyChybVystupu(int)* – nastaví chybu výstupním prvkům neuronové sítě tak, že od hodnoty na výstupu neuronové sítě odečte očekávanou hodnotu výstupu z trénovacího nebo testovacího vzoru v atributu *testovaciPrvky*.

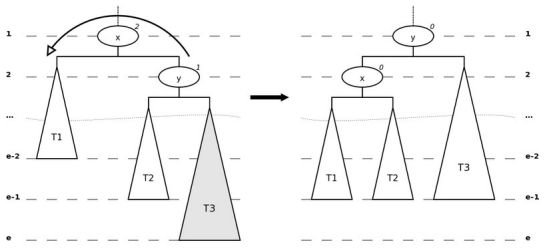
4.3.21Datové struktury

Aby navrhovaný systém dokázal efektivně pracovat s vkládanými daty, je nutné instance navržených tříd uspořádat do vhodných datových struktur. Většina použitých datových struktur je typu pole (seznamy vlaků, kolejí, množin přípustných kolejí, kritérií, vstupů a výstupů neuronové sítě, neuronů, spojů, …). Pole bylo zvoleno, protože ve strukturách uvedených objektů není příliš častá operace hledání nebo je možno přímo vybrat prvek pomocí znalosti jeho pozice anebo se při práci struktura prochází vždy celá. Toto však neplatí u časového rozvrhu kolejí, kde operace hledání převažuje. Zvolená datová struktura je popsána v následující podkapitole.

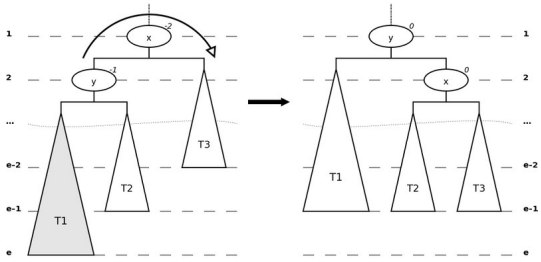
AVL strom

Strom je datová struktura, která uchovává prvky hierarchicky ve vztahu rodič-potomek. AVL strom je výškově vyvážený binární vyhledávací strom (viz 4.3.21), pro který platí, že pro libovolný vnitřní uzel stromu je rozdíl výšek levého a pravého potomka nejvýše 1. Výška je délka nejdelší cesty z kořene stromu k jeho listu. Vyváženost stromu se kontroluje po každé operaci Vlož nebo Odeber. Není-li poté splněno kritérium vyváženosti, musí se provést vyvažování stromu pomocí rotací. Ty jsou realizovány cyklickými záměnami ukazatelů. Rozlišujeme jednoduché pravé nebo levé rotace a dvojité pravo-levé nebo levo-pravé rotace (viz obrázky 40-43).

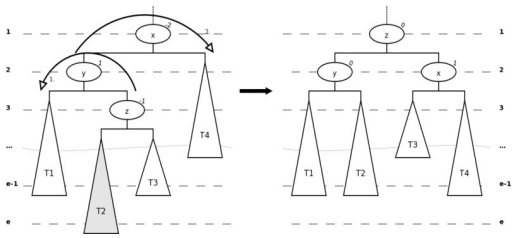
Vyváženost hraje podstatnou roli pro rychlost operací nad stromem. Operace Najdi, Vlož a Odeber mají v AVL stromu složitost $O = (\log_2 n)$, kde *n* je počet vrcholů stromu.[12]



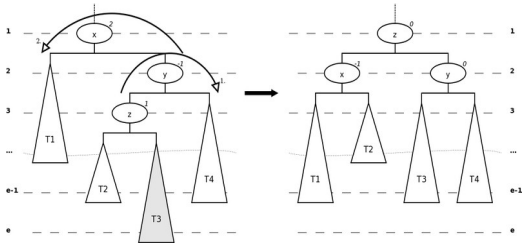
Obr. 41 ***Levá rotace AVL stromu.*** **Zdroj: [13]**



Obr. 42 ***Pravá rotace AVL stromu.*** **Zdroj: [13]**



Obr. 43 ***Levo-pravá rotace stromu.*** **Zdroj: [13]**



Obr. 44 ***Pravo-levá rotace stromu.*** **Zdroj: [13]**

Binární vyhledávací strom

Binární vyhledávací strom je označení pro binární strom, pro jehož každý vrchol x, charakterizovaný klíčem K^x platí:

- je-li y vrchol z levého podstromu vrcholu x, pak $K^y < K^x$,
- je-li y vrchol z pravého podstromu vrcholu x, pak $K^y > K^x$.

Operace nad stromem *Vlož* a *Najdi* opakovaně porovnávají hodnoty klíčů s aktuálním klíčem a výběrem vhodné větve (pravé nebo levé) „traverzují“ stromem ke správné pozici. Při operaci *Vlož* je nalezeno místo pro vkládaný prvek, který je do BVS[8] vložen

[8] BVS – binární vyhledávací strom

jako list. Při operaci *Odeber* je odebíraný prvek nahrazen nejpravějším prvkem z jeho levého podstromu („inorder-předchůdce“) nebo nejlevějším prvkem z jeho pravého podstromu („inorder-následník“) pokud existuje. [12]

Procházení stromu se realizuje buď do šířky (po vrstvách úrovní), nebo do hloubky (začíná se v kořeni stromu a postupuje se vždy na potomky daného vrcholu). Podle pořadí, ve kterém se procházejí uzly do hloubky, se rozlišují tři metody:

- *Preorder* – proveď akci, projdi levý podstrom, projdi pravý podstrom,
- *Inorder* – projdi levý podstrom, proveď akci, projdi pravý podstrom,
- *Postorder* – projdi levý podstrom, projdi pravý podstrom, proveď akci.

Při použití metody *Inorder* se prochází uzly podle jejich přirozeného pořadí.

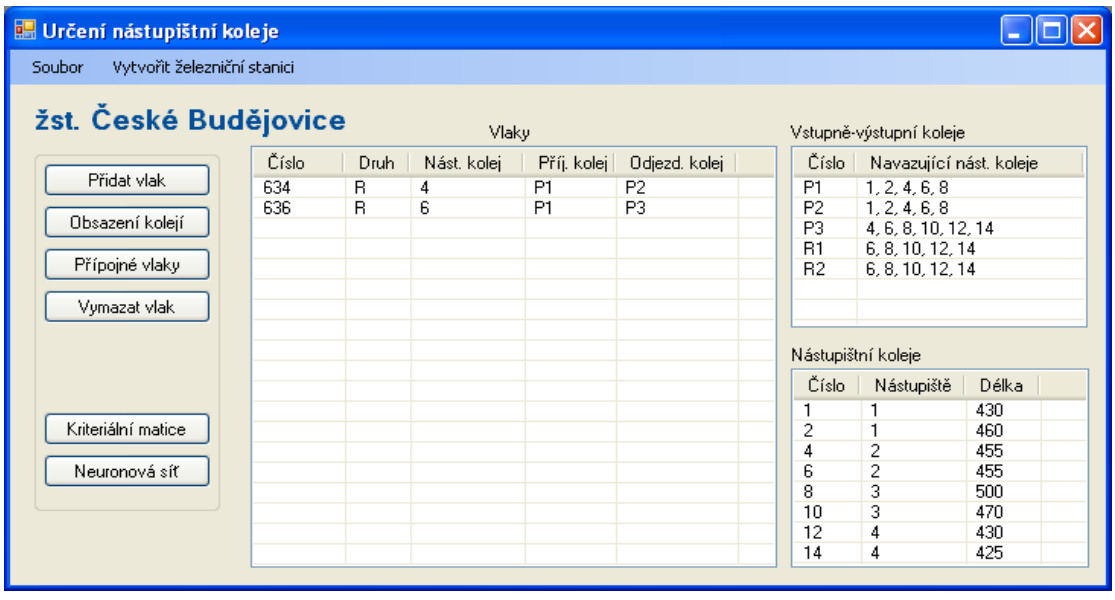
4.4 Implementace

Proces implementace spočívá v převodu návrhového modelu do spustitelného kódu a popsání postupu instalace a nasazení softwaru. Na konci této fáze také dochází ke kontrole splnění funkčních a nefunkčních požadavků na systém.

Jak již bylo zmíněno implementace proběhne v prostředí jazyka C# (splnění požadavku 14). Nyní budou představeny implementace jednotlivých tříd s důrazem na výslednou podobu a chování programu.

4.4.1 Hlavní okno aplikace

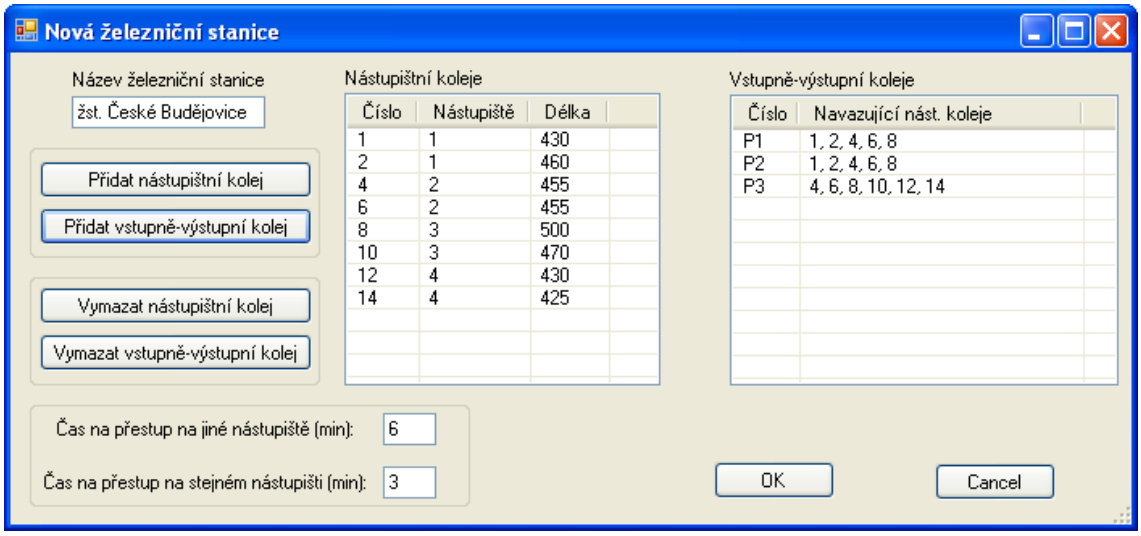
V hlavním okně aplikace se zobrazuje vytvořená železniční stanice (obrázek 45). Jsou zde vidět přijíždějící a odjíždějící vlaky, vstupně-výstupní koleje a nástupištní koleje. Vlevo je hlavní menu s tlačítky pro editaci GVD a vytvoření kriteriálních matic a neuronové sítě. Obsloužení okna má na starosti třída *MainForm*. (plní požadavek 8)



Obr. 45 ***Hlavní okno aplikace***

4.4.2Vytvoření nové železniční stanice

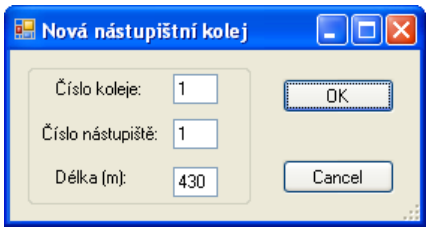
Po stisknutí tlačítka *Vytvořit železniční stanici* v hlavním okně aplikace se zobrazí formulář pro zadávání parametrů nové železniční stanice (obrázek 46). Lze zde přidávat i mazat nástupištní a vstupně-výstupní koleje a je nutné zadat povinné položky: název stanice a doby na přestup mezi nástupišti. Formulář je obsluhován třídou *NoveNadraziDialog*. (ID požadavku: 1, 15)



Obr. 46 ***Vytvoření nové železniční stanice***

Nástupištní kolej

Pro vytvoření nástupištní koleje slouží formulář zobrazený na obrázku 47. Po vyplnění všech parametrů a potvrzení tlačítkem *OK* se vytvoří nová instance třídy *Kolej* a volá se metoda *VytvorNastupKolej(kolej)* třídy *Nadrazi*.



Obr. 47 ***Nová nástupištní kolej***

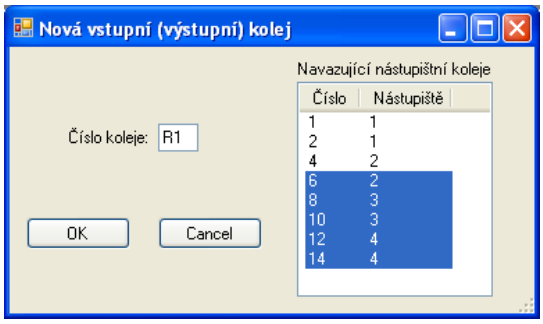
Vstupně-výstupní kolej

Pro vytvoření vstupně-výstupní koleje slouží formulář zobrazený na obrázku 48. Uživatel zde kromě čísla koleje zadává i navazující nástupištní koleje tak, že je vybere ze seznamu nástupištních kolejí.

Ukázka kódu (uložení navazujících kolejí a vytvoření vstupně-výstupní koleje):

```
navazujiciKoleje = new Kolej[navazujiciKolejeListView.SelectedIndices.Count];
Kolej[] poleNastupKoleji = nadrazi.DejPoleNastupKoleji();
for (int i = 0; i < navazujiciKoleje.Length; i++)
{
  navazujiciKoleje[i] =
```

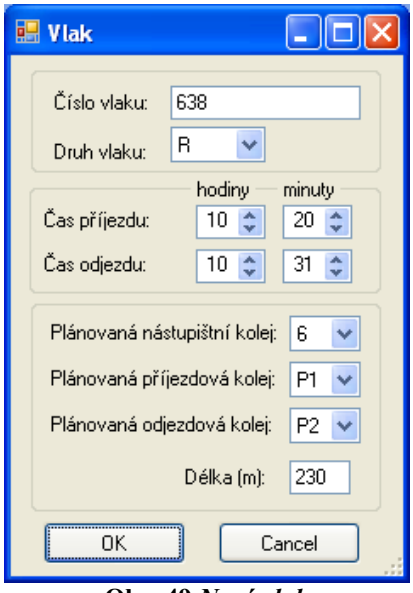
```
                poleNastupKoleji[navazujiciKolejeListView.SelectedIndices[i]];
}

...
kolej = new VVKolej(navazujiciKoleje.Length);
kolej.IdKolej = cisloKoleje;
kolej.NavazujiciKoleje = navazujiciKoleje;
nadrazi.VytvorVVKolej(kolej);
```



Obr. 48 ***Nová vstupně-výstupní kolej***

4.4.3Vytvoření a editace vlaku

Vytvoření a editaci vlaku umožňuje formulář na obrázku 49 obsluhovaný třídou *NovyVlakDialog*. Při vytváření vlaku se formulář vyvolá stisknutím tlačítka *Přidat vlak* v hlavním okně aplikace, editační mód se vyvolá dvojklikem na příslušný vlak v seznamu vlaků. V editačním módu nelze měnit číslo vlaku. (ID požadavku: 2, 16)



Obr. 49 ***Nový vlak***

Před uložením vlaku se kromě správně zadaných parametrů kontroluje i návaznost nástupištní koleje na vstupní (příjezdovou) a výstupní (odjezdovou) kolej. V ukázce implementace je vidět i použití statické třídy *ChybovaHlaseni*.

```
if ( !
nadrazi.KontrolaNavaznostiKoleji((VVKolej)prijezdovaKolejComboBox.SelectedItem,
(Kolej)planovanaKolejComboBox.SelectedItem))
{
   ChybovaHlaseni.ChybaVZadavani("Příjezdová kolej nemá návaznost na
```

```
nástupištní!");
 return;
}
```

Metoda *KontrolaNavaznosti* je implementována takto:

```
public bool KontrolaNavaznostiKoleji(VVKolej vvKolej, Kolej nastupKolej)
{
  if (vvKolej.JeVNavazujicich(nastupKolej)) return true;
  return false;
}
```

Poté se třída *Nadrazi* postará o uložení vlaku a obsazení příslušné nástupištní koleje tímto vlakem. Samotné obsazení koleje provede třída *CasovyRozvrhKoleje* tím, že vytvoří instanci třídy *CasovyInterval* a uloží ji na správné místo v AVL stromu.

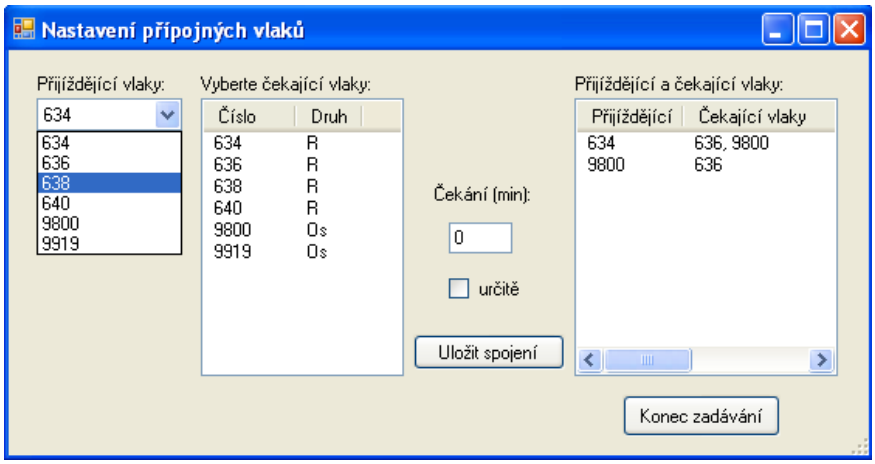
```
//třída Nadrazi
public void VytvorVlak(Vlak vlak)
{
   spravceVlaku.VytvorVlak(vlak);
   PridelKolej(vlak);
}

public void PridelKolej(Vlak vlak)
{
   spravceKoleji.ObsadKolej(vlak);
}

//třída SpravceKoleji
public void ObsadKolej(Vlak vlak)
{
  Kolej obsazovanaKolej = VyhledejNastupKolej(vlak.PlanovanaKolej.IdKolej);
  obsazovanaKolej.PridejObsazeniKoleje(vlak);
}

//třída Kolej
public void PridejObsazeniKoleje(Vlak vlak)
{
  this.obsazenost.VytvorInterval(vlak);
}
//třída CasovyRozvrhKoleje
public void VytvorInterval(Vlak vlak)
{
  CasovyInterval interval = new CasovyInterval();
  interval.CasOd = vlak.CasPrijezdu;
  interval.CasDo = vlak.CasOdjezdu;
  interval.Vlak = vlak;
  vlak.ObsazeniKoleje = interval;
  rozvrh.Add(interval);
}
//třída AVLStrom
public override void Add(T hodnota)
{
   AVLStromPrvek<T> prvek = new AVLStromPrvek<T>(hodnota);
   base.Add(prvek);
   AVLStromPrvek<T> rodicovskyPrvek = prvek.Rodic;
   while (rodicovskyPrvek != null)
   {
     int vyvazenost = this.dejVyvazenost(rodicovskyPrvek);
     if (Math.Abs(vyvazenost) == 2) //-2 nebo 2 je nevyvazeny
     {
       this.vyvazNa(rodicovskyPrvek, vyvazenost); //je treba vyvazit
     }
     rodicovskyPrvek = rodicovskyPrvek.Rodic; //pokracuje v prochazeni
   }
}
```

Při přidávání prvku do stromu se volá metoda předka, kterým je třída *BinarniStrom*. V té je metoda *Add* implementována takto:

```
public virtual void Add(BinarniStromPrvek<T> prvek)
{
  if (this.koren == null)
  {
    this.koren = prvek;
    prvek.Strom = this;
    velikost++;
  }
  else
  {
    if (prvek.Rodic == null)
    prvek.Rodic = koren; //zacina u korene

    //Prvek je vlozen vlevo je-li mensi nez rodic
    bool vlozitVlevo = comparer((IComparable)prvek.Hodnota,
                                (IComparable)prvek.Rodic.Hodnota) <= 0;
    if (vlozitVlevo)
    {
      if (prvek.Rodic.LevyPotomek == null) // je volne misto?
      {
        prvek.Rodic.LevyPotomek = prvek;
        velikost++;
        prvek.Strom = this;
      }
      else
      {
        prvek.Rodic = prvek.Rodic.LevyPotomek; //prochazeni leve vetve
        this.Add(prvek); //rekurzivni volani
      }
    }
    else //vlozeni vpravo
    {
      if (prvek.Rodic.PravyPotomek == null)
      {
        prvek.Rodic.PravyPotomek = prvek;
        velikost++;
        prvek.Strom = this;
      }
      else
      {
        prvek.Rodic = prvek.Rodic.PravyPotomek;//prochazeni prave vetve
        this.Add(prvek);//rekurzivni volani
      }
    }
  }
}
```

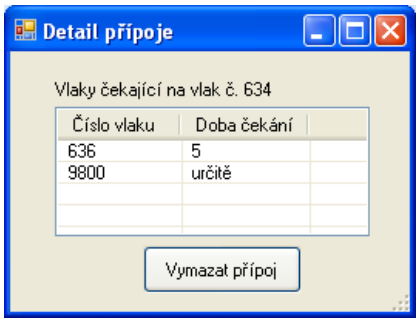
4.4.4Nastavení přípojných vlaků

Přípojné vlaky se nastavují ve formuláři zobrazeném na obrázku 50, který je obsluhován třídou *PripojneVlakyDialog*. Spojení mezi vlaky se vytvoří vybráním přijíždějícího vlaku ze seznamu, označením čekajícího vlaku a zadáním doby čekání. Lze vybrat i více vlaků, pokud čekají stejně dlouho. Přepínač *určitě* slouží k nastavení příznaku, že vlak čeká určitě (stanoveno v [3]). (ID požadavku: 3)



Obr. 50 ***Nastavení přípojných vlaků***

Detail přípojů jednotlivých přijíždějících vlaků lze zobrazit dvojklikem na příslušný řádek v seznamu přijíždějících a čekajících vlaků. V dialogovém okně (obrázek 51), které se pak otevře, je možné přípoje mazat.



Obr. 51 ***Detail přípoje***

4.4.5Výpis obsazenosti kolejí

O zobrazení vytíženosti jednotlivých nástupištních kolejí se stará třída *ObsazeniKolejiDialog*, která obsluhuje formulář uvedený na obrázku 52. Zde je možno výběrem koleje zobrazit časy příjezdů a odjezdů vlaků.

Obr. 52 ***Obsazení kolejí***

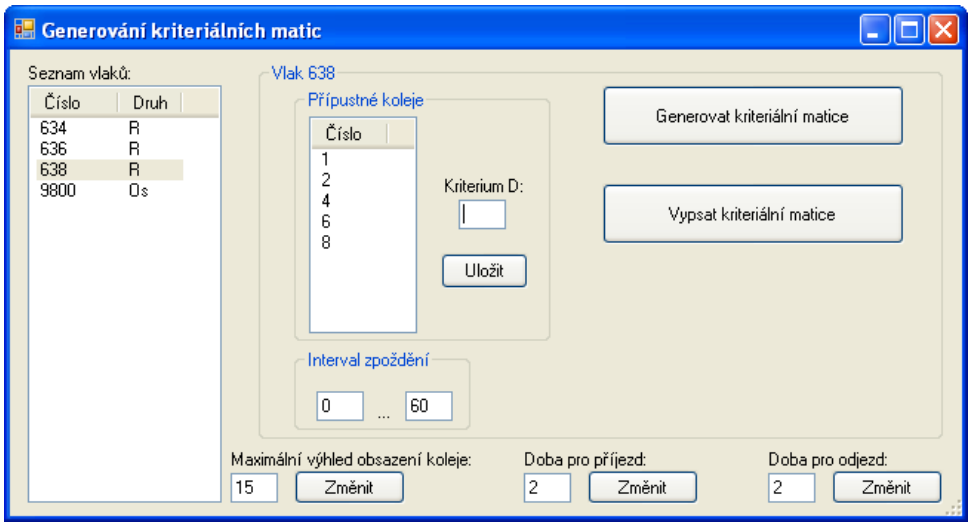
Při vypisování intervalů obsazení koleje se volá inorder prohlídka (viz kapitola 4.3.21) implementovaná v třídě *BinarniStrom*:

```
public virtual T[] Prohlidka()
{
  T[] polePrvku = new T[Count + 1];
  IEnumerator<T> enumerator = GetEnumerator();
  enumerator.Reset();
  int i = 0;
  while (enumerator.MoveNext())
  {
    polePrvku[i] = enumerator.Current;
    i++;
  }
  return polePrvku;
}

public BinarniStromInOrderEnumerator(BinarniStrom<T> strom)
{
  this.strom = strom;
  fronta = new Queue<BinarniStromPrvek<T>>();
  navstivPrvek(this.strom.Koren);
}

private void navstivPrvek(BinarniStromPrvek<T> prvek)
{
  if (prvek == null) return;
  else
  {
    navstivPrvek(prvek.LevyPotomek);
    fronta.Enqueue(prvek);
    navstivPrvek(prvek.PravyPotomek);
  }
}

public bool MoveNext()
{
   if (fronta.Count > 0)
    aktualni = fronta.Dequeue();
   else
    aktualni = null;
   return (aktualni != null);
}
```

4.4.6Generování kriteriálních matic

Z vytvořeného GVD a topologie kolejiště umí program sestavit kriteriální matice pro intervaly zpoždění určených vlaků. Na obrázku 53 je zobrazen formulář umožňující vygenerování těchto matic. Po výběru vlaku se nastaví interval zpoždění (matice se generují pro každou minutu tohoto intervalu). Je také možno určit hodnoty kritéria D pro jednotlivé přípustné nástupištní koleje, program je však dokáže určit i sám na základě očíslování kolejí, to ale nemusí vždy odpovídat reálnému rozmístění ve stanici.

Dále je možno měnit koeficienty T_{vyh} (maximální výhled vzhledem k možné změně stavu obsazení koleje), T_{pr} a T_{od} (doba uvažovaná pro příjezd resp. odjezd vlaku na resp. z koleje). Změněné hodnoty však musí být respektovány v sestaveném GVD, jinak by docházelo k nekorektním výsledkům. (ID požadavku: 4 – 6)



Obr. 53 *Generování kriteriálních matic*

O obsloužení tohoto formuláře se stará třída *GenKritMatDialog*. Po stisku tlačítka *Generovat kriteriální matice* se zavolá následující kód, kde je patrné, že se před samotným výpočtem kritérií metodou *naplnMatici* musí daný vlak odstranit z GVD:

```
int pocetMatic = (zpozdeniDo - zpozdeniOd) + 1;
vybranyVlak.KriterialniMatice = new KriterialniMatice[pocetMatic];
OdeberVlakZObsazeniKoleji();//pred urcenim kriterii, je treba vlak odstranit z
planovaneho obsazeni
for (int i = 0; i < pocetMatic; i++)
{
  vybranyVlak.KriterialniMatice[i] = new KriterialniMatice();
  vybranyVlak.KriterialniMatice[i].Vlak = vybranyVlak;
  vybranyVlak.KriterialniMatice[i].Zpozdeni = zpozdeniOd + i;
  vybranyVlak.KriterialniMatice[i].naplnMatici(nadrazi);
}
VratVlakDoObsazeniKoleji();// po urceni kriterii vrati vlak do obsazeni
ChybovaHlaseni.Hotovo("Kriteriální matice byly vytvořeny!");

// metoda naplnMatici v třídě KriterialniMatice
public void naplnMatici(Nadrazi nadrazi)
{
  Kolej[] pripustneKoleje = vlak.DejPripustneKoleje(nadrazi);
  koleje = new Kriteria[pripustneKoleje.Length];
  for (int i = 0; i < koleje.Length; i++)
  {
    koleje[i] = new Kriteria(pripustneKoleje[i], vlak, zpozdeni, nadrazi);
  }
}
```

Jednotlivá kritéria pak vypočte metoda *VypocitejKriteria*, která se volá v konstruktoru třídy *Kriteria*:

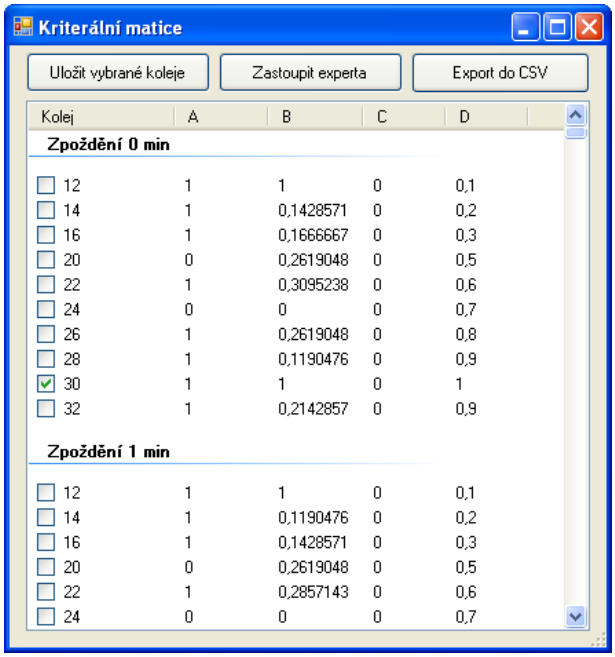
```
private void VypocitejKriteria(Vlak vlak, int zpozdeni, Nadrazi nadrazi)
{
  float Tvyh = nadrazi.TVyh; //maximální výhled
  int Tpr = nadrazi.TPr; //doba příjezdu
  int Tod = nadrazi.TOd; //doba odjezdu
  int jtRa = vlak.CasPrijezdu + zpozdeni;//čas skutečného příjezdu vlaku jR
  CasovyInterval intervalObsazeni = kolej.Obsazenost.NajdiVIntervalu(jtRa);
  //interval, ve kterem je kolej obsazena v dobe jtRa
//kriterium A
```

```
if (intervalObsazeni == null) //v dobe prijezdu je kolej volna
{
  a = 1;
}
else // v dobe prijezdu je kolej obsazena
{
   int itPd = intervalObsazeni.CasDo;
   float iTuv = itPd - jtRa + Tod;
   float x = 1 - (iTuv / Tvyh);
   a = Math.Max(0, x);
}

//kriterium B
float Tvol;//doba, po kterou je uvažovaná kolej volná
float jTobs;//plánovaná doba obsazení dané koleje vlakem jR
if (intervalObsazeni == null) //v dobe prijezdu je kolej volna
{
  int itPa;
  CasovyInterval nasledujiciInterval =
                            kolej.Obsazenost.NajdiNasledujiciInterval(jtRa);
  jTobs = vlak.DobaPobytu + Tpr + Tod;
  itPa = nasledujiciInterval.CasOd;//čas plánovaného příjezdu vlaku iR
  Tvol = itPa - jtRa - Tpr;
  float x = Tvol / jTobs;
  b = Math.Min(1, x);
}
else // v dobe prijezdu je kolej obsazena
{
  int ktPa;// cas prijezdu dalsiho vlaku
  int itPd = intervalObsazeni.CasDo; // cas uvolneni koleje
  CasovyInterval nasledujiciInterval =
                            kolej.Obsazenost.NajdiNasledujiciInterval(itPd);
  jTobs = vlak.DobaPobytu + Tpr + Tod;
  ktPa = nasledujiciInterval.CasOd;
  Tvol = ktPa - Tpr - itPd - Tod;
  float x = Tvol / jTobs;
  b = Math.Min(1, x);
}

//kriterium C
c = 0;
if (vlak.SeznamPripojVlaku != null)
{
  int ltPd; //plánovaný odjezd vlaku lR
  int TCn = nadrazi.CasPrestupJineNastupiste; //normální doba na přestup
  int lTWj; //čekací doba vlaku lR na zpožděný přijíždějící vlak jR
  for (int i = 0; i < vlak.SeznamPripojVlaku.Length; i++)
  {
    if (vlak.SeznamPripojVlaku[i] != null)
    {
      if (vlak.SeznamPripojVlaku[i].Vlak.PlanovanaKolej.IdNastupiste ==
                                                        kolej.IdNastupiste)
      {
        ltPd = vlak.SeznamPripojVlaku[i].Vlak.CasOdjezdu;
        lTWj = vlak.SeznamPripojVlaku[i].DobaCekani;
        if (lTWj == -1) lTWj = nadrazi.CekaUrcite; // vlak ma priznak urcite
        if (jtRa >= (ltPd - TCn) && jtRa <= (ltPd + lTWj))
        {
          c = 1; break;
        }
      }
    }
  }
}

//kriterium D
if (kolej.KriteriumD[vlak.IdVlak] != null)
{
```

```
  d = (float)kolej.KriteriumD[vlak.IdVlak];
}
else { d = 0; }
```

Expertní určení optimální koleje

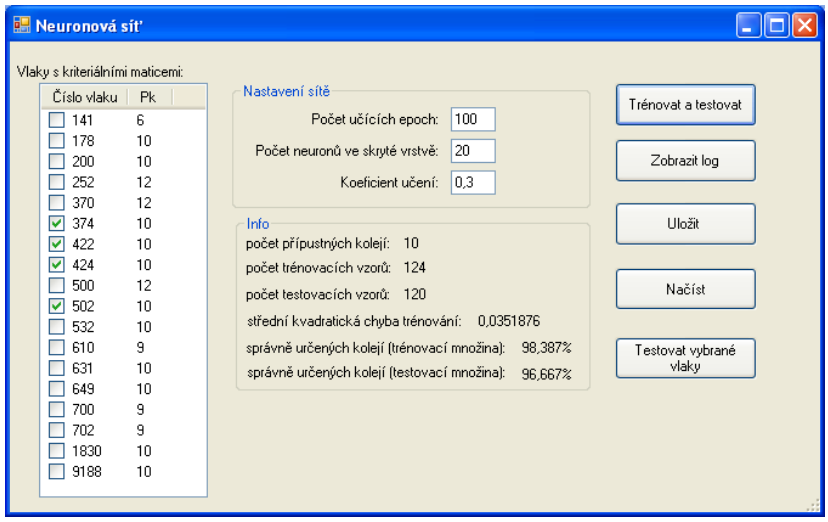
Stiskem tlačítka *Vypsat kriteriální matice* se vyvolá formulář zobrazený na obrázku 54. Zde jsou vypsány jednotlivé kriteriální matice chronologicky podle doby zpoždění. V každé matici je možno označit kolej, kterou by vybral expert jako optimální. Program dokáže experta zastoupit, ale jen do takové míry, že určí nejlepší kolej z hodnot kritérií, přičemž bere kritérium *A* jako nejdůležitější a kritérium *D* jako nejméně důležité. Vytvořené matice lze také exportovat do souboru ve formátu CSV, který je možno otevřít v libovolném tabulkovém procesoru. (ID požadavku: 7)



Obr. 54 ***Výpis kriteriálních matic***

4.4.7Neuronová síť

Hlavní úkol aplikace je implementace umělé neuronové sítě. Obrázek 55 zobrazuje formulář umožňující tuto síť parametrizovat, trénovat a testovat. Formulář je obsluhován třídou *NeuronovaSitDialog*. V levé části se nachází seznam vlaků pro něž byly vytvořeny kriteriální matice, příznak *Pk* znamená počet přípustných kolejí. Výběrem vlaků se určí trénovací a testovací množiny vzorů. Lze vybrat pouze vlaky se stejným počtem přípustných kolejí, a to z důvodu stejného počtu vstupů do neuronové sítě. Program rozdělí množinu kriteriálních matic na dvě podmnožiny tak, že první reprezentuje trénovací množinu (sudé násobky zpoždění vlaků vč.0) a druhá reprezentuje testovací množinu (liché násobky zpoždění vlaků). (ID požadavku: 9 – 12)



Obr. 55 ***Parametrizace, trénování a testování UNS***

Stiskem tlačítka *Trénovat a testovat* se volá metoda *TrenujATestuj*, která provede výstavbu, natrénování a otestování neuronové sítě dle zadaných parametrů (počet epoch učení, počet neuronů ve skryté vrstvě a koeficient učení) a zapíše průběh do souboru *log.txt*. Tvorba sítě je implementována následujícím kódem:

```
#region tvorba site
NeuronovaSit nsit = new NeuronovaSit();

// vstupy
List<NeuronSitVstup> vstupy = new List<NeuronSitVstup>();
for (int i = 0; i < pocetVstupu; i++)
{
  vstupy.Add(nsit.VytvorVstup("x" + (i + 1)));
}

// vystupy
List<NeuronSitVystup> vystupy = new List<NeuronSitVystup>();
for (int i = 0; i < pocetVystupu; i++)
{
  vystupy.Add(nsit.VytvorVystup("y" + (i + 1)));
}

// skryta vrstva
List<NeuronTelo> neuronySkrytaVrstva = new List<NeuronTelo>();
for (int i = 0; i < pocetVeSkryteVrstve; i++)
{
  neuronySkrytaVrstva.Add(nsit.VytvorNeuron(koefUceni, true));
}

// vystupni vrstva
List<NeuronTelo> neuronyVystupniVrstva = new List<NeuronTelo>();
for (int i = 0; i < pocetVystupu; i++)
{
  neuronyVystupniVrstva.Add(nsit.VytvorNeuron(koefUceni, false));
}

// spoji vsechny vstupy se vsemi neurony ve skryte vrstve
for (int i = 0; i < vstupy.Count; i++)
{
  for (int j = 0; j < neuronySkrytaVrstva.Count; j++)
  {
   nsit.VytvorSpoj(vstupy[i], neuronySkrytaVrstva[j], koefUceni);
  }
}
```

```
// spoji vsechny neurony skryte vrstvy s neurony ve vystupni vrstve
for (int i = 0; i < neuronySkrytaVrstva.Count; i++)
{
  for (int j = 0; j < neuronyVystupniVrstva.Count; j++)
  {
    nsit.VytvorSpoj(neuronySkrytaVrstva[i], neuronyVystupniVrstva[j],
                                                              koefUceni);
  }
}

// spoji vystupni neurony s vystupy
for (int i = 0; i < neuronyVystupniVrstva.Count; i++)
{
  // koefUceni=0, vaha=1
  nsit.VytvorSpoj(neuronyVystupniVrstva[i], vystupy[i], 0.0, 1.0);
}
#endregion
```

Následuje trénování sítě, ve kterém se volá metoda *ProvedTrenovaciIteraci-AZpocitejChyby* třídy *NeuronovaSit*. Tato metoda trénuje síť na všechny prvky z trénovací množiny:

```
#region trenovani site
for (int i = 0; i < pocetIteraci; i++)
{
 double stredKvadrChyba =
          nsit.ProvedTrenovaciIteraciAZpocitejChyby(trenovaciMnozina);
#endregion


// třída NeuronovaSit
public double ProvedTrenovaciIteraciAZpocitejChyby(int pocPropIteraci,
NeuronSitTrenTestMnozina trenovaciMnozina)
{
  if (null == trenovaciMnozina) return (0.0);
  double stredniKvadrChyba = 0.0;
  int pocTrenovacichPrvku = trenovaciMnozina.DejPocetTestovacichPrvku();

// natrenuje sit na vsechny prvky z trenovaci mnoziny
  double kvadrChyba = 0.0;
  for (int i = 0; i < pocTrenovacichPrvku; i++)
  {
    trenovaciMnozina.NastavHodnotyVstupnichPrvku(i);
    RozsirHodnotySiti();
    trenovaciMnozina.NastavHodnotyChybVystupu(i);
    PropagaceChyby();
    AktualizujVahyAStrmosti();
    foreach (NeuronSitVystup vystup in vystupyList)
    {
      kvadrChyba += (vystup.Chyba * vystup.Chyba);
    }
  }
  if (pocTrenovacichPrvku > 0)
  {
    stredniKvadrChyba = kvadrChyba / (double)pocTrenovacichPrvku;
  }
  return stredniKvadrChyba;
}
```

Implementace metody *RozsirHodnotySiti*, která rozšíří hodnoty od vstupů přes všechny spoje sítě a vypočítá výstupy z neuronů:

```
public void RozsirHodnotySiti()
{
```

```
  foreach (NeuronTelo neuron in neuronyList)
  {
    neuron.VynulujVstupniCitac();
  }
  //kazdemu spoji priradi vstupni hodnotu
  foreach (NeuronSpoj spoj in spojeList)
  {
    if (null != spoj.VstupniPrvek)
    {
      double vstupniHodnota = 0.0;
      if (true == (spoj.VstupniPrvek is NeuronSitVstup))
      {
        NeuronSitVstup vstupniPrvek = (spoj.VstupniPrvek as NeuronSitVstup);
        vstupniHodnota = vstupniPrvek.Vstup;
      }
      if (true == (spoj.VstupniPrvek is NeuronTelo))
      {
        NeuronTelo neuron = (spoj.VstupniPrvek as NeuronTelo);
        vstupniHodnota = neuron.DejVystup();
      }
      spoj.Vstup = vstupniHodnota;
    }
  }
  //vypocita vystupy ze spoju
  foreach (NeuronSpoj spoj in spojeList)
  {
    spoj.VypocitejVystup();
  }
  //pro kazdy spoj priradi vystupni hodnoty do vstupniho citace neuronu
  //nebo ji posle na vystup
  foreach (NeuronSpoj spoj in spojeList)
  {
    if (null != spoj.VystupniPrvek)
    {
      double vystupniHodnota = spoj.DejVystup();
      if (true == (spoj.VystupniPrvek is NeuronSitVystup))
      {
        NeuronSitVystup vystupniPrvek=(spoj.VystupniPrvek as NeuronSitVystup);
        vystupniPrvek.Vystup = vystupniHodnota;
      }
      if (true == (spoj.VystupniPrvek is NeuronTelo))
      {
        NeuronTelo neuron = (spoj.VystupniPrvek as NeuronTelo);
        neuron.PridejDoVstupnihoCitace(vystupniHodnota);
      }
    }
  }
  // vypocita vystupy z neronů
  foreach (NeuronTelo neuron in neuronyList)
  {
    neuron.VypocitejVystup();
  }
}
```

Metoda *PropagaceChyby* provede zpětné šíření chyby v síti:

```
public void PropagaceChyby()
{
 // kazdemu spoji priradi chybu
 foreach (NeuronSpoj spoj in spojeList)
 {
   if (null != spoj.VystupniPrvek)
   {
     double chyba = 0.0;
     if (true == (spoj.VystupniPrvek is NeuronSitVystup))
     {
      NeuronSitVystup vystupniPrvek = (spoj.VystupniPrvek as NeuronSitVystup);
      chyba = vystupniPrvek.Chyba;
```

```
      }
      if (true == (spoj.VystupniPrvek is NeuronTelo))
      {
        NeuronTelo neuron = (spoj.VystupniPrvek as NeuronTelo);
        chyba = neuron.DejChybovyCitac();
      }
      spoj.Chyba = chyba;
    }
  }
  // vynuluje chybovy citac v kazdem neronu
  foreach (NeuronTelo neuron in neuronyList)
  {
    neuron.VynulujChybovyCitac();
  }
  // vypocita vazenou chybu v kazdem neuronu
  foreach (NeuronSpoj spoj in spojeList)
  {
    spoj.VypocitejVazenouChybu();
  }
  //kazdemu neuronu preda vazenou chybu tak, ze aktualizuje jeho chybovy citac
  foreach (NeuronSpoj spoj in spojeList)
  {
    if (null != spoj.VstupniPrvek)
    {
      double chyba = spoj.DejVazenouChybu();
      if (true == (spoj.VstupniPrvek is NeuronSitVstup))
      {
        //vstup nema chybu, neni treba tedy nic provadet
      }
      if (true == (spoj.VstupniPrvek is NeuronTelo))
      {
        NeuronTelo neuron = (spoj.VstupniPrvek as NeuronTelo);
        neuron.AktualizujChybovyCitacPodleChybyVystupu(chyba);
      }
    }
  }
}
```

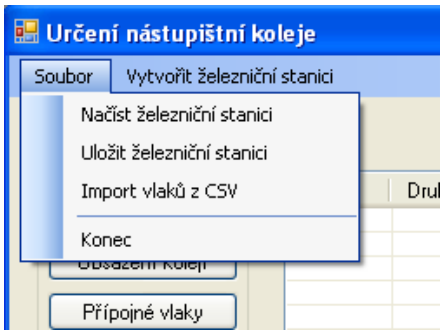
Testování sítě zjistí počet správně určených nástupištních kolejí, testují se trénovací a testovací vzory. Stiskem tlačítka *Testovat vybrané vlaky* lze také vytvořit pouze testovací množinu a její prvky testovat na natrénované neuronové síti. Ukázka vyhodnocení testovací množiny:

```
#region testovani site
float pocTestovacichPrvku = testovaciMnozina.DejPocetTestovacichPrvku();
float pocetSpravnychTest = 0;
for (int i = 0; i < pocTestovacichPrvku; i++)
{
  testovaciMnozina.NastavHodnotyVstupnichPrvku(i);
  nsit.RozsirHodnotySiti();
  if (testovaciMnozina.VypisVstupyVystupyChyby(i))
  { pocetSpravnychTest++; }
}
this.procentSpravneTest = pocetSpravnychTest / pocTestovacichPrvku * 100;
#endregion
```

V tomto formuláři lze také vytvořenou neuronovou síť uložit do souboru a následně ji i načíst a pokračovat v trénování nebo testování (i jiných vlaků než na které byla síť natrénována).

4.4.8Export a import souborů

Vytvořenou železniční stanici lze uložit do datového souboru. Ukládají se parametry stanice s kolejištěm, vlaky včetně přípojů a sestavené kriteriální matice. Ze souboru je pak možno opětovně tyto data načíst. Obrázek 56 ukazuje menu v hlavním okně aplikace, kde lze se soubory pracovat. (ID požadavku: 13, 17)



Obr. 56 ***Export a import souborů***

Při vytváření vlaků může uživatel využít funkci *import vlaků z CSV*. Ta načte do vytvořené stanice vlaky uložené v souboru formátu CSV. Tabulka 1 ukazuje příklad takového souboru v tabulkovém procesoru (KOLEJS1 – nástupištní kolej, KOLEJT1 – vstupní kolej, KOLEJT2 – výstupní kolej).

Tab. 1 ***Příklad importovaného CSV souboru***

CISLO_V	DRUH_V	PRIJEZD	ODJEZD	KOLEJS1	KOLEJT1	KOLEJT2	DELKA
71	EC	5:44	6:00	7	Vs3	Hr	250
73	EC	7:36	8:00	7	Vs3	Hr	250

4.4.9Nasazení a instalace

Program je určen pro platformu Windows a tvoří jej pouze jeden spustitelný soubor. Nevyžaduje žádnou zvláštní instalaci a je možno ho ihned používat. Souborům, které software vytváří, lze určit místo jejich uložení nebo se ukládají implicitně do složky, kde je program umístěn.

4.4.10Kontrola splnění požadavků

Z požadavků na systém, které byly specifikovány v kapitole 4.1, se postupně podařilo splnit požadavky 1 – 17. Dodržení posledního požadavku (ID 18) se ověří až při poslední fázi vývoje tzn. při testování programu, které je popsáno v následující kapitole.

5 Případová studie žst. Praha hlavní nádraží

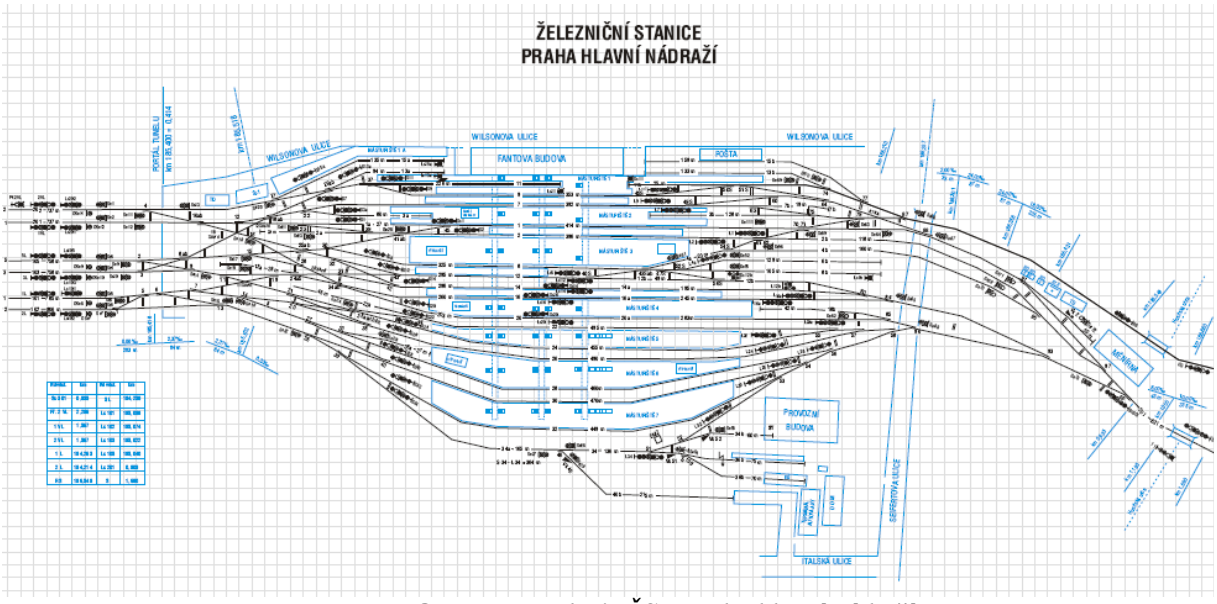
Správná funkčnost navržené implementace, určení nástupištní koleje přijíždějícímu zpožděnému vlaku, bude ověřena na reálných datech z prostředí železniční stanice Praha hlavní nádraží. Použitá infrastruktura a grafikon vlakové dopravy jsou z období 2004/2005. Tato stanice byla vybrána z důvodu velkého množství kolejí a vysoké frekvenci přijíždějících vlaků.

5.1 Příprava dat

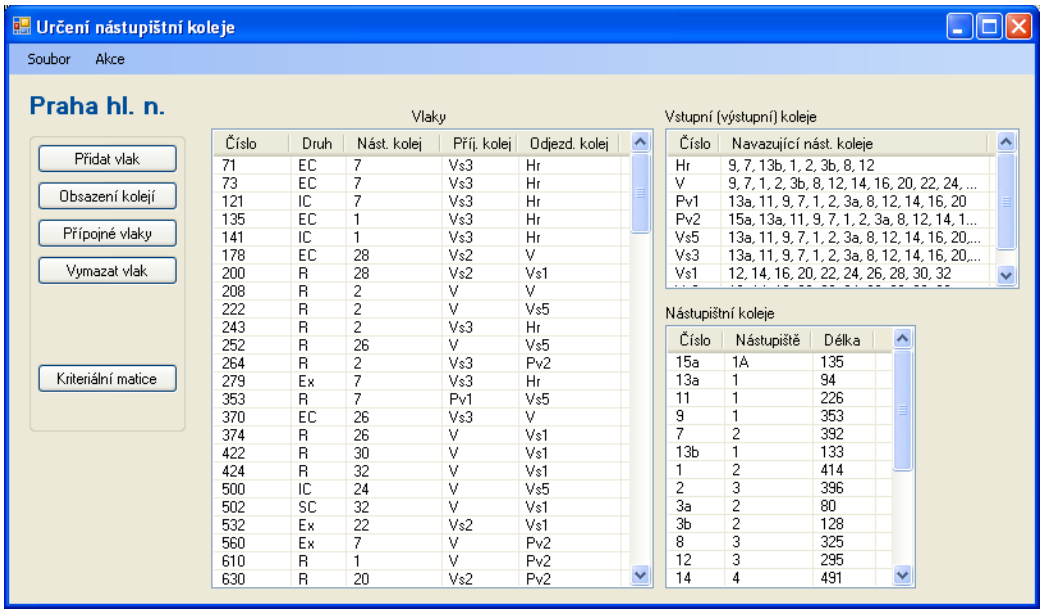
Před samotným testováním bylo potřeba vybrat vhodná testovací data a adaptovat je na prostředí naimplementovaného řešení. Jako vstupní množina dat byl použit soubor *vlaky.dbf* obsahující informace o přijíždějících a odjíždějících vlacích, který byl upraven na soubor *vlaky_upravene.csv* tak, aby jeho struktura vyhovovala importu do vytvořeného programu (oba soubory se nacházejí na přiloženém CD).

Při testování budeme uvažovat provoz pouze ve všední dny a zanedbáme zvláštní a soupravové vlaky. Jako reprezentant problematické části dne byla vybrána ranní dopravní špička 5.30 až 10.00.

Parametrizace vybrané železniční stanice vychází z uspořádání kolejiště znázorněné na obr. 57. Ukázka zachycení topologie železniční stanice a vybrané části GVD v prostředí vytvořeného programu je na obr. 58. Normální a zkrácená doba na přestup byla určena podle [3]. Parametrům ${}^{i}T_{pr}$ a ${}^{i}T_{od}$ potřebným k určení kritérií A a B přidělil expert hodnotu 2 minuty, parametru T_{vyh} pak hodnotu 15 minut (význam parametrů viz kapitola 2.3.1).



Obr. 57 ***Topologie ŽST Praha hlavní nádraží***



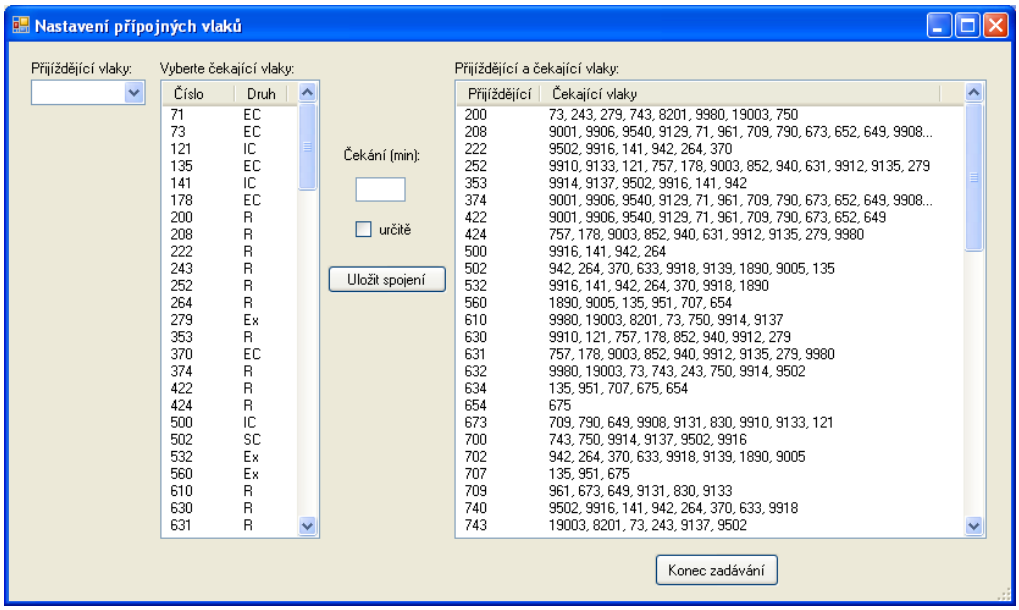
Obr. 58 ***Zachycení dat v prostředí vytvořeného programu***

Pro stanovení kriteria *C* bylo nutné určit k jednotlivým vlakům jejich přípojné (čekající) vlaky (viz kapitola 2.3.3). Ty byly určeny na základě definice přípojného vlaku uvedené v kapitole 2.1.3. Základní i odchylné čekací doby byly přiděleny podle [3]: Základní čekací doba je stanovena mezi jednotlivými kategoriemi vlaků pro všechny stanice ČD shodně:

- expresní vlak (definovaný podle čl.185 předpisu ČD D2[9]) čeká na přípojný expresní vlak 5 minut, na ostatní přípojné vlaky čeká 0 minut,
- vlak R, Sp čeká na přípojné vlaky 5 minut,
- vlak Os čeká na přípojné vlaky 10 minut.

Čekací doby odchylné od základní čekací doby jsou uvedené v *tabulce A* v [3]. Obrázek 59 ilustruje přidělení přípojných vlaků v programu.

[9] ČD D2 – předpis pro organizování a provozování drážní dopravy



Obr. 59 ***Zobrazení vybraných přípojných vlaků***

5.2 Určení testovací a trénovací množiny

Z připravené množiny vstupních dat bylo vybráno 10 přijíždějících vlaků (viz tab. 1) se shodnou množinou přípustných kolejí (nástupištní koleje č. 12–32). Těmto vlakům byla přidělena doba zpoždění od 0 do 60 minut. Vzniklo tak 610 kriteriálních matic (viz kapitola 2.4), které se vyčíslily pomocí vytvořeného programu. Kritéria *A*–*C* určil program sám (viz kapitoly 2.3.1–2.3.3). Hodnota kritéria *D* byla stanovena podle vzdálenosti ohodnocované nástupištní koleje od plánované nástupištní koleje tak, že plánovaná kolej má hodnotu tohoto kritéria rovnu 1 a nejvzdálenější pak hodnotu 0.

Tab. 2 ***Tabulka s vybranými vlaky***

Číslo vlaku	Čas příjezdu	Příjezdová kolej	Čas odjezdu	Odjezdová kolej	Nástupištní kolej	Nástupiště
200	7.25	Vs2	8.02	Vs1	28	6
374	5.37	V	5.51	Vs1	26	6
422	5.24	V	6.02	Vs1	30	7
424	6.57	V	7.45	Vs1	32	7
502	9.02	V	9.19	Vs1	32	7
532	8.36	Vs2	9.10	Vs1	22	5
631	6.57	V	7.23	Vs1	22	5
649	5.39	Vs2	6.23	Vs1	22	5
1830	7.14	Vs2	7.57	Vs1	26	6
9188	5.31	Vs2	6.01	Vs1	28	6

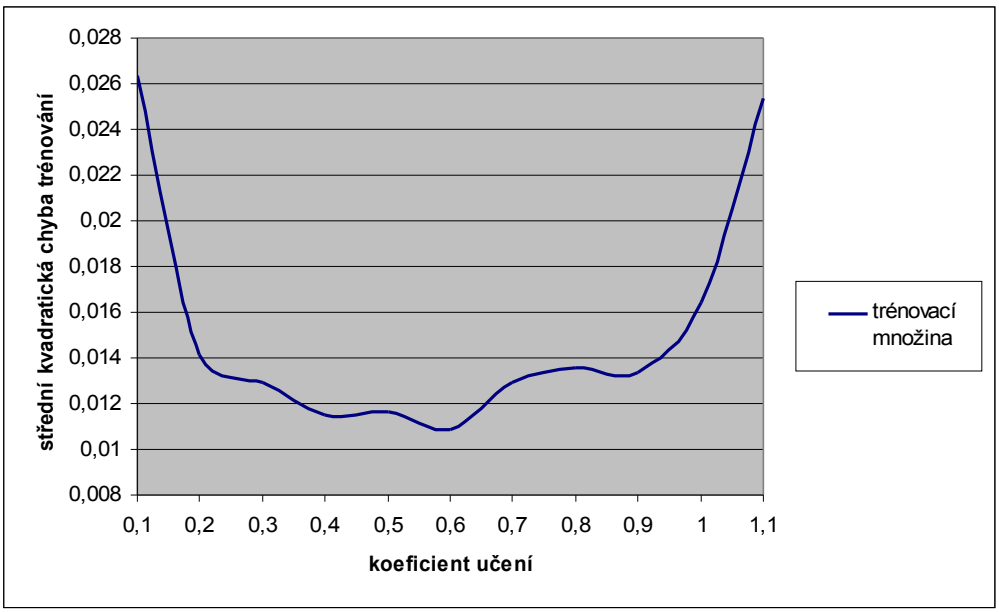
Každá ze 610 variant kriteriální matice byla expertně vyhodnocena a pro každou byla určena optimální nástupištní kolej. Poté došlo na rozdělení množiny těchto kriteriálních matic na trénovací a testovací množinu (viz kapitola 4.4.7).

5.3 Parametrizace neuronové sítě

Správná volba parametrů zásadně ovlivňuje úspěšnost natrénování neuronové sítě. Jak již bylo naznačeno v kapitole 3.3, neexistuje žádná univerzální metodika, která by umožňovala přesný výpočet jednotlivých koeficientů, lze se tak řídit pouze doporučeními. V programu lze zadávat počet neuronů ve skryté vrstvě, koeficient učení a počet učících epoch. Za počáteční hodnoty vah spojení a prahů jednotlivých neuronů jsou programem voleny náhodné hodnoty (viz kapitola 3.3).

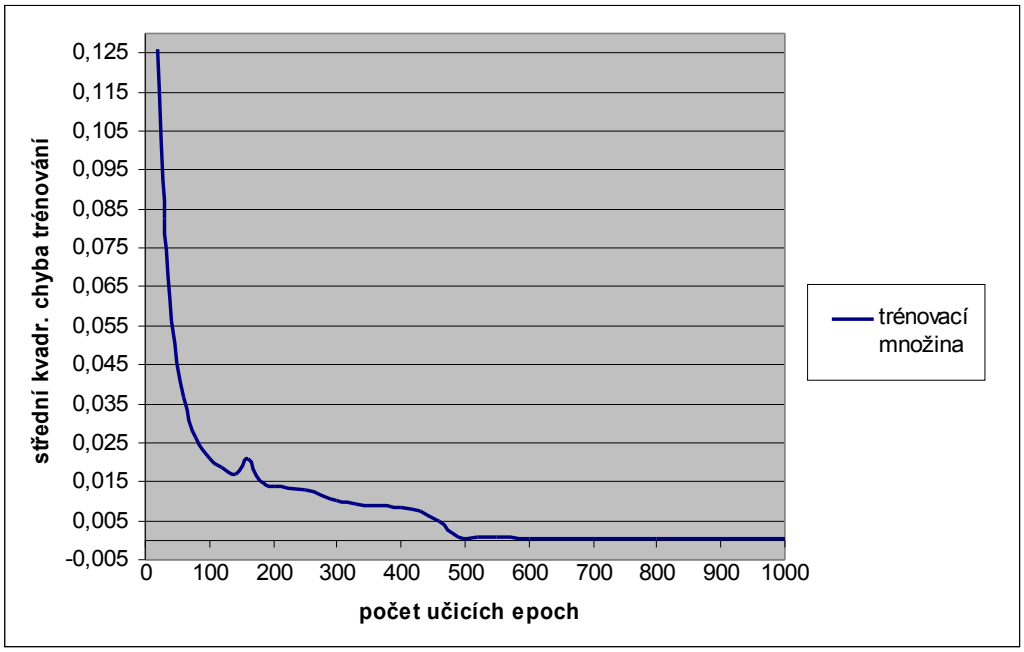
Vzhledem k tomu, že při hledání nejvhodnějších parametrů vzniká množství různých modelů UNS, je třeba tyto nějak rozlišit podle jejich kvality. Nejčastěji využívaným kritériem pro řešení tohoto problému je střední kvadratická chyba trénování E_{2tr}, kterou se snažíme minimalizovat.

Na obrázku 60 je graf závislosti E_{2tr} na hodnotě koeficientu učení při počtu 200 učících epoch a 20 neuronech ve skryté vrstvě. Počet neuronů ve skryté vrstvě byl stanoven dle vztahu (8). Na obrázku je vidět, že nejmenší hodnotu E_{2tr} dosáhneme při koeficientu učení $\alpha = 0{,}6$.



Obr. 60 ***Graf závislosti střední kvadr. chyby trénování na koef. učení***

Pro úspěšné natrénování neuronové sítě je také důležité zvolit dostatečný počet epoch učení. Graf na obrázku 61 ilustruje závislost E_{2tr} na počtu učících epoch při koeficientu učení $\alpha = 0{,}6$ a 20 neuronech ve skryté vrstvě. Z grafu plyne, že po 500 učících iteracích se již E_{2tr} nijak výrazně nemění.

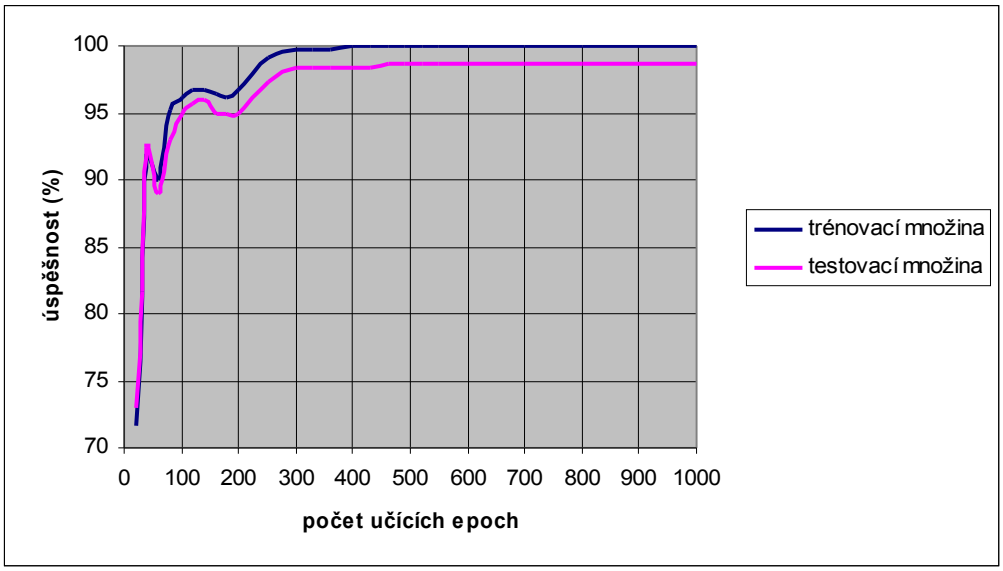


Obr. 61 *Graf závislosti střední kvadr. chyby trénování na počtu epoch učení*

5.4 Úspěšnost neuronové sítě

V předchozí kapitole byly otestovány hodnoty nastavitelných parametrů trénování neuronové sítě. Jako nejlepší kombinace se jeví koeficient učení $\alpha = 0{,}6$ a 20 neuronů ve skryté vrstvě.

Obrázek 62 zobrazuje jak se mění úspěšnost správného určení koleje neuronovou sítí v závislosti na počtu epoch učení. Po 500 trénovacích iterací byla neuronová síť schopna určit správně 100 % případů z trénovací množiny a 98,333 % z testovací množiny.



Obr. 62 *Graf závislosti počtu epoch učení na úspěšnost UNS*

6 Závěr

Cílem práce bylo vytvoření programu, který ověří možnosti implementace umělé neuronové sítě jako prvku pro rozhodování o přidělení nástupištní koleje přijíždějícímu zpožděnému vlaku.

V teoretické části byl popsán problém přidělení nástupištní koleje zpožděnému vlaku s charakterizací čtyř kritérií výběru koleje *A–D* a dle nich vypočtené kriteriální matice. Kritéria však sama o sobě nestačí, je třeba stanovit jejich váhu pomocí vhodné metodiky, která určí relativní důležitost těchto kritérií a rozhodne o přidělení správné koleje. K tomuto účelu byly vybrány neuronové sítě a nastíněna jejich problematika. Bylo naznačeno rozdělení typů sítí podle různých kritérií a popsán neuron. Podrobněji byl pak rozebrán návrh topologie neuronové sítě, její parametrizace a algoritmus učení Back-propagation.

V praktické části byl vytvořen výše zmíněný program. Při vývoji tohoto programu byla využita metodika Unified Process (metodika tvorby softwarového vybavení) a vizuální modelovací jazyk UML. Program byl vytvořen v prostředí jazyka C# s využitím software MS Visual Studio 2005.

Hotový program umožňuje zachycení topologie železniční stanice, zadávání vlaků včetně jejich přípojů, vytvoření kriteriálních matic pro zpoždění vlaků a výstavbu, natrénování a otestování neuronové sítě na správné přidělení nástupištních kolejí. Je také možno všechna vytvořená data ukládat a načítat. Neuronovou síť lze natrénovat a testovat vždy jen pro vlaky se stejnou množinou tzv. přípustných kolejí. Takových sítí lze však v programu vytvořit neomezeně, tím pádem je možné mít natrénovanou neuronovou síť pro každou množinu přípustných kolejí. Program by bylo vhodné doplnit o modul, který by umožňoval graficky vykreslit stav obsazení kolejí v čase příjezdu daného zpožděného vlaku. Tento úkol však nebyl z časových důvodů realizován.

Program byl prakticky vyzkoušen na případové studii žst. Praha hlavní nádraží. Vybraným vlaků byly vypočteny kriteriální matice, které byly expertně vyhodnoceny. Vyhodnocená data byla rozdělena do tzv. trénovací a testovací množiny. Po nalezení optimálních parametrů byla neuronová síť natrénována a otestována. Bylo ukázáno, že při dostatečném množství trénovacích iterací, je schopna daná neuronová síť určit správně 100 % případů z trénovací množiny a přibližně 98 % z testovací množiny. Pro zvýšení efektivity by bylo nutné předložit neuronové síti více trénovacích vzorů.

Uložené natrénované sítě by bylo možno využít jako rozhodovacích prvků v přidělování náhradní nástupištní koleje zpožděným vlakům v simulaci provozu železniční stanice.

POUŽITÁ LITERATURA

[1] BAŽANT, Michael, ŽARNAY, Michal. *Formalizace řešení přidělení náhradní nástupištní koleje pro zpožděný vlak. Sborník příspěvků konference INFOTRANS 2005.* Pardubice: Univerzita Pardubice, 2005. s.29-34. ISBN 80-7194-792-X.

[2] KUBÁT, Bohumil. *Železniční tratě a stanice.* Praha: Vydavatelství ČVUT, 1998. s.80, 89-90. ISBN 80-01-01850-4

[3] *Čekací doby a opatření při zpoždění vlaků osobní dopravy, platný od 12. 12. 2004.* Praha: Odbor 16, GŘ České dráhy a. s., 2004.

[4] NOVÁK, M. a kol. *Umělé neuronové sítě. Teorie a aplikace.* Praha: Nakladatelství C. H. BECK, 1998. s.153-154, 184, 191-194. ISBN 80-7179-132-6

[5] TAUFER, I., DRÁBEK, O., SEIDL, P. *Umělé neuronové sítě– teorie a aplikace (4)*. CHEMagazín, 2 (XVI), 2006, s. 33-36. ISSN 1210-7409.

[6] ŠKUTOVÁ, Jolana, *Neuronové sítě v řízení systémů. Aktivační funkce neuronových sítí* [online]. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2004. Dostupný z WWW: <http://www.fs.vsb.cz/books/NeuronoveSite/Aktiv_funkce.htm>

[7] KUPKA, Karel, *Kvalimetrie. Řízení a plánování jakosti pomocí NN a PLS metod.* [online]. Pardubice, Dostupný z WWW: <http://www.statspol.cz/request/request2006/sbornik/kupka.pdf>

[8] TAUFER, I., DRÁBEK, O., SEIDL, P. *Umělé neuronové sítě– teorie a aplikace (6)*. CHEMagazín, 6 (XVI), 2006, s. 31-33. ISSN 1210-7409.

[9] ČESNEK, M., *Metody určování nástupištní koleje pro zpožděný přijíždějící vlak v osobních železničních stanicích s využitím výpočetní techniky.* Pardubice, 2008. 81 s. Diplomová práce na Dopravní fakultě Jana Pernera Univerzity Pardubice. Vedoucí diplomové práce Michael Bažant.

[10] ARLOW, J., NEUSTADT, I., *UML 2 a unifikovaný proces vývoje aplikací. Objektově orientovaná analýza a návrh prakticky*., přeložil KISZKA, B., Brno: Nakladatelství Computer Press, a.s., 2007. s.28, 53, 78, 330. ISBN 978-80-251-1503-9

[11] TAUFER, I., DRÁBEK, O., SEIDL, P. *Umělé neuronové sítě– teorie a aplikace (1)*. CHEMagazín, 4 (XV), 2005, s. 32-35. ISSN 1210-7409.

[12] KAVIČKA, A., *Sylaby přednášek předmětu „Datové struktury a algoritmy" v magisterském studiu 4-1&2*. Pardubice: Univerzita Pardubice, 2007.

[13] AVL strom. In *Wikipedie : otevřená encyklopedie* [online]. St. Petersburg (Florida): Wikimedia Foundation, 2001, strana naposledy edit. 2009-04-22 [cit. 2009-05-18]. Česká verze. Dostupný z WWW: < http://cs.wikipedia.org/wiki/AVL-strom >

[14] TAUFER, I., DRÁBEK, O., SEIDL, P. *Umělé neuronové sítě– teorie a aplikace (3)*. CHEMagazín, 1 (XVI), 2006, s. 12-14. ISSN 1210-7409.