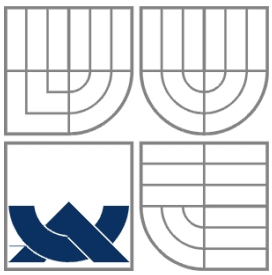


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PROGRAMOVÝ SYSTÉM PRO ŘEŠENÍ ÚLOH DYNAMICKÉHO PROGRAMOVÁNÍ

PROGRAM SYSTEM FOR SOLVING DYNAMIC PROGRAMMING PROBLEMS

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE **BC. PETR ZETKA**
AUTHOR

VEDOUCÍ PRÁCE **RNDR. JIŘÍ DVOŘÁK, CSC.**
SUPERVISOR

BRNO 2011

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2010/11

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Petr Zetka

který/která studuje v **magisterském studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Programový systém pro řešení úloh dynamického programování

v anglickém jazyce:

Program system for solving dynamic programming problems

Stručná charakteristika problematiky úkolu:

Dynamické programování je nástroj pro optimalizaci rozhodovacích procesů. Cílem je najít takovou rozhodovací strategii, která optimalizuje funkci sdruženou s daným procesem.

Cíle diplomové práce:

1. Charakterizovat problematiku dynamického programování.
2. Popsat vybrané typy úloh dynamického programování včetně postupů jejich řešení.
3. Navrhnout a implementovat programový systém umožňující řešení těchto úloh na počítači.
4. Ověřit funkčnost vytvořeného programového systému a vypracovat návod na jeho použití.

Seznam odborné literatury:

BRONSON, R.; NAADIMUTHU, G. Schaum's Outline of Theory and Problems of Operations Research. New York, McGraw-Hill, 1997.
LEW, A.; MAUCH H. Dynamic Programming: A Computational Tool.
Springer-Verlag Berlin Heidelberg, 2007.
de MOOR, O. Dynamic Programming as a Software Component. In: Mastorakis, N. (editor), Procs. of CSCC, Athens, July 1999. WSES Press.
LAŠČIAK, A. a kol. Dynamické modely. Bratislava, Alfa/SNTL, 1985.

Vedoucí diplomové práce:RNDr. Jiří Dvořák, CSc.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2010/11.

V Brně, dne 24. 11. 2010

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ DĚKANÁT

Ing. Jan Roupec, Ph.D.
Ředitel ústavu

prof. RNDr. Miroslav Doupovec, CSc.
Děkan

ABSTRAKT

Diplomová práce se zabývá tvorbou programového systému pro řešení úloh dynamického programování na počítači. V teoretické části práce je popsáno dynamické programování jako nástroj pro optimalizaci víceetapových rozhodovacích procesů a vybrané úlohy dynamického programování, které byly implementovány do programového systému. V praktické části je popsán návrh, implementace a ověření funkčnosti programového systému.

ABSTRACT

This work deals with building a program system for solving dynamic programming problems on a computer. The theoretical part describes dynamic programming as a tool used for optimizing multistage decision processes and dynamic programming problems implemented in the program system. The practical part describes the design and implementation of the program system and verification of its functionality.

KLÍČOVÁ SLOVA

Optimalizace, dynamické programování, úlohy dynamického programování, C#.

KEYWORDS

Optimization, dynamic programming, dynamic programming problems, C#.

BIBLIOGRAFICKÁ CITACE

ZETKA, P. Programový systém pro řešení úloh dynamického programování. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2011. 79 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, CSc..

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že jsem diplomovou práci zpracoval samostatně dle pokynů vedoucího práce s použitím uvedené literatury.

Datum: 27. 5. 2011 .....

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval vedoucímu práce RNDr. Jiřímu Dvořákovi, CSc za cenné rady a připomínky, které vedly k úspěšnému dokončení diplomové práce.

Obsah:

1 ÚVOD

Práce se zabývá optimalizací víceetapových rozhodovacích procesů pomocí dynamického programování. Dynamické programování je nástroj pro optimalizaci rozhodovacích procesů probíhajících v čase, ale je možné jej využít i u procesů, kde čas přímo nevystupuje. Dynamické programování bylo poprvé publikováno v roce 1957 a autorem je Richard Ernest Bellman [1]. Dynamické programování je založeno na principu optimality a při výpočtu využívá funkcionálních rovnic. Pro výpočet úloh neexistuje obecná funcionální rovnice, ale je nutné pro každou konkrétní třídu problémů sestavit funkcionální rovnici pro její řešení.

Hlavním cílem této práce je vytvořit programový systém pro řešení úloh dynamického programování. Systém by měl sloužit jako pomůcka při výuce předmětu Optimalizace. V dosavadní výuce se využíval pro řešení úloh tabulkový procesor Excel, kde rovnice pro výpočet byly zavedeny funkcemi Excelu.

Dalším cílem je implementovat do systému některé úlohy dynamického programování a pomocí těchto úloh ověřit funkčnost systému. Před návrhem a vytvořením programového systému je nutné nejdříve prostudovat problematiku optimalizace pomocí dynamického programování a popsat vybrané typy optimalizačních úloh, řešitelných pomocí dynamického programování. Už při návrhu by mělo být bráno v úvahu, že systém by měl umožňovat snadné doprogramování dalších úloh.

Posledním cílem je vytvořit návod pro použití programového systému a ověřit funkčnost programového systému. Konkrétně jde o návod pro řešení úloh pomocí programového systému a návod pro přidávání nových úloh do systému. Funkčnost programového systému a úloh implementovaných v programovém systému lze ověřit vypočtením vhodného množství příkladů pro každý typ úlohy.

2 DYNAMICKÉ PROGRAMOVÁNÍ

Dynamické programování je matematický přístup k řešení problémů víceetapových rozhodovacích procesů. Tento matematický přístup je založen na rekurentních fukcionálních rovnicích a na tzv. principu optimality. Autorem je Richard Ernest Bellman, který publikoval dynamické programování v roce 1957 [1].

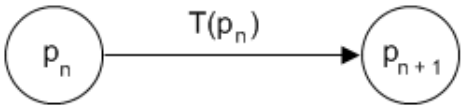
Dynamické programování, jak je uvedeno v názvu, se zabývá převážně rozhodovacími procesy probíhajícími v čase, je však možné je použít i v úlohách, kde čas přímo nevystupuje, ale může být do nich zaveden uměle. Dynamické programování je použitelné k řešení široké třídy optimalizačních problémů a to deterministických i stochastických, diskrétních i spojitých.

Optimalizace procesů pomocí dynamického programování bývá využívána ve velkém množství vědních oborů, jako např.: v ekonomii, technice, chemii, fyzice, ale i v matematice. Dynamické programování je použitelné pro získání přesného řešení i v případech, kdy jiné metody selhávají, avšak vyžaduje zpravidla náročnější matematickou přípravu než aplikace např. lineárního programování.

Problémy řešitelné dynamickým programováním nemusí být charakterizovány systémem rovnic nebo nerovic, proto neexistuje žádný obecný algoritmus pro výpočet dané třídy úloh, ale každá úloha v sobě zahrnuje také vytvoření algoritmu pro její řešení. Algoritmus přitom značně závisí na struktuře řešeného problému. Dynamické programování můžeme rozdělit podle vlastností časové množiny na spojité a diskrétní, podle vlivu náhodných faktorů na deterministické a stochastické; dále je možné je dělit podle toho, zda výsledek transformace závisí na volbě počátečního okamžiku na stacionární a nestacionární [2].

2.1 Víceetapový proces

Jelikož metoda dynamického programování se zabývá řešením víceetapových rozhodovacích procesů, je vhodné se nejprve zmínit o některých vlastnostech, které mají víceetapové procesy. Víceetapový proces lze rozložit na jednotlivé části, v dynamickém programování o jednotlivých částech mluvíme jako o etapách nebo stupních; je-li proces zkoumán v čase, pak proces rozdělujeme podle časových úseků.



Obr. 1 Schéma víceetapového procesu

Každá etapa procesu je charakterizována určitým stavem a každému stavu etapy přísluší hodnoty proměnných veličin. Stav procesu budeme označovat symbolem p a budeme předpokládat, že p je prvkem množiny R přípustných stavů procesu. Uvažujeme transformační funkci $T(p)$, která má takovou vlastnost, že transformovaný stav $p_1 = T(p)$ patří do množiny přípustných stavů R, pokud stav p patřil do množiny R, kde p představuje počáteční stav systému a p_1 stav o jednu etapu později. Pokud bychom víceetapový proces znázornili grafem, tak uzly grafu by představovaly stavy procesu a hrany grafu by představovaly aplikaci transformační funkce, jak je znázorněno na Obr. 1.

$$p_{n+1} = T(p_n) \tag{2.1}$$

$$\{p, p_1, p_2, \ldots, p_N\} \tag{2.2}$$

Obecně tedy můžeme proces popsat rovnicí (2.1), kde p_n představuje stavy procesu pozorované v časových okamžicích $n = 0,1,2,\ldots,N\text{-}1$. Posloupnost stavů (2.2) nazýváme N-etapovým procesem. N-etapový proces, kde N je konečné, nazýváme procesem konečným, ovšem můžeme použít i proces nekonečný pro aproximaci procesu s velkým počtem etap. Pokud počet etap procesu

není předem dán, ale závisí na počátečním stavu, potom jde o proces neohraničený. Tento proces skončí, jakmile splňuje danou podmínku. Opakem procesu neohraničeného je potom proces ohraničený, který má předem daný počet etap a skončí po uvedeném počtu etap [2].

Dále můžeme rozlišovat procesy podle toho, zda tvar transformační funkce závisí na etapě procesu, nebo ne. Pokud tvar transformační funkce závisí na etapě procesu (2.3), jedná se o proces nestacionární, v opačném případě se jedná o proces stacionární a platí (2.1).

$$p_{n+1} = T_n(p_n) \tag{2.3}$$

Zatím jsme se zabývali pouze takovými procesy, u kterých transformační funkce T převádí stav p_n na stav p_{n+1}, kde stav p_{n+1} je jednoznačně určen předchozím stavem p_n. V takovém případě se jedná o proces deterministický. V mnohých případech, kdy transformační funkce T není zcela známa, musíme nahradit deterministické procesy složitějšími, k čemuž nám poslouží teorie pravděpodobnosti [2].

Pokud zavedeme do transformační funkce T vzájemně nezávislé náhodné proměnné r, pak dostaneme transformační funkci pro stochastický proces (2.4). Z hlediska filozofického determinismu je každý reálný proces deterministický a stochastický proces je potom pouze matematický model, který je vhodné využít v případě, že nejsme schopni poznat všechny vlivy, které ovlivňují transformační funkci T, nebo zavedení deterministické formulace by bylo oproti stochastické formulaci nadmíru náročnější bez adekvátního přínosu a tím pádem neefektivní [2].

$$p_{n+1} = T(p_n, r_{n+1}) \tag{2.4}$$

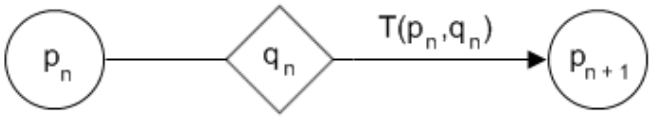
2.2 Víceetapový rozhodovací proces

Vyjdeme-li z víceetapového procesu a budeme-li předpokládat, že můžeme tento proces ovlivnit volbou hodnot nějakých proměnných q_n , pak se transformační funkce stane funkcí dvou proměnných (2.5).

$$p_{n+1} = T(p_n, q_n) \tag{2.5}$$

$$F(p, p_1, p_2, \ldots, q, q_1, q_2, \ldots) \tag{2.6}$$

Potom proměnná q_n se nazývá rozhodovací proměnná a nabývá hodnot z množiny přípustných rozhodnutí $S(p_n)$. Volba hodnoty rozhodovací proměnné q_n v n-té etapě procesu je rozhodnutím. Budeme se zabývat procesy, u kterých jsou hodnoty rozhodovacích proměnných q_n zvoleny tak, aby maximalizovaly předepsanou skalární funkci stavových a rozhodovacích proměnných (2.6). Potom je funkce F tzv. účelová funkce, nebo také kriteriální funkce [2].



Obr. 2 Schéma víceetapového rozhodovacího procesu

Graficky můžeme znázornit víceetapový rozhodovací proces s transformační funkcí, tak jak je ukázáno na Obr. 2.

N-etapovým rozhodovacím procesem diskrétního deterministického typu nazveme posloupmost stavových a rozhodovacích proměnných (2.7), kde platí (2.5) pro $n = 0,1,2,\ldots,N\text{-}1$.

$$\{p, p_1, p_2, \ldots, p_N, q, q_1, q_2, \ldots q_N\} \tag{2.7}$$

$$q_n = (p, p_1, p_2, \ldots, p_n, q, q_1, q_2, \ldots, q_{n-1}) \tag{2.8}$$

Je-li rozhodovací funkce (2.8) funkcí minulých a přítomného stavu a minulých rozhodnutí, pak posloupnost rozhodovacích proměnných $\{q,q_1,q_2,...,q_N\}$ nazveme strategií procesu. Optimální strategií procesu pak nazveme takovou posloupnost rozhodovacích proměnných, která maximalizuje účelovou funkci F. Hlavní využití dynamického programování je při hledání optimální strategie rozhodovacího procesu, tj. optimalizace tohoto procesu [2].

Základní dva tvary účelových funkcí F jsou: aditivní

$$F(p, p_1, \ldots, p_N, q, q_1, \ldots, q_N) = \sum_{i=0}^{N} g_i(p_i, q_i)$$

kde $g_i(p_i, q_i)$ je složna aditivní funkce pro etapu procesu i, hodnotu stavu p_i a rozhodnutí q_i a multiplikativní

$$F(p, p_1, \ldots, p_N, q, q_1, \ldots, q_N) = \prod_{i=0}^{N} g_i(p_i, q_i)$$

kde $g_i(p_i, q_i)$ je složna multiplikativní funkce pro etapu procesu i.

2.3 Funkcionální rovnice a princip optimality

Funkcionální rovnice N-etapového diskrétního deterministického procesu s aditivní účelovou funkcí je (2.9). K funkcionální rovnici přísluší podmínka pro první etapu procesu (2.10).

$$f_N(p) = \max_q [g(p, q) + f_{N-1}(T(p, q))] \tag{2.9}$$

$$f_0(p) = \max_q g(p, q) \tag{2.10}$$

Tuto funkcionální rovnici lze získat na základě Bellmanova principu optimality, což je intuitivní přístup, kterým lze najít optimální strategii. Princip optimality je formulován takto: je-li posloupnost rozhodnutí $\{q,q_1,q_2,...q_N\}$ optimální strategií N-etapového procesu s počátečním stavem p, pak posloupnost rozhodnutí $\{q_1,q_2,...q_N\}$ tvoří opět optimální strategii (N-1)-etapového procesu s počátečním stavem p_1, kde počáteční stav $p_1 = T(p,q)$ [2]. Takto byl převeden problém hledání optimální rozhodovací strategie na problém řešení funkcionální rovnice. Zatím co optimálních strategií může mít proces více než pouze jednu, tak optimální hodnota účelové funkce je pouze jedna, pokud ovšem existuje.

Při řešení úloh dynamického programování se v této práci využívá tabulková metoda. Tato metoda je založena na výpočtu úlohy po etapách a zapsání vypočtených hodnot do tabulek. V jednotlivých etapách se řeší funkcionální rovnice pro všechny přípustné hodnoty stavů, tj. všechny stavy, které by mohly vlivem všech přípustných rozhodnutí a případně náhodných proměnných u stochastických úloh nastat. Výpočet úlohy tabulkovou metodou jde rozdělit do dvou fází. V první fázi se počítá optimální hodnota účelové funkce a ve druhé fázi se hledá optimální rozhodovací strategie. Při výpočtu optimální hodnoty účelové funkce se postupuje při zpětném dynamickém programování od poslední etapy procesu k první etapě procesu. V poslední etapě výpočtu je vypočtena optimální hodnota účelové funkce pro počáteční hodnotu stavu. Při výpočtu optimální strategie se postupuje od první etapy procesu směrem k poslední. V poslední etapě výpočtu optimální hodnoty účelové funkce se získá optimální hodnota prvního rozhodnutí. Pomocí transformace se vypočítá hodnota stavu po prvním rozhodnutí. Pro vypočtenou hodnotu stavu se hledá optimální rozhodnutí vypočtené v předchozí fázi výpočtu. Tento postup se opakuje až jsou získány všechny hodnoty optimálních rozhodnutí a tím je získána optimální strategie.

2.4 Deterministické procesy

Deterministické procesy byly z velké části vysvětleny v podkapitolách 2.2 a 2.3, kde právě na deterministických diskrétních procesech byly vysvětlovány některé pojmy. Následující část bude z větší části pouze shrnutí a ucelení zavedených pojmů z předcházejících kapitol. Předpoklad pro deterministické procesy je, že transformační funkce je tvaru (2.5), tj. funkce dvou proměnných a to funkce stavu p a rozhodnutí q. Proto stav procesu p_{n+1} lze získat na základě předchozího stavu p_n a rozhodnutí q_n.

Užitím funkcionálních rovnic (2.9) a (2.10) dostaneme optimální strategii pro aditivní účelovou funkci pro deterministický proces. Další možné tvary transformačních funkcí, funkcionálních rovnic a účelových funkcí budou uvedeny v rámci podkapitoly 3.1 o deterministických úlohách dynamického programování.

2.5 Stochastické procesy

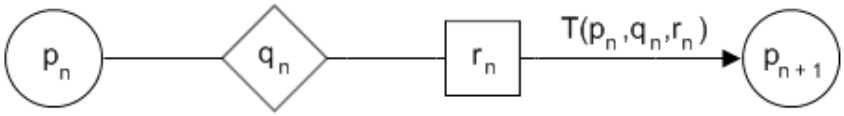
Budeme se zabývat procesy, ve kterých hraje svou roli neurčitost, tj. stochastickými neboli pravděpodobnostními procesy a omezíme se pouze na procesy diskrétní stochastické.

Budeme-li předpokládat, že v počátečním stavu p bylo přijato rozhodnutí q, pak na rozdíl od deterministických procesů nebude jednoznačně určen nový stav p_1. Nový stav procesu p_1 bude kromě předchozího stavu p a rozhodnutí q záviset také na hodnotě náhodné proměnné, kde náhodná proměnná vyjadřuje vliv náhodných faktorů na proces [2].

$$p_1 = T(p, q, r_0) \tag{2.11}$$

$$p_{n+1} = T(p_n, q_n, r_n) \tag{2.12}$$

Nový stav p_1 bude určen vztahem (2.11). Protože je stav p_1 závisý dle vztahu (2.11) na náhodné proměnné r_0, je také náhodnou proměnnou. Analogickým postupem, tj. volbou rozhodnutí q_1 bychom dostali vztah pro výpočet stavu p_2. Tento postup bychom mohli opakovat n-krát a dostali bychom obecný vztah pro výpočet stavu p_{n+1} pomocí rozhodnutí q_n (2.12).



Obr. 3 Schéma stochastického víceetapového rozhodovacího procesu

Stochastický proces a tvar transformace pro stochastický proces můžeme schématicky znázornit způsobem, jakým je znázorněn na Obr. 3. Budeme předpokládat vzájemnou nezávislost proměnných r_i, které představují vliv náhodných faktorů na proces v v jednotlivých etapách. Budeme-li mít N-etapový proces a přiřadíme mu aditivní účelovou funkci, dostaneme vztah (2.13).

$$F(p, p_1, ..., p_N, q, q_1, ..., q_N, r_0, r_1, ..., r_N) = \sum_{i=0}^{N} g_i(p_i, q_i, r_i) \tag{2.13}$$

Jestli strategie $\{q,q_1,q_2,...,q_N\}$ maximalizuje očekávanou hodnotu účelové funkce (2.13), pak ji budeme považovat za optimální strategii. Optimální strategie může být v každé konkrétní realizaci procesu obecně jiná, závisí totiž na konkrétních hodnotách náhodných faktorů a to i při stejném počátečním stavu. Úkolem ve stochastických procesech na rozdíl od deterministických není najít pouze posloupnost rozhodnutí, ale pro každý přípustný stav, tj. stav, který může volbou rozhodovací proměnné a vlivem náhodné proměnné nastat, najít optimální rozhodnutí. Skutečnou hodnotu účelové funkce F lze oproti deterministickým procesům určit až v průběhu procesu v závislosti na působení náhodných faktorů [2].

Zavedeme-li do vztahů (2.9) a (2.10) očekávanou hodnotu E_r, dostaneme funkcionální rovnici pro N-tou etapu (2.14) a k rovnici příslušící omezující podmínku (2.15) [2].

$$f_N(p)=\max_q E_{r_0}[g_0(p,q,r_0)+f_{N-1}(T(p,q,r_0))] \tag{2.14}$$

$$f_0(p)=\max_q E_{r_0} g_0(p,q,r_0) \tag{2.15}$$

Předpokládáme-li, že všechny náhodné proměnné r_i mají stejnou distribuční funkci $G(r)$, dostaneme vztahy (2.16) a (2.17). Budou-li r_i diskrétní náhodné proměnné, které mohou nabývat hodnot $S_1,...,S_M$ s pravděpodobnostmi $P_1,...P_M$ lze psát vztahy (2.16) a (2.17) ve tvaru (2.18) a (2.19) [2].

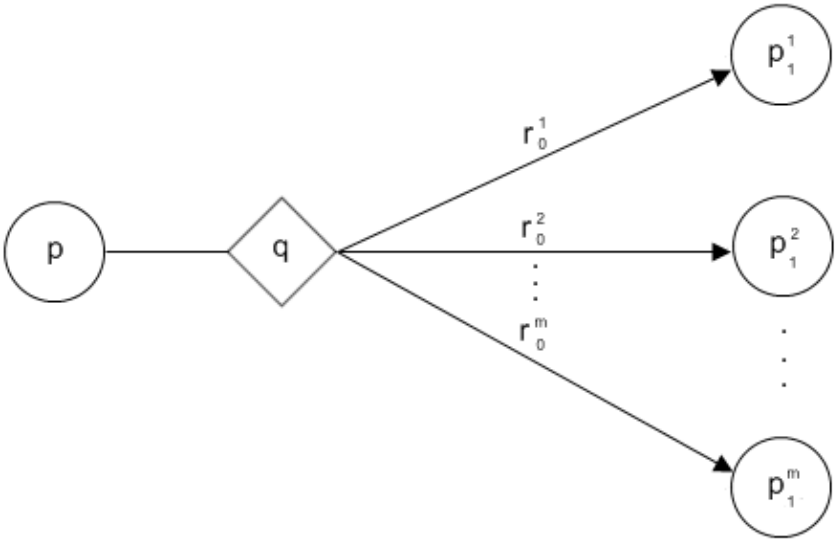
$$f_N(p)=\max_q \int [g_0(p,q,r_0)+f_{N-1}(T(p,q,r_0))]dG(r) \tag{2.16}$$

$$f_0(p)=\max_q \int g_0(p,q,r_0)\,dG(r) \tag{2.17}$$

$$f_N(p)=\max_q \{\sum_{j=1}^{M} P_j[g_0(p,q,S_0)+f_{N-1}(T(p,q,S_j))]\} \tag{2.18}$$

$$f_N(p)=\max_q [\sum_{j=1}^{M} P_j g_0(p,q,S_0)] \tag{2.19}$$

Užitím rekurentních funkcionálních vztahů (2.18) a (2.19) dostaneme optimální posloupnost rozhodovacích funkcí $\{q_i(p_i)\}$, kde členy této posloupnosti jsou funkcemi náhodného stavu procesu.



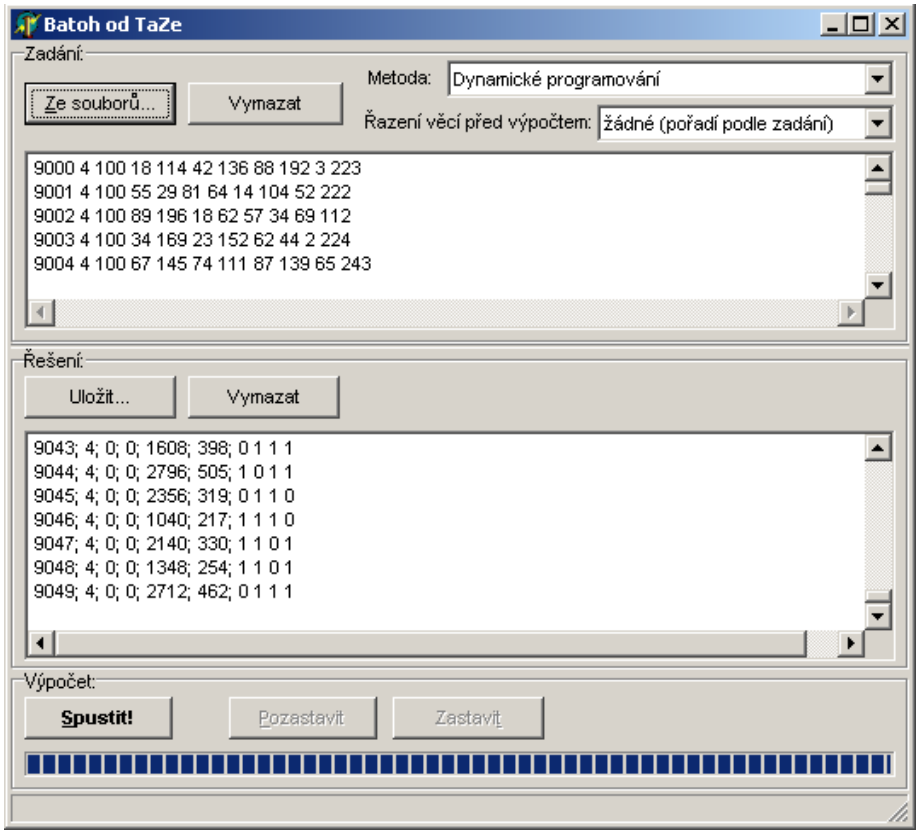
Obr. 4 Vliv náhodné proměnné na stochastický proces

Takto získané optimální řešení použijeme následujícím způsobem: v počátečním stavu p provedeme rozhodnutí q a čekáme do jakého stavu p_1 se vlivem náhodné proměnné r_0 dostaneme, v další etapě zjistíme rozhodnutí q_1 které je funkcí stavu p_1 do kterého jsme se dostali. Vliv hodnot náhodné proměnné v první etapě procesu je znázorněn na Obr. 4. Takto bychom postupovali i v dalších etapách procesu.

2.6 Systémy pro řešení úloh dynamického programování

Tato práce se zabývá vytvořením programového systému pro řešení úloh dynamického programování, proto je vhodné uvést některé programy nebo systémy, které se zabývají řešením úloh dynamického programování.

Jedním z programů je program *Mystik*, který se zabývá řešením úlohy o batohu. Úlohu o batohu řeší nejen pomocí dynamického programování, ale umožňuje vybrat metodu řešení. Program obsahuje kromě metody dynamického programování také metodu hrubé síly (brutal force), metodu větví a mezí (branch and bound) a hladový (greedy) algoritmus. Program byl vytvořen v rámci semestrální práce ve vývojovém prostředí Delphi pomocí programovacího jazyka Pascal. Program umožnuje načítat zadání úlohy, ukládat výsledky a řešit více zadání úlohy zároveň.



Obr. 5 Okno programu Mystik při řešení úlohy o batohu

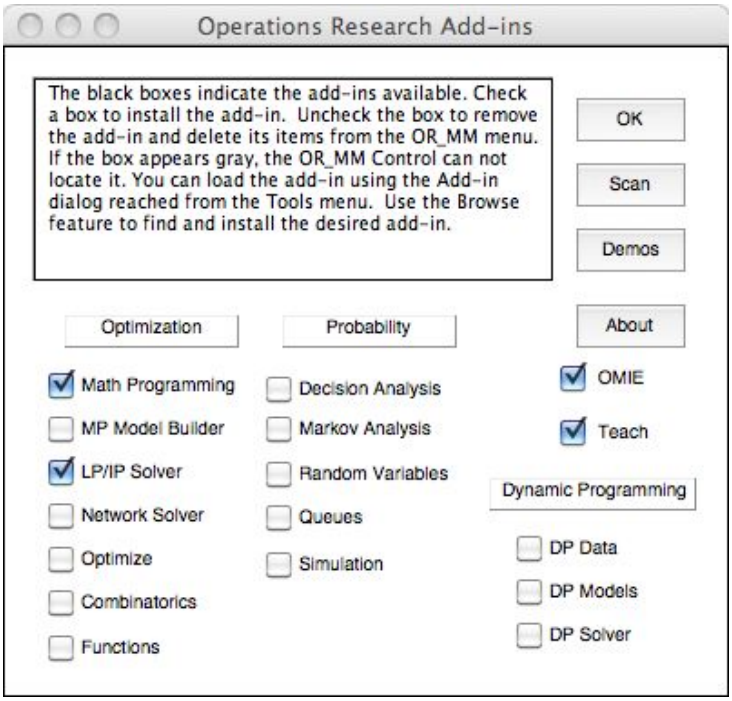
Okno programu s řešenou úlohou o batohu metodou dynamického programování je na Obr. 5. Program byl vytvořen v roce 2003 a autorem programu je Tomáš Zaňka. Více informací o programu *Mystik*, včetně možnosti stažení programu je na stránkách [8].

Dalším program, o kterém stojí za to se zmínit je program *DynamicHanoi*. Jde o program, který řeší úlohu hanojské věže pomocí dynamického pogramování a pro srovnání řeší program úlohu zároveň pomocí rekurze. Program je napsán v jazyce Java a jde o konzolovou aplikaci. Autorem programu je Timothy Rolfe. Více informací o programu, včetně možnosti stažení programu a zdrojových kódů lze nalézt zde [9].

Další program *Knapsack* se zabývá řešením úlohy o batohu. Úlohu o batohu řeší jednak dynamickým programováním, ale také metodou větví a mezí a heuristikou. Hodnota heuristické funkce je vypočtena poměrem cena / hmotnost s testem nejcennější věci. Program byl napsán v programovacím a skriptovacím jazyce Lua a autorem programu je Ondřej Čermák. Zdrojový kód programu je možné stáhnout z těchto stránek [10], kde kromě zdrojového kódu je uvedeno i srovnání jednotlivých algoritmů.

Další zajímavou možností je nástavba nebo zásuvný modul pro operační výzkum pro program Excel. Přesný název zásuvného modulu je *Operations research add-ins*. Zásuvný modul obsahuje

velké množství výpočetních metod pro operační výzkum, jako např.: dynamické programování, ale také lineární, nelineární, celočíselné programování a další. Modul umožňuje řešit úlohy dynamického programování nejen deterministické, ale i stochastické. Umožnuje dokonce i řešit úlohy pomocí Markovových řetězů. Modul přímo obsahuje některé úlohy dynamického programování, jako např.: obnova strojového parku (equipment replacement problem), rozdělení zdrojů (resources problem), investiční problém (investment problem) a některé další méně známé úlohy. Kromě úloh které má modul přímo vestavěné, umožňuje vytvořit zadání vlastní úlohy.



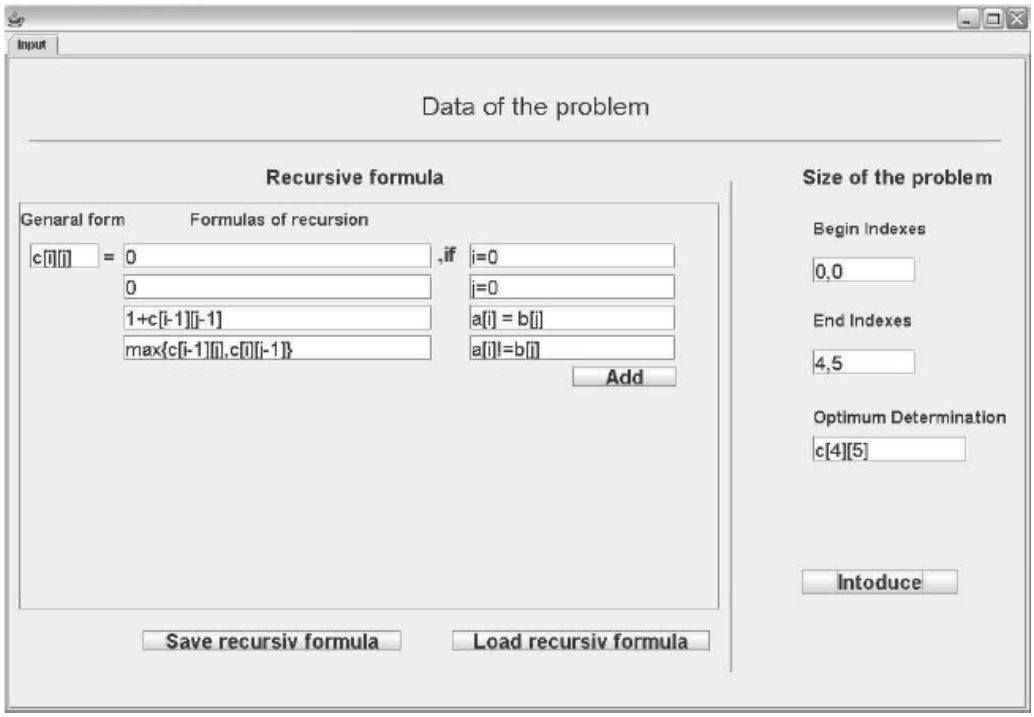
Obr. 6 Uživatelské rozhraní modulu pro operační výzkum [11]

Zásuvný modul je naprogramován pomocí programovacího jazyka Visual Basic a využívá grafického uživatelského rozhraní, které poskytuje Microsoft Excel. Uživatelské rozhraní zásuvného modulu je na Obr. 6. Modul je možné stáhnout na stránkách [11], kde je i podrobný návod na instalaci a používání modulu.

Dalším programem je program *DynProg*, který se zabývá řešením úlohy obchodního cestujícího pomocí dynamického programování. Program byl vytvořen v rámci diplomové práce [13], je naprogramován v jazyce Java a jde o konzolovou aplikaci. Vstupem je matice sousednosti měst a výstupem je optimální cesta ve formě seznamu měst a délka optimální cesty. Autorem programu je Vlastislav Weiner a program je dostupný ke stažení včetně zdrojových kódů na stránkách [12].

Dalším z programů, vytvořených pro řešení úloh dynamického programování, je program *ORToolkit*. Program umožňuje řešit jednak úlohu o batohu, ale i problém obnovy strojového parku, popřípadě doprogramovat úlohu další. Program je vytvořen v prostředí Visual Studio pomocí programovacího jazyku C#. Pro oddělení kódu řešitele od kódu úloh a zajištění tak univerzálnosti řešitele je použita genericita. Program byl vytvořen v roce 2006 a autorem je Andrew Tweddle. Článek popisující vytváření programu včetně návodu na instalaci a možnosti stažení zdrojových kódů, je zde [14].

Jedním z užitečných systémů, zabývajících se řešením úloh dynamického programování, je systém uvedený v práci [15], kde pomocí tohoto systému jsou řešeny problémy průchodu grafem. Vstup systém umožňuje uživateli zadat charakter úlohy rovnicemi a nerovnicemi, a takto zadané úlohy ukládat, případně načítat.



Obr. 7 Vstup umožňující zadat úlohu rovnicemi [15]

Vstup systému je zobrazen na Obr. 7. Jednou z výhod tohoto systému je i možnost zobrazovat a krokovat postup, při kterém je úloha řešena metodou dynamického programování.

Posledním systémem pro řešení úloh dynamického programování, který zde uvedeme, je systém *DP2PN2Solver*. Při vytváření tohoto systému bylo cílem vytvořit obecný systém, který by dokázal řešit obecně jakoukoliv úlohu dynamického programování. Pro reprezentaci úloh dynamického programování používá systém Bellmanovy sítě, které jsou založeny na Petriho sítích. Systém umožňuje řešit pouze diskrétní úlohy. Některé spojité úlohy umožňuje řešit aproximací. Systém dokáže řešit velkou množinu optimalizačních úloh jak deterministických, tak i stochastichkých, dokáže řešit problémy integrálního lineárního programování, problémy s více koncovými stavy, ale i neoptimalizační úlohy, jako např. hanojské věže. Systém byl vytvořen pomocí programovacího jazyka Java a je více popsán v publikaci [16].

3 ÚLOHY DYNAMICKÉHO PROGRAMOVÁNÍ

V této kapitole budou uvedeny některé problémy, které se dají řešit pomocí dynamického programování. Každá úloha bude kromě zadání a specifikace problému obsahovat také sestavení funkcionální rovnice pro řešení dané úlohy. Úlohy, kterými se zde budeme zabývat, můžeme rozdělit do dvou základních skupin na deterministické a stochastické. Úlohy můžeme řešit dynamickým programováním v zásadě dvojím způsobem, a to využitím zpětné nebo dopředné rekurze. Při použití dopředné rekurze postupujeme při výpočtu od první etapy k poslední n-té etapě. Tento způsob výpočtu je snadněji představitelný. Naopak při použití zpětné rekurze postupujeme od konce procesu směrem na začátek, tj. od n-té etapy k první etapě. V případě použití zpětné rekurze zavádíme rozhodovací proměnnou $q_j = x_{n-j}$, a tedy potom budeme provádět nejdříve rozhodnutí pro n-tou etapu procesu a postupovat směrem k první etapě. Bývá uváděno, že využití zpětné rekurze při výpočtu úloh je více efektivní a také bývá častěji používáno v literatuře [5].

3.1 Deterministické úlohy

Největší zastoupení v literatuře věnující se dynamickému programování mají právě diskrétní deterministické úlohy. Úkolem v deterministických úlohách je najít optimální hodnotu účelové funkce a najít optimální strategii, kterou lze optimální hodnoty účelové funkce dosáhnout.

3.1.1 Úloha o batohu

Problém batohu, někdy též známý pod názvem problém loupežníka, v anglické literatuře nejčastěji označovaný jako knapsack problem. Jde o problém, který je zadán následovně: máme batoh o omezené nosnosti a n věcí, kdy každá z věcí má svou cenu a hmotnost. Součet hmotností všech věcí je vyšší než nosnost batohu. Tento požadavek není bezpodmínečně nutný, ale pokud by nebyl splněn tak by se jednalo o triviální problém, kde optimální strategií by bylo vložit do batohu postupně všechny věci. Cílem této úlohy je naplnit batoh tak, aby jednak součet všech hmotností věcí v batohu nepřesáhl nosnost batohu a také aby součet všech cen věcí v batohu byl co možná největší. Přípustná rozhodnutí jsou pouze dvě: buď celou věc dát do batohu nebo ne. Úkolem tedy je maximalizovat účelovou funkce, která je v této úloze aditivní.

$$\sum_{j=1}^{n} c_j x_j \tag{3.1}$$

$$\sum_{j=1}^{n} a_j x_j \leqslant b \tag{3.2}$$

Účelová funkce pro tuto úlohu má tvar (3.1), kde c_j je cena j-té věci, x_j je rozhodnutí zda j-tou věc vzít či nikoliv. Omezující podmínka pro tuto funkci je (3.2), kde a_j je hmotnost j-té věci, b je nosnost batohu. Rozhodnutí x_j mohou nabývat pro kažnou věc pouze hodnot $\{0,1\}$, kde hodnotou 0 rozumíme rozhodnutí j-tou věc nedat do batohu a hodnotou 1 rozhodnutí věc dát do batohu. Dynamické programování formuluje tuto úlohu takto.

$$p_{j+1} = T(p_j, x_j) = p_j - a_j x_j \quad p_1 = b \tag{3.3}$$

$$f_{n-j+1}(p_j) = \max_{x_j \in S_j(p_j)} [c_j x_j + f_{n-j}(p_j - x_j a_j)] \tag{3.4}$$

$$f_1(p_n) = \max_{x_n \in S_n(p_n)} [c_n x_n] \tag{3.5}$$

$$x_j \in S_j(p_j) = \{x \in \{0,1\}, p_j - a_j x \geqslant 0\}$$

Pro řešení této úlohy pomocí dynmického programování je nutné stanovit, co budeme považovat za stav procesu. Stav procesu budeme chápat jako zbývající nosnost batohu a můžeme ho vypočítat pomocí vztahu (3.3), kde v stav procesu v první etapě bude roven nosnosti batohu. Funkcionální rovnice pro řešení tohoto problému jsou (3.4) pro n-tou etapu procesu a omezující okrajová podmínka (3.5) pro první etapu procesu. Tímto jsme převedli řešení n-rozměrného problému batohu na řešení *n* jednorozměrných problémů pomocí funkcionální rovnice. Řešený příklad úlohy o batohu je v podkapitole 6.1. Více informací o úloze o batohu můžeme nalézt v literatuře [2], [5], [7].

3.1.2 Plnění kontejneru

Úloha o plnění kontejneru, v anglické literatuře označovaná jako conteiner loading problem, je zobecněním úlohy o batohu. Úloha je zadána následujícím způsobem. Je dáno n-věcí, kdy každá věc má svou cenu, hmotnost a počet kusů dané věci. Úkolem je naplnit kontejner takovým způsobem, aby celková cena všech věcí vložených do kontejneru byla co největší a zároveň aby celková hmotnost všech věcí vložených do kontejneru nepřevyšovala nosnost kontejneru. Zobecnění úlohy spočívá v tom, že množina přípustných rozhodnutí není pouze dvouprvková, jak tomu bylo u úlohy o batohu, ale nabývá obecně d_j hodnot, kde d_j je počet kusů j-té věci. I přes toto zobecnění můžeme použít účelovou rovnici stejnou jako v úloze o batohu. Účelová rocnice bude mít tvar (3.1) s omezující podmínkou (3.2), kde c_j je cena j-té věci, x_j je počet kusů j-té věci vložených do kontejneru a a_j je hmotnost j-té věci, b je kapacita kontejneru. Dynamickým programováním můžeme tuto úlohu formulovat pomocí transformační rovnice (3.3) a pomocí funkcionálních rovnic (3.4) a (3.5). Rozhodnutí x_j přísluší do množiny přípustných rozhodnutí $S_j(p_j)$, která je definovaná následovně.

$$x_j \in S_j(p_j) = \{x \in \{0,1,\ldots,d_j\},\ p_j - a_j x \geqslant 0\}$$

Řešený příklad úlohy plnění kontejneru je v podkapitole 6.2.

3.1.3 Rozdělení zdrojů

Problém optimálního rozdělení zdrojů nebo též přidělení zdrojů je v anglické literatuře označován názvem resource allocation problem. Problém je formulován následovně. Je dáno omezené množství nějakého zdroje p, kde p je nezáporné celé číslo a zdrojem může být např.: surovina, investice, pracovní síly, stroje atd. Dále je dáno n možných způsobů, jak danou jednotku zdroje použít. Každý způsob z n možných způsobů má účelovou funkci $g(x_j)$, která je funkcí rozhodnutí, jaké množství zdroje danému způsobu přidělit. Cílem této úlohy je rozdělit zdroj takovým způsobem, aby byla maximalizována účelová funkce. Tuto funkci si můžeme představit např. jako zisk ze zdroje, který byl použit daným způsobem.

$$\sum_{j=1}^{n} g_j(x_j) \tag{3.6}$$

$$\sum_{j=1}^{n} x_j \leqslant p \tag{3.7}$$

Celkový zisk z použití zdroje je (3.6), kde $g_j(x_j)$ je hodnota účelové funkce, nebo též ziskové funkce, pro j-tou etapu závisející na rozhodnutí x_j. Často platí, že funkce $g_j(x_j)$ je neklesající a že $g_j(0) = 0$. Podmínka (3.7) stanoví, že nemůže být přiděleno větší množství zdroje než jaké je k dipozici. Rozhodnutí x_j mohou nabývat pouze celých kladných čísel a rozhodnutí $x_j = 0$ chápeme jako j-témuu použití nepřidělit žádný zdroj. Stav ztotožníme s množstvím zdroje, které zatím ještě nebylo rozděleno.

Transformační funkce pro tuto úlohu bude mít tvar (3.8), tzn. že stav procesu p_{j+1} vypočteme jako rozdíl předcházejícího rozhodnutí x_j od předcházejícího stavu p_j a v první etapě se stav procesu

bude rovnat celkovému počtu zdrojů. Úlohu můžeme řešit pomocí funcionálních rovnic (3.9) a (3.10), kde rozhodnutí x_j může nabývat pouze hodnot z množiny přípustných rozhodnutí $S_j(p_j)$.

$$p_{j+1} = p_j - x_j \quad p_1 = p \tag{3.8}$$

$$f_{n-j+1}(p_j) = \max_{x_j \in S_j(p_j)} [g_j(x_j) + f_{n-j}(p_j - x_j)] \tag{3.9}$$

$$f_n(p) = \max_{x_n \in S_n(p_n)} [g_n(x_n)] \tag{3.10}$$

$$x_j \in S_j(p_j) = \{0, 1, \ldots, p_j\} \tag{3.11}$$

Řešený příklad úlohy rozdělení zdrojů je v podkapitole 6.3. Více informací o problému rozdělení zdrojů, včetně řešení ukázkového příkladu lze nalézt v literatuře [3], [4].

3.1.4 Distribuce zdravotních týmů

Úloha distrubuce zdravotních týmů je v anglické literatuře označovaná distributing medical teams to countries. V této úloze je dáno nějaké množství zdravotních týmů a *n* států, kde by se týmy daly využít. Pro každý stát je dána účelová funkce, která je závislá na počtu přidělených zdravotních týmů. Účelová funkce představuje vliv zvýšení zdravotní péče ve státech, kam byly týmy poslány a její hodnota je tisícinásobkem součtu nárůstu délky života všech občanů ve státě. Na tento problém můžeme aplikovat stejný postup jako u obecnější úlohy rozdělení zdrojů. V této úloze se totiž jedná o konkretizaci problému rozdělení zdrojů, kde zdrojem je pracovní síla, přesněji řečeno zdravotní týmy a státy představují možné způsoby použití těchto týmů. Z tohoto důvodu můžeme použít rovnici (3.8) pro transformaci a získání nového stavu a funkcionální rovnice (3.9) a (3.10) pro výpočet úlohy. Kde množima přípustných rozhodnutí bude dána vztahem (3.11). Řešený příklad úlohy distribuce zdravotních týmů je v podkapitole 6.4. Více informací o úloze distribuce zdravotních týmů, včetně postupu jak úlohu řešit, je v publikaci [6].

3.1.5 Vytváření výzkumných týmů

V anglické literatuře se tento problém označuje názvem distributing scientists to research teams. Jde o problém výzkumu, kde je dán zadaný počet vědců *p* a *n* výzkumných týmů, kam je možné vědce přidělit. Pro každý výzkumný tým je dána účelová funkce $g_j(x_j)$, jejíž hodnota je závislá na počtu přidělených vědců do daného týmů. Účelová funkce nabývá hodnot z intervalu (0, 1) a můžeme ji chápat jako pravděpodobnost selhání týmu na daném výzkumu. Ve většině případů bude účelová funkce nerostoucí a $g_j(0) = 1$, tj. pro žádné přidělené vědce bude mít tým pravděpodobnost selhání 100%. Toto nebude platit v případě, že rozdělujeme vědce do týmů, kde už nějací vědci jsou. Předpokladem pro tuto úlohu je, že pravděpodobnosti selhání jednotlivých týmů jsou na sobě vzájemě nezávislé. Úkolem je přidělit vědce týmům takovým způsobem, aby byla minimalizovaná pravděpodobnost selhání výzkumu. I tuto úlohu lze chápat jako konkretizaci úlohy rozdělení zdrojů, ale z důvodu odlišnosti charakteru úlohy, kde účelová funkce nebude aditivní, ale multiplikativní a nepůjde o maximalizaci účelové funkce, ale o minimalizaci, nelze použít pro výpočet rovnice stejné jako v problému rozdělení zdrojů.

$$\prod_{j=1}^{n} g_j(x_j) \tag{3.12}$$

Pravděpodobnost selhání výzkumu je dána vztahem (3.12) s omezující podmínkou (3.7) shodnou jako u úlohy rozdělení zdrojů, kde $g_j(x_j)$ je účelová funkce a úkolem je ji minimalizovat. Stav procesu pro jednotlivé etapy budeme chápat jako počet vědců, kteří ještě nebyli přiděleni žádnému týmu.

$$f_{n-j+1}(p_j) = \min_{x_j \in S_j(p_j)} [g_j(x_j) \cdot f_{n-j}(p_j - x_j)] \tag{3.13}$$

$$f_1(p_n) = \min_{x_n \in S_n(p_n)} [g_n(x_n)] \tag{3.14}$$

Transformační rovnice pro výpočet nového stavu je dána vzorcem (3.8), stejným jako v úloze o rozdělení zdrojů. Funkcionální rovnice pro výpočet optimální hodnoty jsou (3.13) pro (n−j)-tou etapu procesu s okrajovou podmínkou (3.14). Rozhodnutí x_j mohou nabývat pouze hodnot z množiny přípustných rozhodnutí $S_j(p_j)$, která je dána vztahem (3.11). Řešený příklad úlohy vytváření výzkumných týmů je v podkapitole 6.5. Podrobněji popsána je tato úloha včetně postupu řešení v publikaci [6].

3.1.6 Obnova strojového parku

Problém obnovy strojového parku se v anglické literatuře označuje jako equipment replacement model. Jde o problém najít optimální obnovovací strategii, tj. takovou, která maximalizuje celkový zisk ze stroje za požadované období. Stroj je charakterizován nákupní cenou *c* a ziskovou funkcí *n*(*t*). Zisková funkce je závislá na stáří stroje *t* tak, že s rostoucím stářím stroje klesá jeho zisk. To je způsobené tím, že jak stroj stárne, zvyšují se jeho nároky na údržbu, jak finanční tak časové a to zapříčiní nižší zisk. Předpokládáme, že stroj je natolik specifický, že nemá prodejní cenu. V této úloze se jedná o *n* etapový proces, kde *n* je délka období, pro kterou se má stanovit obnovovací strategie. Na začátku každé etapy se činí rozhodnutí, zda stroj ponechat nebo obnovit.

$$\sum_{j=1}^{n} [n(t_j) - c\, x_j] \tag{3.15}$$

Zisk z celého období je dán vzorcem (3.15), kde $n(t_j)$ tvoří část hodnoty ziskové funkce stroje závislé na stáří stroje, *c* je nákupní cena nového stroje a x_j rozhodnutí, zda stroj v j-té etapě obnovit či ne. Rozhodnutí může nabývat pouze hodnot 0 pro ponechání stroje a 1 pro jeho obnovení.

$$t_{j+1} = (1 - x_j)\, t_j + 1 \quad t_1 = t \tag{3.16}$$

$$f_{n-j+1}(t_j) = \max_{x_j \in S_j} [n((1 - x_j)t_j) - c\, x_j + f_{n-j}((1 - x_j)t_j + 1)] \tag{3.17}$$

$$f_1(t_n) = \max_{x_n \in S_n(t_n)} [n((1 - x_n)t_n) - c\, x_n] \tag{3.18}$$

$$x_j \in S_j = \{0,1\} \tag{3.19}$$

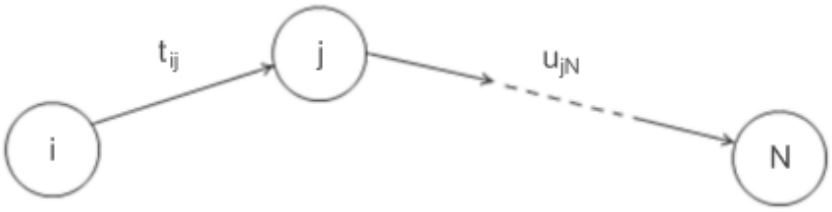
Se stavem procesu ztotožníme stáří stroje a budeme ho počítat transformační funkcí pomocí vztahu (3.16) a stáří stroje v první etapě procesu t_1 se bude rovnat zadanému stáří stroje *t*. Výpočet úlohy provedeme pomocí funkcionálních rovnic (3.17) a (3.18), kde rozhodnutí x_j patří do množiny přípustných rozhodnutí S_j podle vzorce (3.19). Řešený příklad úlohy obnovy strojového parku je v podkapitole 6.6. Více informací o této úloze včetně postupu řešení lze naleznout v literatuře [3], [5].

3.1.7 Průchod sítí

Průchod sítí je problémem hledání nejkratší cesty v síti neboli grafu, v anglické literatuře se tato úloha označuje shortest-route problem, nebo také stagecoach problem. Jelikož se jedná o problém nalezení nejkratší cesty v grafu, budeme zde používat termíny používané v teorii grafů. Máme zadaný graf skládající se z *N* uzlů a jednotlivé uzly můžeme označit čísly od 1 do *N*. Dále je graf zadán hranami, které spojují jednotlivé uzly grafu. Každá hrana má hodnotu $t_{ij} > 0$, která značí vzdálenost

mezi uzly, které hrana spojuje.

Úkolem je najít nejkratší cestu sítí ze zadaného počátečního uzlu do koncového uzlu. Tato úloha je dost obecná. Pro snadnější pochopení si uzly grafu, ve kterém hledáme nejkratší cestu můžeme představit jako města a hrany grafu jako silnice, které města spojují. Potom hodnota t_{ij} může představovat např.: vzdálenost, čas potřebný k překonání vzdálenosti nebo spotřebované pohonné hmoty. Grafem můžeme reprezentovat jakoukoliv úlohu dynamického programování a to tak, že uzly grafu budou představovat stavy procesu a hrany grafu transformace. Tohoto faktu můžeme využít při řešení této úlohy tak, že stav procesu p_j budeme chápat jako uzel grafu a přechod mezi stavy budeme provádět transformací, jak je znázorněno na Obr. 8.



Obr. 8 Schéma řešení úlohy průchod sítí

Rozhodnutí *j* bude ovlivňovat přes kterou hranu se dostaneme do nového uzlu. Označíme si počáteční uzel jako *i* a koncový uzel *N*; účelovou funkci, která představuje nejkratší cestu z uzlu *i* do uzlu *N* si označíme u_{iN}.

$$u_{iN}^{(n+1)} = \min_{j \neq i} [t_{ij} + u_{jN}^{(n)}] \tag{3.20}$$

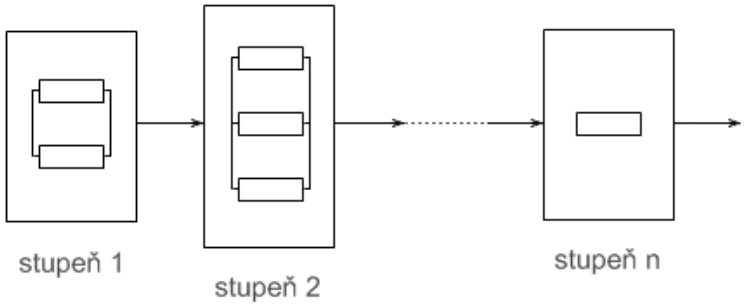
$$u_{iN}^{(1)} = \min_{j \neq i} [t_{ij} + u_{jN}^{(0)}] \tag{3.21}$$

$$u_{iN}^{(0)} = t_{iN} \tag{3.22}$$

Z principu optimality dostáváme funkcionální rovnice (3.20) pro hledání nejkratší cesty z uzlu *i* do uzlu *N* přes nejvýše *n*+1 meziuzlů, kde t_{ij} je vzdálenost mezi uzly *i* a *j*. Volba následujícího uzlu je dána rozhodnutím *j*. Pokud neexistuje přímá spojnice mezi uzly *i* a *j*, pak t_{ij} se rovná nekonečně velké hodnotě. Funkcionální rovnice (3.21) představuje výpočet nejkratší cesty přes nanejvýš jeden meziuzel a rovnice (3.22) nejkratší přímé cesty, tj. přes žádný meziuzel. Řešení příkladu úlohy průchodu sítí je v podkapitole 6.7. Více informací o zmíněném problému lze nalézt v literatuře [3], [5], [6], [7].

3.1.8 Spolehlivost zařízení

Tato úloha se zabývá optimalizací spolehlivosti zařízení. Spolehlivost je jedním z hlavních požadavků kladených na elektronické systémy. Budeme zkoumat elektronický systém ve stupních, kde každému stupni přísluší nějaký elektrický prvek. Prvkem může být např. zesilovač, transformátor nebo jakékoliv jiné zařízení.



Obr. 9 Schéma úlohy spolehlivosti zařízení [3]

Pro zvýšení spolehlivosti zařízení lze v některých stupních zdvojit prvky, tj. zavedeme místo jednoho původního prvku dva, kdy v případě poruchy jednoho ze dvou prvků bude zajištěna funkčnost celého zařízení, jak je znázorněno na Obr. 3. Samozřejmě je možné zavést i více paralelních prvků.

V úloze je zadáno množství nákladů M, které lze využít pro zvýšení spolehlivosti zařízení. Cílem je maximalizovat spolehlivost zařízení při nepřekročení nákladů M. Spolehlivost zařízení budeme chápat jeko pravděpodobnost, že během daného časového období nedojde k poruše zařízení. Dále budeme předpokládat, že spolehlivost můžeme zvýšit pouze zavedením paralelních prvků.

$$\prod_{j=1}^{n} g_j(x_j) \tag{3.23}$$

$$\sum_{j=1}^{n} c_j x_j \leqslant M \tag{3.24}$$

Matematicky lze celkovou spolehlivost popsat pomocí (3.23) za předpokladu, že spolehlivost j-tého stupně nezávisí na spolehlivosti ostatních stupňů. Funkce $g_j(x_j)$ vyjadřuje spolehlivosti j-tého stupně, která závisí na rozhodnutí x_j. Rozhodnutí x_j chápeme jako počet paralelních prvků, které přidělíme j-tému stupni zařízení. Počet všech prvků v j-tém stupni je potom roven $1+x_j$. Pro tuto úlohu lze formulovat omezující podmínku (3.24), kde c_j je cena zařízení v j-tém stupni a M jsou celkové náklady, které lze využít pro maximalizaci spolehlivosti zařízení.

$$p_{j+1} = p_j - c_j x_j \quad p_1 = M \tag{3.25}$$

$$f_{n-j+1}(p_j) = \max_{x_j \in S_j(p_j)} [g_j(x_j) \cdot f_{n-j}(p_j - C_j x_j)] \tag{3.26}$$

$$f_1(p_n) = \max_{x_n \in S_n(p_n)} [g_n(x_n)] \tag{3.27}$$

$$x_j \in S_j(p_j) = \{x \in \{0,1,\ldots,p_j\},\ p_j - c_j x \geqslant 0\}$$

Stav procesu p_j budeme chápat jako zbývající množství nákladů v j-tém stupni procesu a můžeme ho vypočítat pomocí vztahu (3.25), kde hodnota stavu v první etapě procesu je rovna zadanému množství nákladů. Úlohu budeme řešit pomocí funkcionálních rovnic (3.26) a (3.27), kde rozhodnutí x_j může nabývat hodnot z množiny přípustných rozhodnutí $S_j(p_j)$. Řešení příkladu úlohy spolehlivosti zařízení je v podkapitole 6.8. Více informací o této úloze lze nalézt v literatuře [3].

3.1.9 Využití pracovních sil

Úloha se zabývá optimalizací strategie nabírání a propouštění zaměstnanců, v anglické literatuře je označovaná jako workforce size model, nebo scheduling employment levels.

Předpokládáme, že výrobní úkoly na požadované období a z nich vyplývající požadovaný počet zaměstnanců je dopředu znám.

Úkolem úlohy je stanovit optimální počet zaměstnanců pro každé po sobě jdoucí období. Požadovaný počet zaměstnanců potřebných pro splnění úkolů v j-tém období označíme m_j. Pokud by byly nulové náklady na nabírání a propouštění zaměstnanců, tak bychom mohli v daném období jednoduše použít požadovaný počet zaměstnanců. Reálně ale náklady na nabírání a propouštění zaměstnanců jsou natolik zásadní, že může být nevhodné použít přesně požadovaný počet zaměstnanců pro dané období. Předpokládáme, že úkoly v j-tém období nemohou být splněny menším počtem, než je požadovaný počet zaměstnanců. Rozhodovací proměnná x_j vyjadřuje skutečný počet zaměstnanců použitých v j-tém období. Náklady na přijmutí nebo propuštění jednoho zaměstnance jsou rovny c, potom nákady na změnu zaměstnanců v j-tém období jsou dány vztahem

$$c \cdot (x_j - x_{j-1})^2$$

kde x_j je počet zaměstnanců použitých v j-tém období a x_{j-1} je počet zaměstnanců použitých v předchozím období. Ve vztahu předpokládáme, že náklady na přijmutí nebo propuštění zaměstnanců mají kvadratickou závislost [6], ale v některé literatuře se můžeme setkat i s lineární závislostí [5]. Nemění-li se počet zaměstnanců, jsou náklady na změnu zaměstnanců rovny nule. Rozdíl mezi požadovaným a skutečným počtem zaměstnanců způsobuje náklady; jelikož neumožňujeme nižší než požadovaný počet zaměstnanců, tak jsou to náklady na nepracující zaměstnance a jsou vyjádřeny vztahem

$$d \cdot (x_j - m_j)$$

kde d jsou náklady na jednoho nepracujícího zaměstnance. Pokud rozdíl mezi požadovaným a skutečným počtem zaměstnanců je nulový, pak náklady na nepracující zaměstnance jsou rovny nule. V počátečním období je počet zaměstnanců roven m_0.

$$\sum_{j=1}^{n} [c(x_j - x_{j-1})^2 + d(x_j - m_j)] \tag{3.28}$$

Celkové náklady pro všechna období jsou dány vztahem (3.28), kde x_0 je rovno počtu zaměstnanců v počátečním období m_0 a cílem této úlohy je minimalizovat celkové náklady. Stav procesu p_j pro j-tou etapu procesu budeme v této úloze chápat jako vypočtený skutečný počet zaměstnanců pro předchozí období podle vzorce (3.29).

$$p_j = x_{j-1} \quad p_1 = x_0 = m_0 \tag{3.29}$$

$$f_{n-j+1}(p_j) = \min_{x_j \in S_j(m_j)} [c(x_j - p_j)^2 + d(x_j - m_j) + f_{n-j}(x_j)] \tag{3.30}$$

$$f_1(p_n) = \min_{x_n \in S_n(p_n)} [c(x_n - p_n)^2 + d(x_n - m_n)] \tag{3.31}$$

$$x_j \in S_j(m_j) = \{x \in \{m_j, m_{j+1}, \ldots, M\}, M = \max_j(m_j)\}$$

Úlohu budeme řešit pomocí funkcionálních rovnic, odvozených z principu optimality. Pro j-tou etapu dostáváme rovnici (3.30) s omezující podmínkou pro první etapu procesu (3.31). Rozhodnutí x_j mohou nabývat pouze hodnot z množiny přípustných rozhodnutí $S_j(m_j)$, která je závislá na požadovaném počtu zaměstnanců m_j a na maximálním požadovaném počtu zaměstnanců M pro všechna období. Řešení příkladu úlohy využití pracovních sil je v podkapitole 6.9. Více informací o této úloze, včetně postupů řešení lze nalézt v literatuře [3], [5], [6].

3.2 Stochastické úlohy

Ve stochastických procesech hrají důležitou roli náhodné proměnné. Úkolem stochastických úloh je podobně jako deterministických vypočítat optimální hodnotu účelové funkce, která je nejčastěji ve formě očekávané hodnoty, a také zjištění optimální strategie optimalizující hodnotu účelové funkce. Na rozdíl od deterministických úloh není optimální strategie jednoznačně určena, ale závisí na působení náhodných proměnných.

3.2.1 Sklad náhradních dílů

Úloha se zabývá optimalizací skladu náhradních dílů. Účelová funkce této úlohy má stochastický charakter, ale někdy bývá zařazována mezi deterministické úlohy, jelikož tvar transformační funkce je deterministický, tj. nezávisí na náhodných faktorech. V úloze je dán skladovací prostor o velikosti d prostorových jednotek. Ve skladu je potřeba skladovat n druhů náhradních dílů, kde každý náhradní díl zabírá a_j prostorových jednotek. Předpokládejme, že poptávka v_j po j-tém dílu je Poissonova náhodná proměnná s parametrem n_j, což znamená, že očekávaná hodnota v_j je rovna hodnotě n_j:

$$E\,v_j = n_j$$

Každý náhradní díl má přiřazenu hodnotu π_j, což značí ztrátu za náhradní díl pokud není na skladě. Úkolem je zjistit počet kusů jednotlivých náhradních dílů, které mají být skladovány tak, aby byla minimalizována očekávaná hodnota ztráty za díly, které nejsou na skladě.

$$\sum_{j=1}^{n} \pi_j A_j(x_j, n_j) \tag{3.32}$$

$$\sum_{j=1}^{n} a_j x_j \leqslant d \tag{3.33}$$

Pro výpočet očekávané hodnoty ztrát za chybějící díly lze použít vzorec (3.32), kde $A_j(x_j, n_j)$ je očekávaná hodnota neuspokojenosti poptávky po j-tém druhu náhradního dílu. Omezující podmínka (3.33), vyjadřuje vliv omezení skladovacího prostoru.

$$\begin{aligned} &A_j(x_j, n_j) = n_j P(x_j - 1, n_j) - x_j P(x_j, n_j) \quad pro \quad x_j \geqslant 1 \\ &A_j(x_j, n_j) = n_j \quad pro \quad x_j = 0 \end{aligned} \tag{3.34}$$

Očekávanou hodnotu neuspokojenosti poptávky $A_j(x_j, n_j)$ můžeme vypočítat pomocí vzorce (3.34), kde $P(x_j, n_j)$ je pravděpodobnost, že Poissonova náhodná proměnná nabývá hodnoty rovné x_j nebo větší. Jedná se o funkci Poissonova kumulativního rozdělení [3].

$$p_{j+1} = p_j - a_j x_j \quad p_1 = d \tag{3.35}$$

$$f_{n-j+1}(p_j) = \min_{x_j \in S_j(p_j)} [\pi_j A_j(x_j, n_j) + f_{n-j}(p_j - a_j x_j)] \tag{3.36}$$

$$f_1(p_n) = \min_{x_n \in S_n(p_n)} [\pi_n A_n(x_n, n_n)] \tag{3.37}$$

$$x_j \in S_j(p_j) = \{x \in \{0, 1, \ldots, p_j\},\ p_j - a_j x \geqslant 0\}$$

Stav procesu p_j budeme chápat jako zbývající kapacitu skladu v j-té etapě a nový stav p_{j+1}

získáme transformací pomocí rovnice (3.35). Z principu optimality dostaneme funkcionální rovnice (3.36) a (3.37), kde rozhodnutí x_j nabývá hodnot z množiny přípustných rozhodnutí $S_j(p_j)$. Řešení příkladu úlohy skladu náhradních dílů je v podkapitole 6.10. Další informace včetně podrobného postupu řešení lze nalézt v literatuře [3], [4].

3.2.2 Vyhrávání v Las Vegas

Úloha se zabývá hledáním strategie, která maximalizuje pravděpodobnost na výhru v populární hře Las Vegas. V anglické literatuře je tato úloha označena názvem winning in Las Vegas. Je dán počet žetonů na začátku procesu C_1 a požadovaný počet žetonů na konci procesu C_2. Dále je dán počet etap procesu n, tj. počet možných sázek, pravděpodobnost výhry jedné sázky P_W a pravděpodobnost prohry P_L. Pravděpodobnost prohry je dána následujícím vztahem.

$$P_L = 1 - P_W$$

Výhra sázky vrací dvojnásobné množství vsazených žetonů. Celková výhra nastane tehdy, pokud se podaří daným počtem sázek dosáhnout nejméně požadovaného počtu žetonů C_2. Cílem úlohy je maximalizovat pravděpodobnost celkové výhry.

$$f_{n-j+1}(p_j) = P_W f_{n-j}(p_j + x_j) + P_L f_{n-j}(p_j - x_j) \tag{3.38}$$

$$f_0(p_{n+1}) = 0 \quad pro \quad p_{n+1} < C_2, \quad f_0(p_{n+1}) = 1 \quad pro \quad p_{n+1} \geq C_2 \tag{3.39}$$

Pravděpodobnost celkové výhry je dána vztahem (3.38), kde pravděpodobnost po poslední etapě procesu $f_0(p_{n+1})$ je definována vztahem (3.39), tj. je rovna jedné, pokud bylo dosaženo celkové výhry a je rovna nule, pokud nebylo dosaženo výhry. Stav procesu p_j chápeme jako počet žetonů v j-té etapě procesu a rozhodnutí x_j bude rovno počtu vsazených žetonů j-té etapě procesu.

$$p_{j+1} = p_j + r_j x_j - (1 - r_j) x_j \quad p_1 = C_1 \tag{3.40}$$

$$f_{n-j+1}(p_j) = \max_{x_j \in S_j(p_j)} [(r_j P_W + (1 - r_j) P_L) f_{n-j}(p_j + r_j x_j - (1 - r_j) x_j)] \tag{3.41}$$

$$x_j \in S_j(p_j) = \{0, 1, \ldots, p_j\}$$

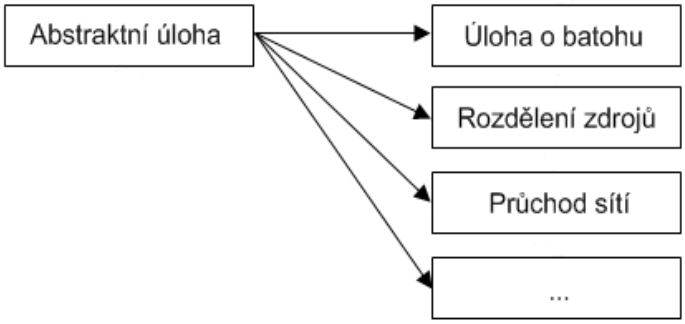
Stav procesu v další etapě p_{j+1} získáme transformací pomocí rovnice (3.40), kde r_j je náhodná proměnná a může nabývat hodnot $r_j = 0$ pro prohru sázky a $r_j = 1$ pro výhru sázky. Úlohu můžeme řešit pomocí funkcionální rovnice (3.41) s omezující podmínkou pro poslední etapu procesu (3.39), kde rozhodovací proměnné x_j mohou nabývat pouze hodnot z množiny přípustných rozhodnutí $S_j(p_j)$. Řešení příkladu úlohy vyhrávání v Las Vegas je v podkapitole 6.11. Více informací a postup řešení této úlohy lze nalézt v literatuře [6].

4 NÁVRH PROGRAMOVÉHO SYSTÉMU

Cílem práce je vytvořit programový systém pro řešení vybraných úloh dynamického programování. Programový systém by měl sloužit při výuce předmětu Optimalizace. Při návrhu systému je třeba dbát na to, aby část programu, která bude řešit úlohy dynamického programování, nebyla závislá na konkrétní úloze, ale byla použitelná pro všechny vybrané úlohy. Část programu, která řeší úlohy dynamického programování, budeme označovat zkráceně řešitel. V této kapitole se budeme zabývat návrhem programového systému, konkrétně návrhem třídy reprezentující řešitele a tříd, reprezentujících úlohy dynamického programování.

Obecně lze pro oddělení dat od režijní struktury v objektově orientovaném programování použít několik způsobů, jako např.: dědění z abstraktní třídy, genericitu, implementaci rozhraní (interfaces), nebo využití delegátů.

V této práci byl pro oddělení kódu řešitele od kódu úloh a zajištění tak univerzálnosti řešitele použit přístup, založený na abstraktní třídě a dědičnosti. Podstata tohoto přístupu spočívá ve vytvoření abstraktní třídy reprezentující abstraktní úlohu, která bude zobecněním konkrétní úlohy a bude obsahovat metody společné pro všechny úlohy. Schéma abstraktní úlohy je znázorněno na Obr. 10.



Obr. 10 Schéma zavedení abstraktní úlohy

Aby tímto postupem došlo k oddělení kódu úloh od řešitele, tak každá úloha, která má být řešena pomocí řešitele, musí být potomkem abstraktní úlohy a přepisovat všechny metody, které abstraktní úloha obsahuje. Řešitel potom pracuje s metodami v konkrétní úloze jako by pracoval s abstraktní úlohou.

4.1 Návrh třídy abstraktní úlohy

Třída reprezentující abstraktní úlohu musí obsahovat metody potřebné pro výpočet obecně jakékoliv úlohy, tj. metody, které bude potřebovat řešitel pro výpočet dané úlohy. Tyto metody obsažené v abstraktní třídy budou také abstraktní. Konkrétně to jsou tyto metody.

- Funkce pro výpočet optimální hodnoty účelové funkce pro n-tou etapu procesu
- Funkce pro výpočet optimální hodnoty účelové funkce pro první etapu procesu
- Funkce pro transformaci a získání nového stavu
- Funkce, která vrací počet přípustných rozhodnutí
- Funkce, která vrací přípustné rozhodnutí

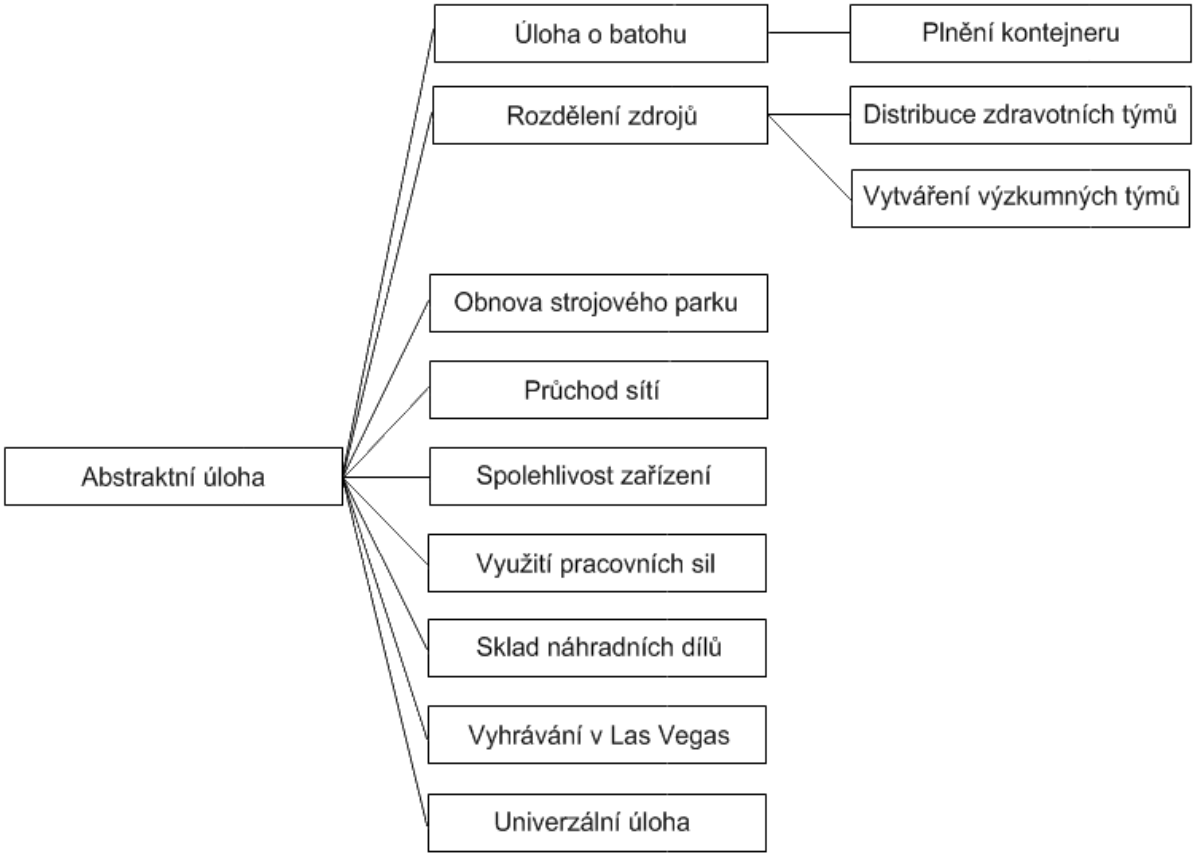
Abstraktní metoda obsažená v abstraktní třídě má definovanou pouze hlavičku metody, tj. vstupní proměnné a jejich typ, popřípadě i typ výstupní proměnné, pokud se jedná o funkci. Pokud by šlo o proceduru, tak by typ výstupní proměnné nebyl uveden. Konkrétní úlohy jsou potomkem abstraktní úlohy a obsahují a přepisují metody obsažené v abstraktní úloze. Konkrétní úloha obsahuje

kromě hlavičky metody, která je shodná s hlavičkou uvedenou v abstraktní úloze, také tělo metody.

Kromě metod shodných s abstraktní úlohou, které konkrétní úloha musí obsahovat, může konkrétní úloha obsahovat libovolné množství pomocných metod a proměnných, potřebných pro vytvoření třídy konkrétní úlohy. Instance abstraktní úlohy nebude nikdy vytvořena, protože neodpovídá žádné konkrétní úloze.

4.2 Návrh tříd konkrétních úloh

Část návrhu konkrétních úloh byla uvedena v rámci kapitoly 3. Přesněji se jedná o část návrhu zabývající se tvarem funkcí pro výpočet dané úlohy. Třídy konkrétních úloh jsou navrženy tak, že každá třída konkrétní úlohy musí být potomkem třídy abstraktní úlohy a musí přepisovat metody v abstraktní úloze. V některých úlohách je vhodné, aby třída reprezentující danou úlohu nebyla pouze potomkem abstraktní úlohy, ale byla také potomkem některé další úlohy. Například úloha plnění kontejneru je jistou obdobou úlohy o batohu, proto je vhodné, aby úloha plnění kontejneru byla potomkem úlohy o batohu. Struktura dědičnosti všech úloh dynamického programování zavedených v programovém systému je zobrazena na Obr. 11.



Obr. 11 Návrh struktury dědičnosti úloh dynamického programování

Z Obr. 11 je například vidět, že úlohy distribuce zdravotních týmů a vytváření výzkumných týmů jsou potomkem úlohy rozdělení zdrojů.

4.3 Návrh třídy řešitele

Třída řešitele by měla být navržena tak, aby obsahovala všechny metody a proměnné spojené s řešením úlohy dynamického programování. Hlavní věcí, kterou by měla třída reprezentující řešitele obsahovat, je metoda pro výpočet zadané úlohy. Dále by měla obsahovat proměnné pro vypočtené hodnoty účelových funkcí podproblému, pomocí kterých bude vypočtena zadaná úloha tabulkovou metodou. Kromě proměnných, obsahujících vypočtené hodnoty účelových funkcí, bude nutné zavést proměnné pro vypočtená rozhodnutí a pro vypočtené hodnoty stavů. Poslední proměnnou, kterou by

měla třída řešitele obsahovat, je proměnná typu abstraktní úlohy, která bude obsahovat konkrétní řešenou úlohu.

4.4 Návrh třídy univerzální úlohy

Při návrhu byla kromě konkrétních tříd, kdy každá třída reprezentuje jednu konkrétní úlohu, navržena i tzv. univerzální úloha. Jedná se o úlohu, která neodpovídá žádné konkrétní úloze. Univerzální úloha umožňuje zadat charakter problému pomocí rovnic; toho je dosaženo tak, že univerzální úloha nemá pevně zavedené funkce pro řešení úlohy, ale tyto funkce jsou v rámci zadání zadány uživatelem. Rovnice pro řešení univerzální úlohy jsou uživatelem zadány ve formě textových řetězců, což není tolik efektivní jako kdyby tyto rovnice byly napevno obsažené v kódu úlohy, ale nadruhou stranu se jedná o vhodnou pomůcku při výuce. Konkrétně půjde o tyto funkce.

- Funkce pro výpočet optimální hodnoty účelové funkce pro n-tou etapu procesu
- Funkce pro výpočet optimální hodnoty účelové funkce pro první etapu procesu
- Funkce pro transformaci a získání nového stavu
- Funkce, která reprezentuje omezující podmínku

Univerzální úloha je navržena tak, že bude potomkem abstraktní úlohy, ale tvar zmíněných funkcí bude závislý na funkcích, které budou zadány uživatelem.

Největším problémem při implementaci univerzální úlohy bude najít způsob, jak funkci zadanou textovým řetězcem převést tak, aby ji byl schopen programový systém správně vyhodnotit.

5 IMPLEMENTACE PROGRAMOVÉHO SYSTÉMU

V této kapitole bude popisována tvorba programového systému. Programový systém byl vytvořen ve vývojovém prostředí Visual Studio 2010 pomocí programovacího jazyku C#. Při tvorbě programového systému byla snaha vytvořit systém co nejobecněji a využívat při tom možnosti objektově orientovaného programování.

5.1 Reprezentace tříd

Nejdříve je vhodné se zmínit o tom, jak byly v proramovém systému implementovány jednotlivé třídy. Třídy se v programovacím jazyce C# označují klíčovým slovem *class*, za ním je uveden název třídy a ve složených závorkách je kód dané třídy.

5.1.1 Třída abstraktní úlohy

Třída abstraktní úlohy byla zavedena z důvodu oddělení kódu řešitele od kódu úloh, více v kapitole 4 a podkapitole 4.1. Při zavedení abstraktní třídy se používá klíčové slovo *abstract* před názvem třídy.

```
abstract class BasicTask
{
    public abstract bool ColumnDrive();

    public abstract int JPosition(int j);

    public abstract int xFirstPosition(int i, int j, out int outI);

    public abstract int fiOptimalPosition(int i);

    public abstract double Function(int j, int x, int i, double[] prev);

    public abstract double Function(int j, int x, int i);

    public abstract int MaxMin();

    public abstract bool MaxMin(int j, int x, int i, double[] prev, double
    maxMin, out double maxMinResult);

    public abstract int Transform(int j, int x, int p, int i, out int
    tranJ);

    public abstract int RetDecision(int j, int x);

    public abstract int RetNumOfDecision(int j);

    public abstract int RetNumOfRandomVar(int j, int x);

    public abstract void CloseTask();
}
```

Obr. 12 Abstraktní třídy BasicTask

Třída *BasicTask* reprezentující abstraktní úlohu je na Obr. 12. Oproti návrhu bylo nutné implementovat některé další metody. Metody, které abstraktní třída obsahuje, jsou abstraktní a označují se klíčovým slovem *abstract*. Dále se před jméno metody zapisuje zda je metoda přístupná i mimo svou třídu. Pokud je metoda přístupná i mimo svou třídu, používá se klíčové slovo *public*, které se zapisuje před jméno metody. Pro metodu přístupnou pouze ze své třídy se používá klíčové slovo *private*, to lze v programovacím jazyce C# vynechat, protože pokud metoda nemá definováno zda je přístupná i mimo svou třídy, tak ji kompilátor uvažuje za privátní, tj. že je přístupná pouze z vlastní třídy. Pro metodu, která má být přístupná mimo svou třídu, ale pouze v rámci potomků této třídy se používá klíčové slovo *internal*.

Funkce *ColumnDrive* vrací informaci o tom, jestli má úloha řešená tabulkovou metodou být řešena po sloupcích, potom vrací hodnotu *true*, nebo po řádcích a v tom případě vrací hodnotu *false*.

Funkce *JPosition* vrací hodnotu etapy procesu, při použití zpětné rekurze vrací hodnotu *počet etap* - *j* - 1, při použití dopředné rekurze vrací hodnotu *j*.

Funkce *xFirstPosition* vrací pozici řádku a sloupce optimálního rozhodnutí *x* pro první etapu procesu. Funkce *fiOptimalPosition* vrací pozici sloupce s optimální hodnotou účelové funkce.

U funkce *Function* je využito přetěžování, což je princip objektově orientovaného programování, který umožňuje mít v kódu více funkcí se stejným názvem. Kompilátor pozná o kterou funkci se jedná podle toho, s jakými vstupními parametry je funkce zavolána. *Function* vrací hodnotu účelové funkce pro j-tou etapu procesu pokud je zavolána se čtyřmi vstupními parametry. Pokud je zavolána se třemi vstupními parametry, vrací hodnotu účelové funkce pro první etapu procesu.

Další metodou, která je opět přetěžovaná, je funkce *MaxMin.* Tato funkce pokud je zavolána se vstupními parametry vrací hodnotu optimální účelové funkce pro j-tou etapu procesu. Bez vstupních parametrů vrací inicializační hodnotu proměnné reprezentující dosavadní optimální hodnotu účelové funkce pro maximalizaci nebo minimalizaci. Inicializační hodnota proměnné nabývá hodnoty co nejmenší pro maximalizaci a co největší pro minimalizaci. V kódu je co nejmenší hodnota reprezentována hodnotou *int.MinValue*, což je nejnižší číslo, které může být reprezentováno celočíselným typem integer. Obdobně největší hodnota je reprezentována hodnotou *int.MaxValue*.

Funkce *Transform* provádí transformaci a vrací hodnotu nového stavu. Funkce *RetNumOfDecision* vrací hodnotu počtu přípustných rozhodnutí. *RetDecision* vrací hodnotu přípustného rozhodnutí. Funkce *RetNumOfRandomVar* vrací počet náhodných proměnných, u deterministických procesů vrací hodnotu jedné, u stochastických procesů vrací hodnotu větší nebo rovnu jedné.

Poslední metoda obsažená v abstraktní úloze je procedura *CloseTask*. Procedura je poslední metoda která se zavolá při řešení úlohy a slouží pro ukončení proměnných, které je potřeba ukončit.

5.1.2 Třída úlohy o batohu

Pro ukázku zde bude uvedena třída reprezentující úlohu o batohu včetně některých metod používaných pro výpočet úlohy o batobu.

```
class Bag : BasicTask
{
    internal double[] price;
    internal int[] weight;
    internal int capacity;

    public Bag(double[] price, int[] weight, int capacity)
    {
        this.price = price;
        this.weight = weight;
        this.capacity = capacity;
    }
```

Obr. 13 Proměnné a konstruktor třídy Bag reprezentující úlohu o batohu

Na Obr. 13 je zobrazena část kódu třídy *Bag*, která reprezentuje úlohu o batohu. Přesněji jde o zavedení proměnných a konstruktor.

Proměnné potřebné v úloze o batohu jsou cena pro každou věc, hmotnost pro každou věc a kapacita batohu. Cena věcí je reprezentována polem reálných čísel *price* a hmotnost věcí polem celých čísel *weight*. Kapacita batohu je obsažena v celočíselné proměnné *capacity*. Konstruktor třídy má tři vstupní parametry a jeho úkolem je inicializovat lokální proměnné.

```
    public override double Function(int j, int x, int i, double[] f)
    {
        if (Weight[j] * x > i) { return MaxMin(); }
        else { return Price[j] * x + f[0]; }
    }
```

Obr. 14 Metoda Function pro výpočet hodnoty účelové funkce

Metoda *Function* obsažená v třídě *Bag* je na Obr. 14 a počítá hodnotu účelové funkce pro j-tou etapu. Metoda obsahuje omezení pokud je rozhodnutí $x = 1$, tj. vzít věc a hmotnost věci je vyšší než kapacita batohu, poté vrací návratovou hodnotu funkce *MaxMin*. Návratová hodnota funkce *MaxMin*

je *int.MinValue*, jelikož se jedná o maximalizační úlohu. V ostatních případech vrací součet ceny věci násobené rozhodnutím *x* a hodnoty optimální účelové funkce pro předchozí etapu procesu *f*.

```
public override bool MaxMin(int j, int x, int i, double[] prev, double  maxMin,
  out double maxMinResult)
{
    if ((maxMinResult = Function(j, x, i, prev)) > maxMin)
    { return true; }
    else { return false; }
}
```

Obr. 15 Metoda MaxMin pro postupný výpočet hodnoty optimální účelové funkce

Metoda *MaxMin* třídy *Bag* je zobrazená na Obr. 15 a slouží pro postupný výpočet hodnoty optimální účelové funkce pro j-tou etapu procesu. V úloze o batohu jde o maximalizaci účelové funkce. Funkce vrací logickou hodnotu *true* pokud hodnota účelové funkce vyšší než doposud nalezené maximum, v ostatních případech vrací *false*. Funkce používá vstupně výstupní parametr *maxMinResult* do kterého se přiřadí hodnota účelové funkce.

```
public override int Transform(int j, int x, int p, int i, out int tranJ)
{
    tranJ = j + 1;
    return (p - weight[j] * x);
}
```

Obr. 16 Metoda Tramsform pro transformaci stavu

Metoda pro transformaci a získání hodnoty nového stavu je na Obr. 16. Výstupní hodnota funkce je hodnota nového stavu. V úloze o batohu je hodnota stavu reprezentována zbývající kapacitou batohu. Hodnoty nového stavu se spočítá jako rozdíl hodnoty stávajícího stavu *p* a hmotnosti věci *a* násobené rozhodnutím *x*.

```
public override int RetNumOfDecision(int j)
{
    if (capacity / weight[j] >= 1) { return 2; }
    else { return 1; }
}
```

Obr. 17 Metoda RetNumOfDecision pro získání počtu přípustných rozhodnutí

Metoda pro zjištění počtu přípustných rozhodnutí je zobrazena na Obr. 17. Počet přípustných rozhodnutí je roven dvěma pokud je kapacita batohu větší než hmotnost věci, v opačném případě je počet přípustných rozhodnutí roven jedné.

5.1.3 Třída řešitele

Třída řešitele reprezentuje řešitel úloh dynamického programování. Třída obsahuje některé proměnné potřebné pro řešení úloh dynamického programování a jednu metodu, pomocí které jsou řešeny úlohy dynamického programování.

```
class Solver
    {
        internal BasicTask task;
        internal double[,] fi;
        internal int[,] x;
        internal double[] fiOpt;
        internal int[,] xOpt;
        internal int[,] pOpt;
        internal int lenght;
        internal int height;
```

Obr. 18 Proměnné třídy Solver reprezentující řešitel

Proměnné zavedené ve třídě *Solver* jsou zobrazeny na Obr. 18. Proměnná *task* obsahuje úlohu řešenou ve třídě řešitele, proměnná je typu abstraktní úlohy *BasicTask*. Klíčové slovo *internal* před proměnnými zajišťuje to, že k zavedeným proměnným lze přistupovat pouze z dané třídy řešitele a případně z třídy, která by byla potomkem třídy řešitele. Proměnná *fi*, která je typu dvourozměrného pole reálných čísel, je zavedena proto, aby obsahovala optimální hodnoty účelových funkcí pro všechny etapy procesu a pro všechny hodnoty stavu úlohy. Proměnná *x* je typu dvourozměrného pole celých čísel a reprezentuje rozhodnutí pro každou hodnotu stavu úlohy v každé etapě procesu. Proměnná *fiOpt* je typu jednorozměrného pole reálných čísel a reprezentuje výslednou optimální hodnotu účelové funkce pro hodoty stavu úlohy. Proměnná *xOpt* typu dvourozměrné pole celých čísel reprezentuje optimální strategii úlohy pro každý počáteční stav. Proměnná *pOpt* typu dvourozměrné pole celých čísel reprezentuje hodnoty stavu úlohy vypočtené během výpočtu optimální strategie úlohy pro každý počáteční stav. Proměnná *lenght* je celočíselného typu a reprezentuje počet počátečních stavů v úloze. Proměnná *height* je opět celočíselná a obsahuje počet etap úlohy.

```
public double[] FiOptimal  { get { return fiOpt; } }

public int[,] POtp  { get { return pOpt; } }

public int[,] XOptimal  { get { return xOpt; } }
```

Obr. 19 Přístupové metody třídy řešitele

Přístupové metody anglicky označované properties třídy řešitele jsou zobrazeny na Obr. 19. Přístupové metody slouží k umožnění přístupu k proměnným. Klíčové slovo *get* pro čtení z proměnné a klíčové slovo *set* pro zápis. Přístupové metody jsou v třídě řešitele použity pro umožnění přístupu k proměnným: *fiOpt* obsahující výsledné optimální hodnoty účelových funkcí, *pOpt* obsahující hodnoty stavů při výpočtu optimální strategie a *xOpt* obsahující optimální strategie úlohy.

```
public Solver(BasicTask task, int lenght, int height)
{
    this.task = task;
    fi = new double[lenght + 1, height];
    x = new int[lenght + 1, height];
    fiOpt = new double[lenght + 1];
    xOpt = new int[lenght + 1, height];
    pOpt = new int[lenght + 1, height + 1];
    this.lenght = lenght;
    this.height = height;
}
```

Obr. 20 Konstruktor třídy řešitele

Na Obr. 20 je zobrazen konstruktor třídy řešitele. Vstupní proměnné konstruktoru jsou: *task* obsahující úlohu, která se má řešit pomocí řešitele, *lenght* obsahující počet počátečních stavů úlohy a *height* obsahující počet etap úlohy. Hlavním úkolem konstruktoru je inicializace proměnných.

```
public void Solve()
```

Obr. 21 Procedura Solve pro řešení úlohy dynamického programování

Samotné řešení úlohy dynamického programování pomocí třídy řešitele se provádí v metodě *Solve*. Jelikož je metoda označena klíčovým slovem *void* jedná se o proceduru. Úkolem procedury je v rámci výpočtu naplnit proměnné třídy *Solver*. Z proměnné *fiOpt* lze po spuštění procedury *Solve* zjistit optimální hodnotu účelové funkce zadané úlohy, z proměnné *xOpt* optimální rozhodovací strategii a z proměnné *pOpt* hodnoty stavů, vypočtené v závislosti na optimálních rozhodnutích.

5.2 Třída univerzální úlohy

Třída univerzální úlohy reprezentuje úlohu, kde charakter úlohy je zadán uživatelem rovnicemi. Proměnné třídy *UniversalTask* reprezentující univerzální úlohu jsou na Obr. 22.

```
class UniversalTask : BasicTask
{
    int height;
    int[] decision;
    bool maximalize;
    bool additive;

    // funkce zadane uzivatelem
    string funFormula;          // funkcionalni rovnice pro j-tou etapu
    string funFormula1;         // funkcionalni rovnice pro 1-ni etapu
    string tranFormula;         // transformacni rovnice
    string limFormula;          // omezujici podminka

    // hodnoty zadanych promenych
    ArrayList arrayVariable = new ArrayList();
    ArrayList arrayValue = new ArrayList();
    double[,] doubleVal;
    int[,] intVal;
```

Obr. 22 Proměnné třídy univerzální úlohy

Celočíselná proměnná *height* obsahuje počet etap úlohy. Proměnná *decision* je typu jednorozměného pole a reprezentuje rozhodovací proměnné. Charakter úlohy je reprezentován logickými proměnnými *maximalize* a *additive*. Proměnná *maximalize* nabývá hodnoty *true*, jestliže se jedná o maximalizační úlohu a proměnná *additive* nabývá hodnoty *true*, pokud je účelová funkce aditivní. Hlavní proměnné této třídy jsou *funFormula* obsahující rovnici zadanou uživatelem pro výpočet hodnoty funkcionální rovnice pro j-tou etapu, *funFormula1* obsahující rovnici pro výpočet hodnoty funkcionální rovnice pro první etapu, *tranFormula* obsahující rovnici pro výpočet transformace a hodnoty nového stavu a rovnice *limFormula* obsahující omezující podmínku. Další proměnné třídy *UniversalTask* reprezentují proměnné zavedené uživatelem, které je možno použít v při tvorbě rovnic. Jsou to proměnné *arrayVariable* obsahující seznam názvů proměnných a *arrayValue* obsahující odpovídající seznam proměnných. Dále je možné zavést dvě proměnné typu dvourozměrného pole. Proměnnou *doubleVal* reprezentující dvourozměrné pole reálných čísel a *intVal* reprezentující dvourozměrné pole celých čísel.

Všechny zde uvedené proměnné jsou inicializovány a naplněny hodnotami v rámci konstruktoru. Kód konstruktoru je uveden na Obr. 23.

```
public UniversalTask(double[,] doubleVal, int[,] intVal, int[] decision,
    int height, string[] varTag, int[] varValue,
    string[] formula, bool max, bool add)
{
    this.doubleVal = doubleVal;
    this.intVal = intVal;
    this.decision = decision;
    this.height = height;
    this.additive = add;
    this.maximalize = max;
    for (int i = 0; i <= varValue.Length - 1; i++)
    {
        arrayVariable.Add(varTag[i]);
        arrayValue.Add(varValue[i]);
    }
    funFormula = formula[0];
    funFormula1 = formula[1];
    tranFormula = formula[2];
    limFormula = formula[3];
}
```

Obr. 23 Konstruktor třídy univerzální úlohy

Na ukázku jsou zde uvedeny kódy metod pro výpočet transformace a pro výpočet hodnoty

funkcionální rovnice. Obě metody využívají funkci *retFunctionValue* pro zjištění výsledné hodnoty rovnice zadané uživatelem.

```
public override int Transform(int j, int x, int p, int i, out int tranJ)
{
    tranJ = j + 1;
    return (int)retFunctionValue(tranFormula, "JS", j, x, i, p, 0);
}
```

Obr. 24 Metoda pro transformaci stavu

Metoda pro výpočet transformace je zobrazena na Obr. 24. Metoda *Transform* počítá hodnotu transformace pomocí rovnice *tranFormula* zadané uživatelem. Pro výpočet výsledné hodnoty rovnice metoda využívá funkci *retFunkctionValue*.

```
public override double Limitation(int j, int x, int i, double f)
{
    if (retFunctionValue(limFormula, "JS", j, x, i, 0, 0) == 0)
    { return MaxMin(); }
    return retFunctionValue(funFormula, "JS", j, x, i, 0, f);
}
```

Obr. 25 Metoda pro výpočet hodnoty funkcionální rovnice

Metoda pro výpočet hodnoty funkcionální rovnice je zobrazena na Obr. 25. Metoda *Limitation* vrací hodnotu funkcionální rovnice, pokud je splněna podmínka *limFormula* zadaná uživatelem. Pokud je podmínka splněna metoda vrací hodnotu pro j-tou etapu úlohy. Pro zjištění výsledné hodnoty rovnic zadaných uživatelem používá opět funkci *retFunctionValue*.

```
double retFunctionValue(string numFormula, string parse, int j, int x,
 int i, int p, double f)
```

Obr. 26 Funkce retFunctionValue pro výpočet hodnot rovnic zadaných uživatelem

Funkce pro výpočet výsledných hodnot rovnic zadaných uživatelem je znázorněna na Obr. 26. Parametr funkce *numFormula* obsahuje rovnici zadanou uživatelem ve formě řetězce znaků. Parametr *parse* obsahuje informaci o metodě, která se použije pro převod rovnice zadané řetězcem na rovnici, kterou dokáže kompilátor zpracovat. Při volání funkce *retFunkcitonValue* lze použít dvě metody pro převod rovnice. První metoda je založená na využití skriptovacího jazyka *JScript.Net*, přesněji využití funkce *eval*. Skriptovací jazyk *JScript.Net* je k dispozici v knihovně .Net Framework od verze 2.0. Část kódu pro převod rovnice pomocí *JScript.Net* je zobrazen na Obr. 27 a byl částečně převzat z [17].

```
private static readonly string _jscriptSource =
@"package Evaluator
{
    class Evaluator
    {
        public function Eval(expr : String) : String
        {
            return eval(expr);
        }
    }
}";
```

Obr. 27 Funkce eval pro výpočet hodnoty rovnice [17]

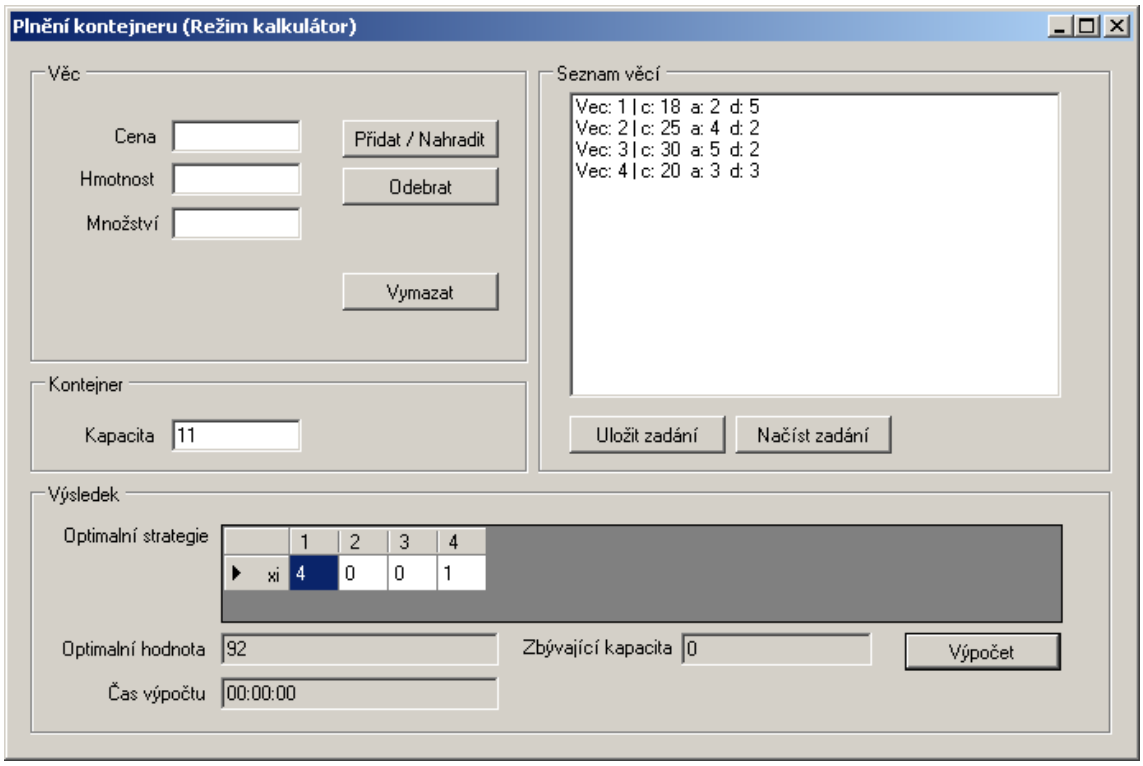
Druhou metodou pro převod rovnice je metoda, která využívá knihovnu *MTParserCOM*. Jedná se o knihovnu vytvořenou pro převod a výpočet rovnic zadaných ve formě řetězce znaků. Oproti první metodě je převod rovnic nepatrně rychlejší a umožnuje, aby rovnice obsahovala i nejpoužívanější funkce. Nevýhodou je oproti tomu nemožnost použití polí v rovnicích a nutnost přidávat knihovnu do registrů systému. Více informací o této knihovně je napsáno zde [18].

5.3 Funkce programového systému

V této kapitole budou zmíněny některé funkce, které programový systém obsahuje, jako např.: ukládání a načítání zadání úloh, nebo výpočtové režimy.

5.3.1 Výpočtové režimy

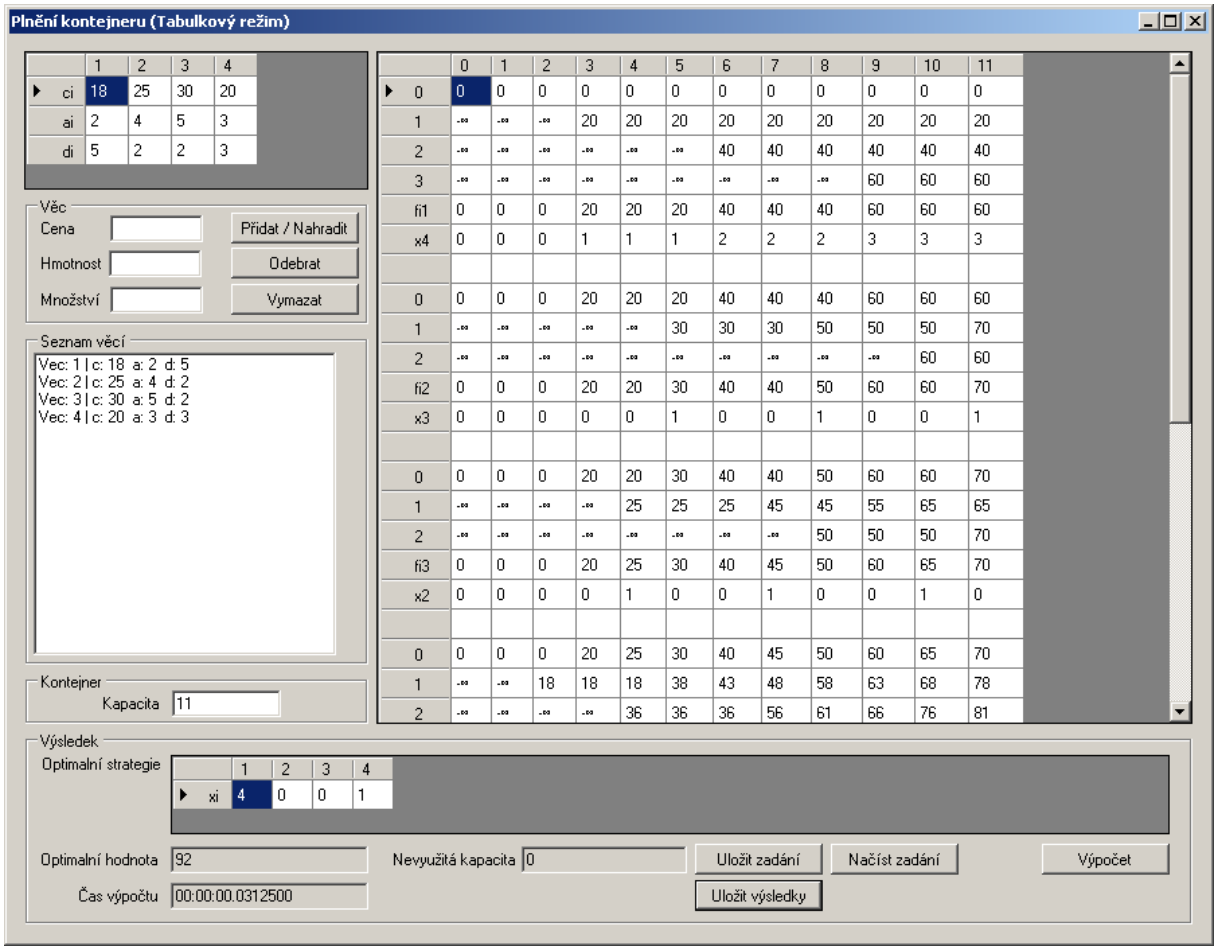
Při tvorbě programového systému byly realizovány dva druhy výpočtových režimů. Jeden režim je pro snadný, rychlý výpočet a jednoduché zobrazení výsledků. Druhý pro zobrazení nejen výsledku, ale také postupu výpočtu formou tabulky.



Obr. 28 Režim kalkulátor programového systému

Prvním výpočtovým režimem je režim kalkulátor. Výpočet úlohy plnění kontejneru v režimu kalkulátor je zobrazen na Obr. 28. Výhody režimu kalkulátor je rychlý výpočet a zobrazení výsledků a výpočet úlohy bez omezení hodnoty počátečního stavu. Oproti tomu nevýhodou režimu kalkulátor je nemožnost zobrazení průběhu výpočtu a tím hůře ověřitelný správný postup výpočtu.

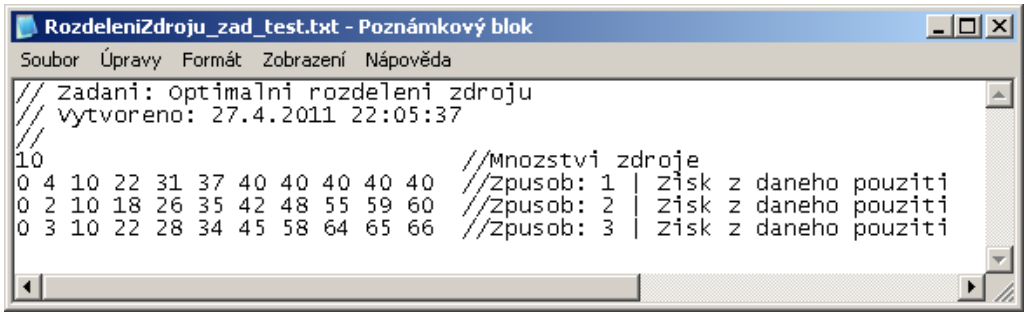
Dalším výpočtovým režimem je tabulkový režim. Tabulkový režim umožnuje zobrazit postup výpočtu ve formě tabulky mezivýsledků pro jednotlivé etapy a jednotlivé hodnoty počátečních stavů. Výhoda tabulkového režimu je tedy v možnosti zobrazení postupu výpočtu, toto je vhodné zejména pro kontrolu a ladění nově vytvořených úloh. Nevýhody tabulkového režimu jsou pomalejší výpočet, ale hlavně pomalejší zobrazení výsledků, které je závislé na velikosti úlohy, tj. počtu etap a počtu počátečních stavů. Další nevýhodou je omezení velikosti počátečního stavu z důvodu omezení komponenty DataGridView, která byla použita pro výpis výsledků. Výpočet úlohy plnění kontejneru v tabulkovém režimu je zobrazen na Obr. 29.



Obr. 29 Tabulkový režim programového systému

5.3.2 Načítání a ukládání

Další funkce, která byla do programového systému implementovaná je ukládání a načítání zadání pro každou úlohu realizovanou v programovém systému. Zadání se ukládá do standardního textového souboru *txt* z důvodu snadné editace uloženého zadání úlohy. Uložené zadání příkladu rozdělení zdrojů otevřené v poznámkovém bloku je ukázáno na Obr. 30.



Obr. 30 Uložené zadání úlohy rozdělení zdrojů

Samozřejmě je umožněno uložená zadání zpětně načítat do programového systému. To usnadňuje práci, protože zadání které je často používané, není nutné zadávat znovu a znovu. Po výpočtu úlohy v tabulkovém režimu je možno výsledek výpočtu s postupem výpočtu uložit. Ukládat výsledek s postupem výpočtu je možno jednak do excelovského tabulkového souboru *xsl*, ale i do souboru *csv*, což je jednoduchý textový soubor pro reprezentaci tabulkových dat, který pro oddělení dat ve sloupcích používá středník.

Soubor *csv* je podporován ve velkém množství různých programů a to i v těch nejzákladnějších pro úpravu textových souborů. Ukládání a načítání je realizováno přes standardní *OpenSaveDialog*, kde je možné vybrat umístění a typ souboru.

5.4 Návod použití programového systému

V rámci této kapitoly byl zpracován zjednodušený návod pro používání programového systému. Návod byl zpracován nejen pro řešení konkrétní úlohy a univerzální úlohy zadané rovnicemi, ale i pro přidávání nových úloh do programového systému.

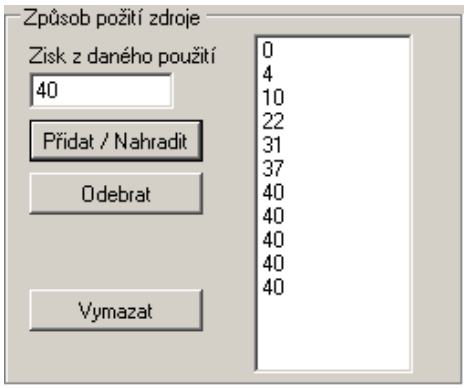
5.4.1 Řešení úlohy

Postup řešení úlohy pomocí programového systému bude ukázán např. na úloze rozdělení zdrojů. Zadání úlohy je uvedeno v podkapitole 6.3 kde je i numericky řešeno. Jedná se o problém přidělení zdroje takovým způsobem, aby se maximalizoval zisk z použití zdroje. Množství zdroje je 10 a zisky z použití zdroje jsou dány tabulkou Obr. 62.

Pro řešení úlohy je nejdříve nutné po spuštění programového systému vybrat z menu typ úlohy, kterou chceme řešit. Menu programového systému je zobrazeno na Obr. 31. Po vybrání úlohy se spustí okno reprezentující úlohu, kterou chceme řešit. Každá úloha je jiná, proto nelze vytvořit univerzální rozhraní pro všechny úlohy.

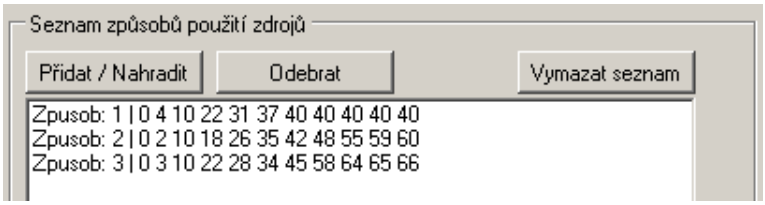
Obr. 31 Menu programového systému

Pro řešení úlohy je potřeba prostřednictvím ovládacích prvků zadat numerické zadání úlohy. Numerické zadání úlohy se vytváří následovně: nejprve je nutné vytvořit seznam zisků daného použití a následně vytvořený seznam zisků pro daný způsob použití přidat do seznamu všech způsobů použití. Vytváření způsobu použití je znázorněno na Obr. 32.



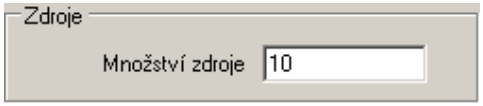
Obr. 32 Vytváření způsobu použití zdroje

Způsob použití zdroje byl vytvořen postupným přidáváním hodnot zisků, kdy první hodnota seznamu je chápána jako zisk ze způsobu použití, kdy množství zdroje je rovno 0. Druhá hodnota je analogicky zisk ze způsobu použití, kdy množstní zdroje je rovno 1. Takto vytvořený seznam zisků z daného způsobu použití zdroje přidáme do seznamu všech způsobů použití, to je znázorněno na Obr. 33.



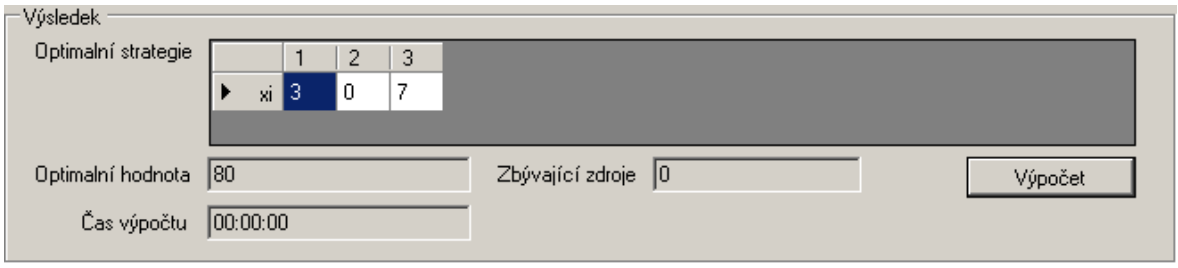
Obr. 33 Vytváření seznamu způsobů použití zdroje

Pro vytváření seznamu, a to jak seznamu zisků tak seznamu způsobů použití zdroje, se využívá příslušných tlačítek. Pomocí tlačítka *Přidat / Nahradit* je možné přidat další prvek do seznamu, případně nahradit stávající prvek seznamu novým. Pomocí tlačítka *Odebrat* lze odebrat označený prvek ze seznamu a pomocí tlačítka *Vymazat* lze odebrat všechny prvky seznamu. Pokud máme vytvořený seznam všech způsobu použití zdroje, tak už zbývá v rámci zadání pouze vyplnit množství zdroje, pro které se má úloha počítat. Vyplnění množství zdroje je zobrazeno na Obr. 34.



Obr. 34 Zadání množství zdroje

Posledním krokem řešení úlohy rozdělení zdrojů je spuštění samotného výpočtu, které se provádí tlačítkem *Výpočet*. Výsledek výpočtu úlohy je zobrazen na Obr. 35.



Obr. 35 Výsledek úlohy rozdělení zdrojů vypočtený programovým systémem

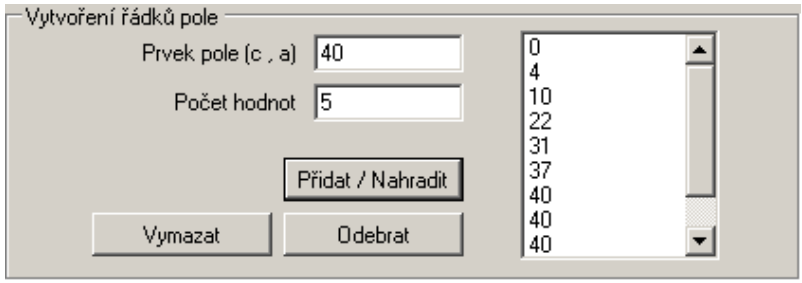
Tímto byla úloha rozdělení zdrojů vyřešena programovým systémem v režimu kalkulátor. Obdobným způsobem by se dala úloha řešit také v tabulkovém režimu.

5.4.2 Řešení úlohy zadané rovnicemi

Postup řešení úlohy zadané rovnicemi bude ukázán na úloze rozdělení zdrojů. Zadání úlohy je shodné se zadáním uvedeném v předchozí podkapitole. Jedná se o problém přidělení zdroje takovým způsobem, aby se maximalizoval zisk z použití zdroje. Množství zdroje je 10 a zisky z použití zdroje jsou dány tabulkou Obr. 62.

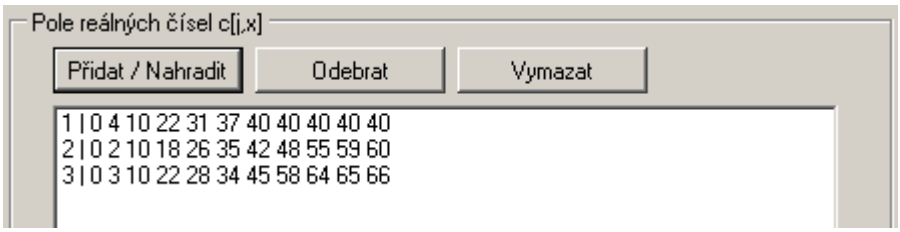
Nejdříve je opět nutné vybrat a spustit úlohu z menu. Po otevření okna reprezentující úlohu je nutné vyplnit zadání a rovnice úlohy. Nejdříve si zde ukážeme jak vytvořit data, se kterými bude úloha pracovat. Protože se jedná o úlohu rozdělení zdrojů, tak data budou reprezentovat zisky z různých způsobů použití zdroje. Tyto zisky mohou obecně nabývat reálných hodnot proto je přidáme do dvourozměrného pole reálných hodnot $c[j, x]$.

Dvourozměrné pole se bude vytvářet po řádcích, kdy řádek bude odpovídat v úloze rozdělení zdrojů jednomu způsobu použití. Vytváření prvního způsobu použití je zobrazeno na Obr. 36.



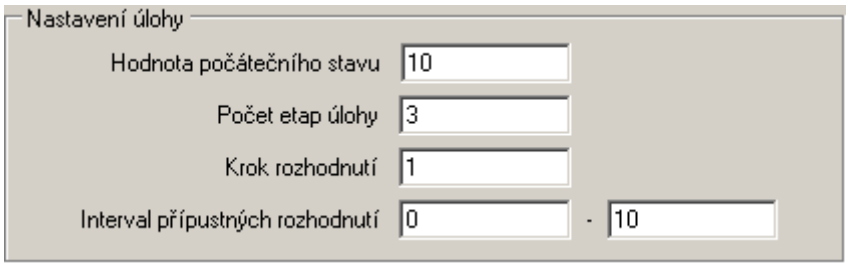
Obr. 36 Vytváření řádků pole c[j, x]

Při vytváření řádku je možné přidat i více stejných hodnot. To se provádí nastavením počtu hodnot na hodnotu vyšší než jedna. Pokud je řádek pole vytvořen přidá se do pole. To je znázorněno na Obr. 37.



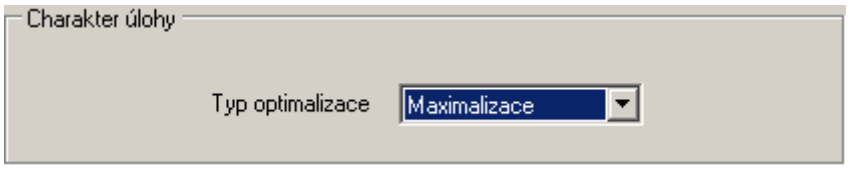
Obr. 37 Vytváření pole c[j, x]

Postup zadání úlohy byl zatím stejný jako v předchozí podkapitole. Dále je potřeba nastavit některé parametry úlohy. Nastavení parametrů úlohy je zobrazeno na Obr. 38.



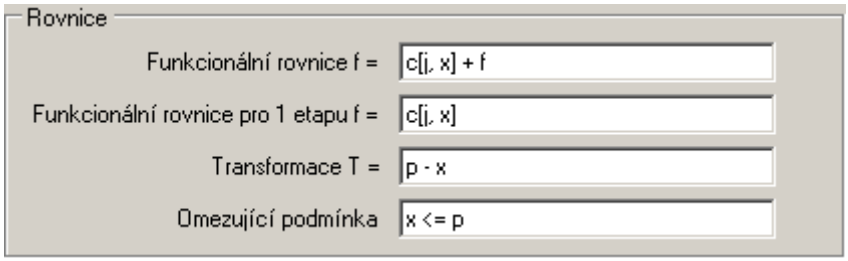
Obr. 38 Postup nastavení úlohy

Při nastavení parametrů úlohy je hodnota počátečního stavu rovna množství zdrojů, které je k dispozici. Počet etap úlohy je roven třem, protože máme tři různé způsoby využití zdroje. Krok rozhodnutí je většinou roven jedné, lze nastavit i vyšší, potom výpočet trvá kratší dobu, ale za cenu získání pouze přibližné hodnoty výsledku. Interval přípustných rozhodnutí je od 0 do 10, jelikož nejmenší přípustné rozhodnutí je nevyužít žádné množství zdroje a nejvyšší možné rozhodnutí je využít všechno množství zdroje.



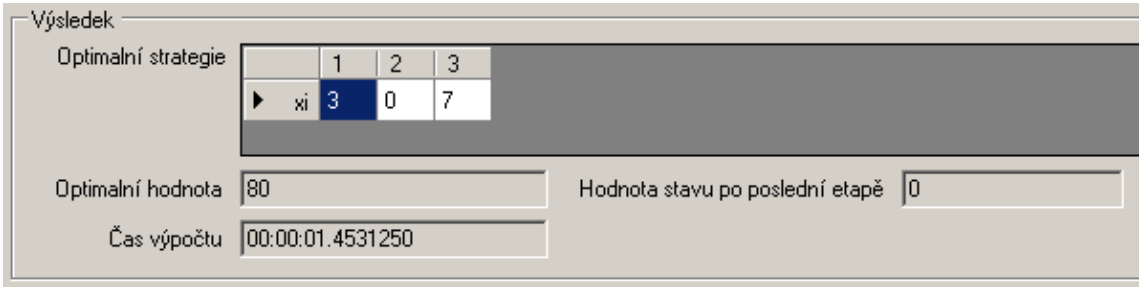
Obr. 39 Nastavení charakteru úlohy

Dále je nutné nastavit charakter úlohy, tzn. jestli se jedná o úlohu maximalizační nebo minimalizační. Nastavení charakteru úlohy je zobrazeno na Obr. 39.



Obr. 40 Nastavení rovnic pro výpočet úlohy

Poslední věc, kterou je nutné udělat, je zadat rovnice pro výpočet úlohy. Zavedení rovnic pro výpočet úlohy rozdělení zdrojů je zobrazeno na Obr. 40. Při zápisu rovnic pro výpočet úlohy lze použít běžné matematické operátory pro sčítání, odčítání, násobení, dělení a mocnění ve tvaru uvedeném v závorce (+ - * / ^). Při vytváření rovnic se využívají proměnné zavedené v programovém systému a jsou to proměnné: optimální hodnota účelové funkce pro předcházející etapu *f*, číslo etapy *j*, rozhodnutí *x* a hodnota stavu *p*. Při vytváření rovnice pro výpočet optimální hodnoty účelové funkce zapisujeme pouze vztah za maximalizací případně minimalizací. U rozdělení zdrojů se jedná o obdobnou rovnici jako (3.9), kde část účelová funkce tvaru *c*[*j, x*] je tabelována v závislosti na etapě procesu; v tomto případě na způsobu použití *j* a na rozhodnutí *x*. Funkcionální rovnice pro první etapu procesu je obdobou rovnice (3.10) a zapisuje se pouze vztah za maximalizací. Transformace je v podobném tvaru jako (3.8), ale zapisuje se pouze pravá strana rovnice. Poslední rovnice je omezující podmínka viz rovnice (3.7). Tímto byly všechny potřebné věci pro výpočet nastaveny a je možno spustit výpočet.



Obr. 41 Výsledek úlohy zadané rovnicemi

Chvíli po spuštění výpočtu se zobrazí výsledek. Výsledek je zobrazen na Obr. 41 a je shodný s výsledkem vypočítaným pomocí úlohy rozdělení zdrojů. Můžeme si všimnout, že čas potřebný pro výpočet úlohy je mnohonásobně vyšší. Větší časová náročnost je způsobena časovou náročností funkce pro převod rovnic, která je opakovaně volána. Časová náročnost je jedna z významných nevýhod výpočtu pomocí úlohy zadané rovnicemi.

5.4.3 Přidávání nových úloh

Postup přidání nové úlohy si předvedeme na úloze rozdělení zdrojů. Při vytváření třídy nové úlohy musí být třída potomkem abstraktní úlohy a musí mít implementované metody nutné pro výpočet dané úlohy. Jedná se o funkce pro výpočet optimální hodnoty účelové funkce, funkce pro výpočet transformace, funkce vracející množinu přípustných rozhodovacích proměnných a funkce s omezující podmínkou.

```
public override double Function(int j, int x, int i, double[] prev)
{
    return Limitation(j, x, i, prev);
}

public override double Limitation(int j, int x, int i, double[] prev)
{
    if (x > i) { return MaxMin(); }
    else { return use[j, x] + prev[0]; }
}
```

Obr. 42 Funkce pro výpočet hodnoty účelové funkce

Na Obr. 42 jsou zobrazeny funkce pro výpočet hodnoty účelové funkce pro j-tou etapu procesu, rozhodnutí *x* a hodnotu počátečního stavu *i*. Pro výpočet hodnoty účelové funkce se volá funkce *Function*, která zavolá funkce *Limitation* z důvodu zda se jedná o přípustné rozhodnutí.

V úloze rozdělení zdrojů je podmínka jednoduchá, rozhodnutí *x* nesmí být větší než hodnota stavu *i* tedy množství zdroje. V případě, že je rozhodnutí *x* vyšší než *i*, jedná se o nepřípustný stav přidělení většího množství zdroje než jaké je k dispozici. Pro jakýkoliv nepřípustný stav a pro maximalizační úlohu funkce vrací hodnotu *int.MinValue*, což je dostatečně malá hodnota aby nezasahovala do následného výpočtu. Pro přípustné hodnoty rozhodnutí vrací funkce *Limitation* součet hodnoty zisku z daného způsobu použití zdroje pro dané rozhodnutí *use*[*j*, *x*] a optimální hodnoty účelové funkce z předcházející etapy.

Tímto jsme získali hodnotu účelové funkce, ovšem se nejedná o optimální hodnotu. Výpočet optimální hodnoty účelové funkce je znázorněn na Obr. 43.

```
public override bool MaxMin(int j, int x, int i, double[] prev,
double maxMin, out double maxMinResult)
{
    if ((maxMinResult = Function(j, x, i, prev)) > maxMin)
    { return true; }
    else { return false; }
}
```

Obr. 43 Funkce pro výpočet optimální hodnoty účelové funkce

Jelikož se jedná o maximalizační úlohu, tak funkce *MaxMin* vrací hodnotu *true* pokud vypočtená hodnota účelové funkce je vyšší než doposud nalezené maximum. Do vstupně výstupního parametru funkce se ukládá vypočtená hodnota účelové funkce. Tato hodnota se využije v případě, že funkce *MaxMin* vrátí hodnotu *true*.

Další funkce, kterou je potřeba zavést, je funkce pro transformaci stavu. Funkce pro výpočet hodnoty stavu po transformaci je zobrazena na Obr. 44.

```
public override int Transform(int j, int x, int p, int i, out int tranJ)
{
    tranJ = j + 1;
    return (p - x);
}
```

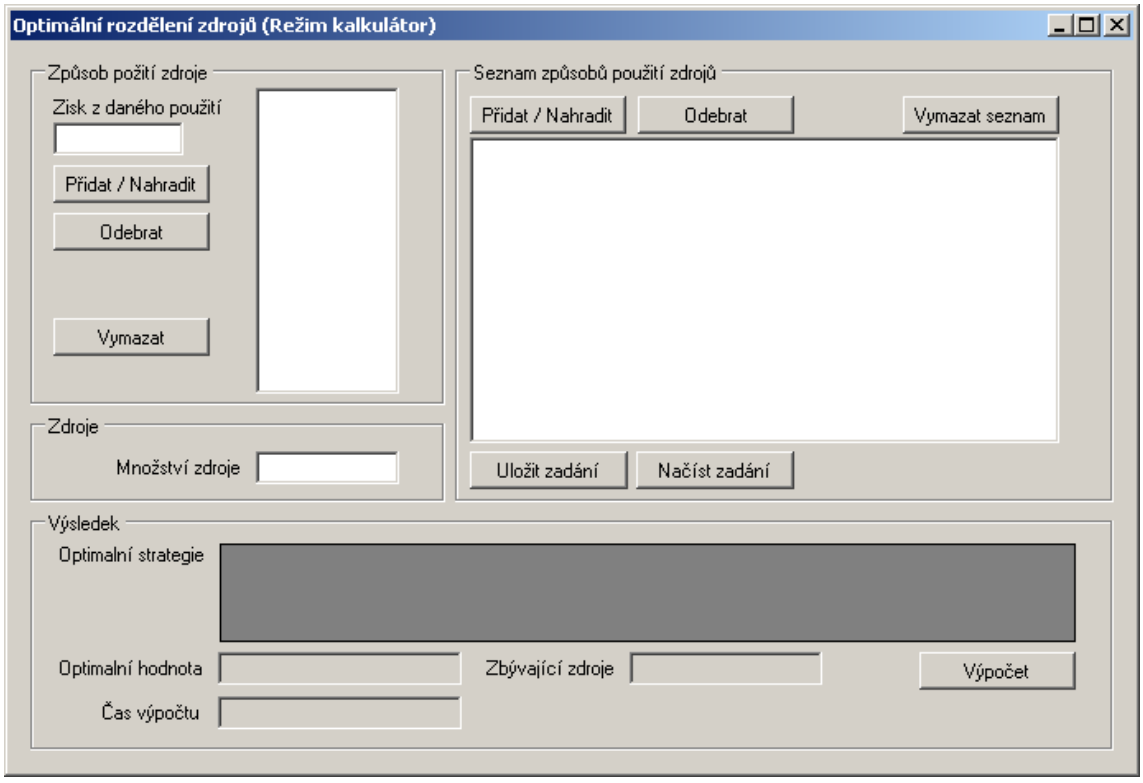
Obr. 44 Funkce pro transformaci stavu

Hodnoty nového stavu po transformaci jsou dány vztahem *p* - *x*, kde *p* je hodnota stavu před transformací a *x* je rozhodnutí. Vstupně výstupní hodnota *tranJ* obsahuje informaci o tom do jaké etapy se vlivem transformace dostaneme. Poslední funkce, kterou je nutné definovat, je funkce vracející počet rozhodnutí. Funkce je zobrazena na Obr. 45.

```
public override int RetNumOfDecision(int j)
{
    return maxNumOfDecision;
}
```

Obr. 45 Funkce pro výpočet počtu rozhodnutí

Funkce *RetNumOfDecision* vrací hodnotu *maxNumOfDecision*, která je rovna, pro úlohu rozdělení zdrojů počtu řádků dvourozměrného pole *use*[*j*, *x*], tj. jedná se o nejvyší množství přidělení zdroje pro všechny způsoby použití zvětšené o jedničku, protože uvažujeme i rozhodnutí nepřidělit žádné množství zdroje. Tímto byla vytvořena třída úlohy rozdělení zdrojů a funkce potřebné pro výpočet úlohy. Pro použití programového systému pro řešení nově vytvořené úlohy je také nutné vytvořit uživatelské rozhraní pro zadávání dat se kterými úloha pracuje. Uživatelské rozhraní úlohy rozdělení zdrojů je zobrazené na Obr. 46.



Obr. 46 Uživatelské rozhraní úlohy rozdělení zdrojů

6 OVĚŘENÍ FUNKČNOSTI SYSTÉMU

Funkčnost řešitele programového systému a vytvořených úloh byla ověřena vypočtením dostatečného množství příkladů pro dané úlohy. Příklady byly vypočteny nejdříve ručně popřípadě pomocí tabulkového procesoru Excel, posléze pomocí programového systému a takto získané výsledky byly následně porovnány.

6.1 Úloha o batohu

Příklad je zadán následovně: je dán batoh o nosnosti $b = 16$, a jsou dány čtyři věci, jejichž ceny c_j a hmotnosti a_j jsou zadány tabulkou na Obr. 47. Úkolem je naplnit batoh tak, aby součet všech cen věcí v batohu byl co největší, ale zároveň aby součet všech hmotností věcí v batohu nepřesáhl nosnost batohu. Více informací o úloze o batohu, včetně funkcionálních rovnic pro výpočet úlohy bylo zmíněno v podkapitole 3.1.1.

j	1	2	3	4
c_j	12	23	30	42
a_j	3	5	7	10

Obr. 47 Zadání příkladu úlohy o batohu

Optimální hodnotu účelové funkce a optimální rozhodovací strategii lze vypočíst následujícím způsobem. Pokud budeme postupovat zpětným dynamickým programováním, tak v první etapě rozhodujeme zda umístíme do batohu poslední tj. čtvrtou věc. V první etapě procesu je optimální hodnota účelová funkce dána vztahem (3.5).

	x_4			
p_4	0	1	$f_1(p_4)$	x_4
16	0	42	42	1

Obr. 48 Výpočet optimální hodnoty účelové funkce pro první etapu

Výpočet účelové funkce pro první etapu můžeme znázornit tabulkou na Obr. 48. Pokud v první etapě procesu bude rozhodnutí $x_4 = 0$, tj. věc nedát do batohu bude hodnota účelové funkce rovna nule. Pokud bude rozhodnutí $x_4 = 1$, bude hodnota účelové funkce rovna 42. Jelikož úloha o batohu je maximalizační tak optimální hodnota účelové funkce $f_1(p_4) = 42$ je maximum ze dvou hodnot účelových funcí a rozhodnutí $x_4 = 1$.

	x_3			
p_3	0	1	$f_2(p_3)$	x_3
16	42	30	42	0

Obr. 49 Výpočet optimální hodnoty účelové funkce pro druhou etapu

Ve všech etapách kromě první je výpočet optimální hodnoty účelové funkce dán vztahem (3.4). Výpočet účelové funkce v druhé etapě je znázorněn tabulkou na Obr. 49. Kde pro rozhodnutí $x_3 = 0$ je hodnota účelové funkce rovna 42, tj. hodnotě optimální účelové funkce v předchozí etapě a pro $x_3 = 1$ je účelová funkce rovna 30. Optimální hodnota účelové funkce je rovna maximu ze dvou hodnot, proto $f_2(p_3) = 42$ a rozhodnutí $x_3 = 0$.

	x_2			
p_2	0	1	$f_3(p_2)$	x_2
16	42	65	65	1

Obr. 50 Výpočet optimální hodnoty účelové funkce pro třetí etapu

Výpočet účelové funkce pro třetí etapu procesu je zobrazen tabulkou na Obr. 50. Kde pro

rozhodnutí $x_2 = 0$ je hodnota účelové funkce rovna 42, a pro $x_2 = 1$ je účelová funkce rovna 65, tj. součtem ceny druhé věci c_2 a optimální účelové funkce pro předchozí etapu pro zbývající kapacitu batohu. Hodnotu účelové funkce jsme získali jako součet $c_2 = 23$ a optimální hodnoty účelové funkce pro zbývající kapacitu batohu po vložení druhé věci $f_3(11) = 42$. Optimální hodnota účelové funkce je rovna maximu ze dvou hodnot, tudíž $f_3 = 65$ a rozhodnutí $x_2 = 1$.

	x_1			
p_1	0	1	$f_4(p_1)$	x_1
16	65	65	65	0; 1

Obr. 51 Výpočet optimální hodnoty účelové funkce pro čtvrtou etapu

Výpočet účelové funkce pro čtvrtou, tedy poslední etapu procesu lze vidět na Obr. 51. Při rozhodnutí $x_1 = 0$ je hodnota účelové funkce rovna 65, a pro $x_1 = 1$ je účelová funkce rovna 65. Hodnotu účelové funkce pro $x_1 = 1$ jsme získali jako součet $c_1 = 12$ a optimální hodnoty účelové funkce pro zbývající kapacitu batohu po vložení první věci $f_3(13) = 53$. Optimální hodnota účelové funkce je rovna maximu ze dvou hodnot, v tomto případě jsou obě hodnoty stejné, proto optimální hodnota účelové funkce nezávisí na tom zda první věc dáme do batohu či ne, v obou případech bude rovna $f_4 = 65$ a rozhodnutí může nabývat hodnot $x_1 = 0$, nebo $x_1 = 1$. Tímto jsme získali optimální hodnotu účelové funkce pro kapacitu batohu $b = 16$, $f_4(p_1) = 65$ a dvě optimální strategie jak tuto hodnotu dosáhnout. Optimální strategie získáme pomocí transformační funkce (3.3). Získání první strategie je znázorněno tabulkou na Obr. 52.

j	p_j	x_j
1	16	0
2	16	1
3	11	0
4	11	1

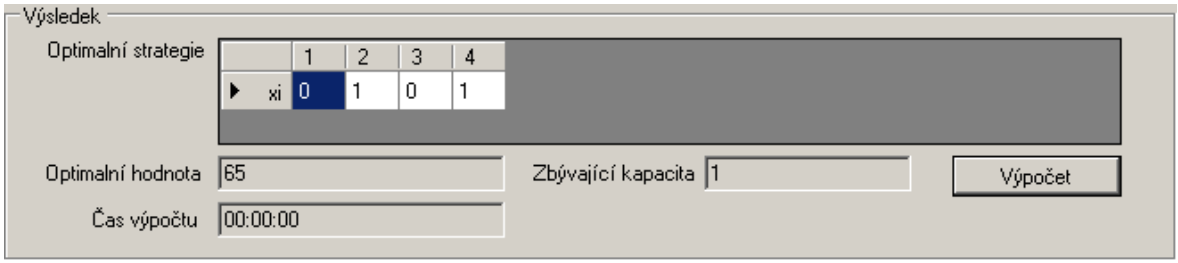
Obr. 52 Výpočet první optimální strategie

Postup získání optimální strategie je následující: nejdříve do tabulky zapíšeme hodnotu stavu p_1, který se rovná počáteční kapacitě batohu $b = 16$, dále zapíšeme rozhodnutí x_1, které jsme při výpočtu optimální strategie vypočítali. Jelikož je rozhodnutí $x_1 = 0$, tak stav $p_2 = p_1$. Dále zapíšeme do tabulky rozhodnutí x_2 pro hodnotu stavu $p_2 = 16$. Protože $x_2 = 1$, tak stav $p_3 = p_2 - a_2$, takto získáme hodnotu stavu $p_3 = 11$. Hledáme rozhodnutí x_3 ale pro zbývající kapacitu batohu $p_3 = 11$. Rozhodnutí $x_3 = 0$, proto se hodnota stavu $p_4 = p_3$. Hledáme rozhodnutí x_4 pro zbývající kapacitu $p_4 = 11$, rozhodnutí $x_4 = 1$. Tímto postupem jsme získali první optimální strategii pro zadanou úlohu, která je dána posloupností rozhodnutí {0, 1, 0, 1}.

j	p_j	x_j
1	16	1
2	13	1
3	8	1
4	1	0

Obr. 53 Výpočet druhé optimální strategie

Obdobným postupem získáme i druhou optimální strategii, postup získání optimální strategie a získaná optimální strategie je na Obr. 53. Tímto jsme vyřešili zadanou úlohu ručně, teď zbývá vyřešit úlohu pomocí programového systému.



Obr. 54 Výsledek příkladu úlohy o batohu spočtený programovým systémem

Výsledek vypočtený v programovém systému je zobrazený na Obr. 54, vypočtená optimální hodnota účelové funkce a optimální strategie je shodná s předchozím vypočteným výsledkem. Z výsledku programového systému je ještě vidět, že programový systém zobrazuje pouze prvě nalezenou optimální strategii.

6.2 Plnění kontejneru

Problém plnění kontejneru je zobecněním úlohy o batohu, kdy rozhodovací proměnné nenabývají pouze hodnot 0 a 1, ale mohou nabývat až hodnot d_j což je počet kusů j-té věci. Příklad je zadán takto: je dán kontejner o kapacitě $b = 11$ a jsou dány čtyři věci, kdy cena c_j, hmotnost a_j a počet kusů d_j jsou dány tabulkou Obr. 55. Úkolem je naplnit kontejner tak aby byl součet všech cen věcí v kontejneru co nejvyšší a zároveň aby součet všech hmotností věcí nepřesáhl limit kapacity kontejneru. Podrobněji rozebrána byla úloha plnění kontejneru v podkapitole 3.1.2. Z důvodu lepšího ověření výsledků je zadání příkladu převzato z publikace [7].

j	1	2	3	4
c_j	18	25	30	20
a_j	2	4	5	3
d_j	5	2	2	3

Obr. 55 Zadání příkladu plnění kontejneru

Optimální hodnotu účelové funkce a optimální rozhodovací strategii lze vypočíst následujícím způsobem. Pokud budeme postupovat zpětným dynamickým programováním, tak v první etapě rozhodujeme, zda umístíme do kontejneru poslední tj. čtvrtou věc. V první etapě procesu je optimální hodnota účelová funkce dána stejným vztahem jako u úlohy o batohu (3.5).

	x_4					
p_4	0	1	2	3	$f_1(p_4)$	x_4
0	0				0	0
1	0				0	0
2	0				0	0
3	0	20			20	1
4	0	20			20	1
5	0	20			20	1
6	0	20	40		40	2
7	0	20	40		40	2
8	0	20	40		40	2
9	0	20	40	60	60	3
10	0	20	40	60	60	3
11	0	20	40	60	60	3

Obr. 56 Výpočet optimální hodnoty účelové funkce pro první etapu

Výpočet optimální hodnoty účelové funkce pro první etapu je na Obr. 56, optimální hodnota účelové funkce pro první etapu procesu je maximum z hodnot účelových funkcí pro stav p_4. Optimální hodnota účelové funkce $f_1(p_4) = 60$ pro $p_4 = 11$ a rozhodnutí $x_4 = 3$. Ostatní optimální hodnoty

účelových funkcí spočítané pro různé stavy $p_4 < 11$ budou použity v dalších etapách výpočtu.

	x_3				
p_3	0	1	2	$f_2(p_3)$	x_3
0	0			0	0
1	0			0	0
2	0			0	0
3	20			20	0
4	20			20	0
5	20	30		30	1
6	40	30		40	0
7	40	30		40	0
8	40	50		50	1
9	60	50		60	1
10	60	50	60	60	0; 2
11	60	70	60	70	1

Obr. 57 Výpočet optimální hodnoty účelové funkce pro druhou etapu

Výpočet optimální hodnoty účelové funkce pro druhou etapu je znázorněn na Obr. 57, optimální hodnota účelové funkce byla vypočtena vztahem (3.4). Optimální hodnota účelové funkce pro druhou etapu procesu pro stav reprezentující kapacitu kontejneru $p_3 = 11$ je $f_2(p_3) = 70$ a rozhodnutí $x_3 = 1$. Ostatní hodnoty byly vypočteny pro další etapy procesu.

	x_2				
p_2	0	1	2	$f_3(p_2)$	x_2
0	0			0	0
1	0			0	0
2	0			0	0
3	20			20	0
4	20	25		25	1
5	30	25		30	0
6	40	25		40	0
7	40	45		45	1
8	50	45	50	50	0; 2
9	60	55	50	60	0
10	60	65	50	65	1
11	70	65	70	70	0; 2

Obr. 58 Výpočet optimální hodnoty účelové funkce pro třetí etapu

Postup výpočtu optimální hodnoty účelové funkce je znázorněn tabulkou na Obr. 58, kde optimální hodnota účelové funkce je maximum ze všech hodnot účelových funkcí pro stav $p_3 = 11$. Maximum a tedy optimální hodnota učelové funkce $f_3(p_2) = 70$ a rozhodnutí $x_2 = 0$. Hodnoty účelových funkcí pro $p_2 < 11$ byly vypočteny pro použití v další etapě procesu.

	x_1							
p_1	0	1	2	3	4	5	$f_4(p_1)$	x_1
11	70	78	81	84	92	90	92	4

Obr. 59 Výpočet optimální hodnoty účelové funkce pro čtvrtou etapu

Výpočet optimální účelové funkce pro poslední etapu, která je rovna výsledné hodnotě optimální účelové funkce tohoto příkladu, je zobrazena tabulkou na Obr. 59.

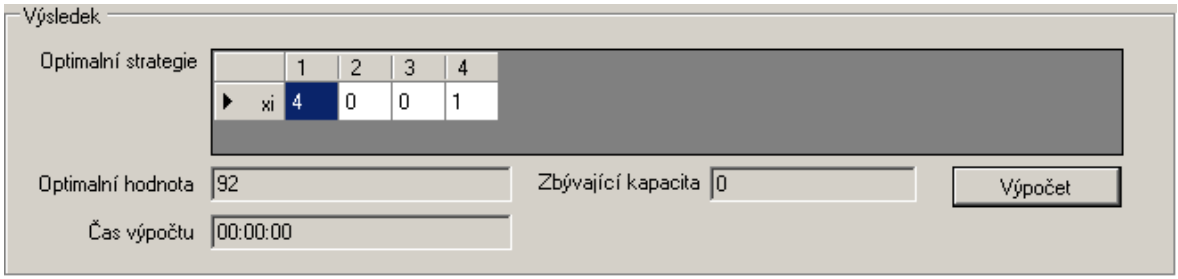
Výsledná optimální hodnota účelové funkce je rovna maximu z hodnot účelových funkcí pro hodnotu stavu $p_1 = 11$, tudíž $f_4(p_1) = 92$ a rozhodnutí $x_1 = 4$. Hodnoty účelových funkcí pro ostatní stavy p_1 není nutné počítat, protože se jedná o poslední etapu procesu.

Když už máme vypočtenou výslednou optimální hodnotu účelové funkce, tak začneme počítat rozhodovací strategii, kterou dosáhneme optimální hodnoty účelové funkce. Výpočet optimální strategie je znázorněn tabulkou Obr. 60.

j	p_j	x_j
1	11	4
2	3	0
3	3	0
4	3	1

Obr. 60 Výpočet optimální strategie

Při výpočtu optimální strategie postupujeme následovně, nejdříve do tabulky zapíšeme hodnotu stavu p_1, který je roven počáteční kapacitě kontejneru $b = 11$, dále zapíšeme rozhodnutí x_1, vypočtené při výpočtu hodnoty optimální účelové funkce pro poslední etapu. Rozhodnutí $x_1 = 4$, stav p_2 vypočteme pomocí transformace $p_2 = p_1 - x_1 a_1$. Stav $p_2 = 3$, dále zapíšeme do tabulky rozhodnutí x_2 pro hodnotu stavu $p_2 = 3$. Protože $x_2 = 0$, tak stav $p_3 = p_2$, takto získáme hodnotu stavu $p_3 = 3$. Hledáme rozhodnutí x_3 ale pro zbývající kapacitu kontejneru $p_3 = 3$. Rozhodnutí $x_3 = 0$, proto se stav $p_4 = p_3$. Hledáme rozhodnutí x_4 pro zbývající kapacitu $p_4 = 3$, rozhodnutí $x_4 = 1$. Tímto postupem jsme získali optimální strategii pro zadaný příklad, která je dána posloupností rozhodnutí {4, 0, 0, 1}.



Obr. 61 Výsledek příkladu plnění kontejneru vypočtený programovým systémem

Výsledek vypočtený v programovém systému je zobrazený na Obr. 61, kde vypočtená optimální hodnota účelové funkce i optimální strategie je shodná s předchozím vypočteným výsledkem.

6.3 Rozdělení drojů

Úloha rozdělení zdrojů je popsána v rámci podkapitoly 3.1.3. Příklad je zadán následovně: je dáno omezené množství zdroje $p = 10$ a tři možné způsoby jak daný zdroj využít. Účelové funkce všech tří způsobů použití jsou zobrazeny tabulkou na Obr. 62. Úkolem je vypočítat optimální hodnotu účelové funkce a optimální strategii rozdělení zdroje. Numerické údaje zadání příkladu byly pro kontrolu převzaty z [3].

$x_1 = x_2 = x_3$	$g_1(x_1)$	$g_2(x_2)$	$g_3(x_3)$
0	0	0	0
1	4	2	3
2	10	10	10
3	22	18	22
4	31	26	28
5	37	35	34
6	40	42	45
7	40	48	58
8	40	55	64
9	40	59	65
10	40	60	66

Obr. 62 Zadání příkladu rozdělení zdrojů

Optimální hodnotu účelové funkce a optimální rozhodovací strategii vypočteme následujícím postupem. V první etapě procesu rozhodujeme o třetím způsobu použití zdroje. Optimální hodnoty účelové funkce pro první etapu procesu vypočteme pomocí vztahu (3.10), pro ostatní etapy použijeme vztah (3.9).

p_3	$f_1(p_3)$	x_3
0	0	0
1	3	1
2	10	2
3	22	3
4	28	4
5	34	5
6	45	6
7	58	7
8	64	8
9	65	9
10	66	10

Obr. 63 Výpočtené optimální hodnoty účelových funkcí pro první etapu

Výpočtené optimální hodnoty účelové funkce pro první etapu jsou znázorněny na Obr. 63. Optimální hodnota účelové funkce pro první etapu procesu $f_1(p_3)$ je maximum z účelových funkcí přes všechna rozhodnutí x_3. Optimální hodnoty účelových funkcí pro množství zdroje $p_3 < 10$, byly vypočteny z důvodu použití v další etapě procesu.

	x_2												
p_2	0	1	2	3	4	5	6	7	8	9	10	$f_2(p_2)$	x_2
0	0											0	0
1	3	2										3	0
2	10	5	10									10	0; 2
3	22	12	13	18								22	0
4	28	24	20	21	26							28	0
5	34	30	32	28	29	35						35	5
6	45	36	38	40	36	38	42					45	0
7	58	47	44	46	48	45	45	48				58	0
8	64	60	55	52	54	57	52	51	55			64	0
9	65	66	68	63	60	63	64	58	58	59		68	2
10	66	67	74	76	71	69	70	70	65	62	60	76	3

Obr. 64 Výpočtené optimální hodnoty účelových funkcí pro druhou etapu

Vypočtené optimální hodnoty účelových funkcí pro druhou etapu procesu $f_2(p_2)$ jsou zobrazeny na Obr. 64, kde hodnoty $f_2(p_2)$ pro $p_2 < 10$ byly vypočteny pro použití v další etapě.

	x_1												
p_1	0	1	2	3	4	5	6	7	8	9	10	$f_3(p_1)$	x_1
10	76	72	74	80	76	72	68	62	50	43	40	80	3

Obr. 65 Vypočtená optimální hodnota účelové funkce pro třetí etapu

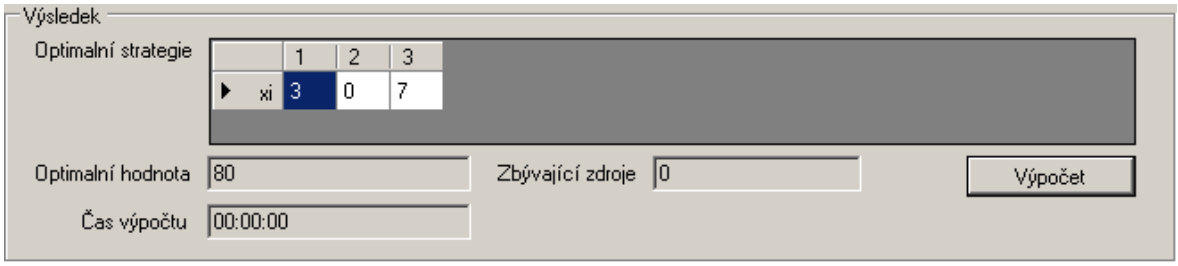
Optimální hodnota účelové funkce pro třetí etapu $f_3(p_1)$ je zároveň výsledná optimální hodnota zadaného příkladu a je zobrazena v tabulce na Obr. 65. Výsledná optimální hodnota účelové funkce $f_3(p_1) = 80$. Ostatní optimální hodnoty účelových funkcí pro množství zdroje $p_1 < 10$ nebylo nutné počítat, protože jde o poslední etapu procesu.

Jestliže už máme vypočtenou optimální hodnotu účelové funkce, tak zbývá spočítat optimální rozhodovací strategii. Výpočet optimální rozhodovací strategie je znázorněn na Obr. 66 a při výpočtu bylo použito vztahu pro transformaci (3.8).

j	p_j	x_j
1	10	3
2	7	0
3	7	7

Obr. 66 Výpočet optimální strategie

Postup výpočtu optimální strategie je následující: nejdříve do tabulky zapíšeme hodnotu stavu p_1 odpovídající zadanému množství zdroje $p = 10$. Následně opíšeme rozhodnutí $x_1 = 3$ z třetí etapy procesu. Další hodnotu stavu p_2 vypočteme pomocí transformace $p_2 = p_1 - x_1$. Zbývající množství zdroje v druhé etapě procesu je $p_2 = 7$. Zjistíme rozhodnutí v druhé etapě procesu x_2 pro hodnotu stavu p_2. Zjištěné rozhodnutí $x_2 = 0$, proto hodnota stavu $p_3 = p_2 = 7$. Nakonec zjistíme poslední rozhodnutí x_3 pro hodnotu stavu p_3. Poslední rozhodnutí je rovno $x_3 = 7$, tedy množství doposud nepřiděleného zdroje. Tímto postupem jsme získali optimální rozhodovací strategii zadaného příkladu, která je dána posloupností rozhodnutí {3, 0, 7}.



Obr. 67 Výsledek příkladu rozdělení zdrojů vypočtený programovým systémem

Výsledek příkladu rozdělení zdrojů, tj. optimální hodnota účelové funkce a optimální rozhodovací strategie, vypočtený v programovém systému, je zobrazen na Obr. 67. Vypočtené výsledné hodnoty se shodují s předchozím vypočteným výsledkem.

6.4 Distribuce zdravotních týmů

Úloha distribuce zdravotních týmů vychází z obecné úlohy rozdělení zdrojů. Úloha je zadána následovně: je dáno pět zdravotních týmů a tři státy, kde by se zdravotní týmy daly využít. Účelová funkce představuje vliv zvýšení zdravotní péče v závislosti na počtu přidělených zdravotních týmů. Hodnota účelové funkce je tisícinásobkem součtu nárůstu délky života všech občanů ve státě. Úkolem je vypočítat optimální hodnotu účelové funkce a rozhodovací strategii, kterou lze tuho hodnotu dosáhnout. Účelové funkce jednotlivých států jsou zobrazeny tabulkou Obr. 68. Numerické hodnoty zadání příkladu byly pro kontrolu převzaty z [6]. Více informací o této úloze bylo zmíněno v rámci podkapitoly 3.1.4.

$x_1 = x_2 = x_3$	$g_1(x_1)$	$g_2(x_2)$	$g_3(x_3)$
0	0	0	0
1	45	20	50
2	70	45	70
3	90	75	80
4	105	110	100
5	120	150	130

Obr. 68 Zadání příkladu distribuce zdravotvích týmů

Pro výpočet optimální hodnoty účelové funkce byly použity vztahy (3.10) pro první etapu procesu a (3.9) pro statní etapy procesu. Postup výpočtu optimální hodnoty účelové funkce je následující.

p_3	$f_1(p_3)$	x_3
0	0	0
1	50	1
2	70	2
3	80	3
4	100	4
5	130	5

Obr. 69 Výpočtené optimální hodnoty účelových funkcí pro první etapu

Vypočtené optimální hodnoty účelových funkcí pro první etapu procesu jsou znázorněny v tabulce Obr. 69, kde optimální hodnota účelové funkce $f_1(p_3) = 130$ pro $p_3 = 5$ a rozhodnutí $x_3 = 5$. Optimální hodnoty účelových funkcí pro $p_3 < 5$ byly vypočteny z důvodu možného použití v další etapě procesu.

	x_2							
p_2	0	1	2	3	4	5	$f_2(p_2)$	x_2
0	0						0	0
1	50	20					50	0
2	70	70	45				70	0; 1
3	80	90	95	75			95	2
4	100	100	115	125	110		125	3
5	130	120	125	145	160	150	160	4

Obr. 70 Výpočtené optimální hodnoty účelových funkcí pro druhou etapu

Vypočtené optimální hodnoty účelové funkce pro druhou etapu procesu jsou vypsány v tabulce Obr. 70, kde optimální hodnota účelové funkce je $f_2(p_2) = 160$ pro $p_2 = 5$ a rozhodnutí $x_2 = 4$. Optimální hodnoty účelových funkcí pro stav $p_2 < 5$ byly opět vypočteny pro výpočet optimální hodnoty účelové funkce v další etapě procesu.

	x_1							
p_1	0	1	2	3	4	5	$f_3(p_1)$	x_1
0	0						0	0
1	50	45					50	0
2	70	95	70				95	1
3	95	115	120	90			120	2
4	125	140	140	140	105		140	1; 2; 3
5	160	170	165	160	155	120	170	1

Obr. 71 Výpočtené optimální hodnoty účelových funkcí pro třetí etapu

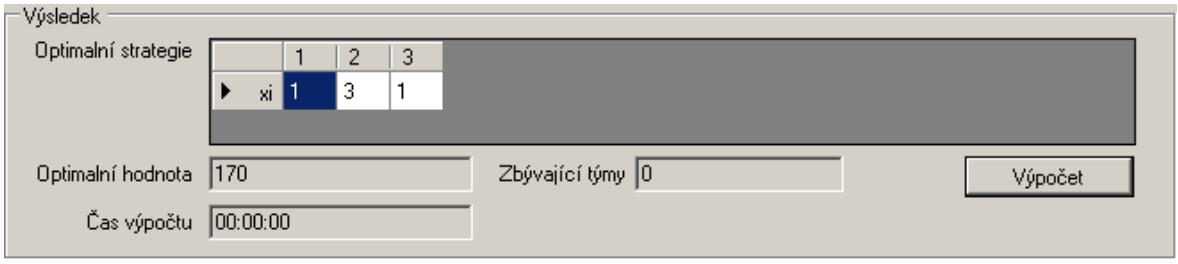
Výpočet optimálních hodnot účelových funkcí pro třetí etapu procesu je znázorněn tabulkou Obr. 71. Výsledná optimální hodnota účelové funkce $f_3(p_1) = 170$. Optimální hodnoty účelových funkcí pro $p_1 < 5$ nebylo nutné počítat, protože jde o poslední etapu procesu. V zadání příkladu bylo zjistit optimální hodnotu účelové funkce pro počet zdravotních týmů $p = 5$, ale zde je názorně ukázáno, že v rámci výpočtu příkladu lze zjistit bez většího počítání i optimální hodnoty účelových funkcí pro množství zdroje menší než bylo zadáno.

Tímto postupem jsme vypočetli optimální hodnotu účelové funkce zadaného příkladu. Nyní zbývá vypočítat rozhodovací strategii, kterou lze dosáhnout optimální hodnoty účelové funkce. Optimální strategii vypočítáme pomocí vztahu (3.8). Postup výpočtu optimální strategie je zobrazen tabulkou Obr. 72.

j	p_j	x_j
1	5	1
2	4	3
3	1	1

Obr. 72 Výpočet optimální strategie

Postup výpočtu optimální strategie je následující, nejdříve do tabulky zapíšeme hodnotu stavu p_1 odpovídající zadanému počtu zdravotních týmů $p = 5$. Následně opíšeme rozhodnutí $x_1 = 1$ z třetí etapy procesu. Další hodnotu stavu p_2 vypočteme pomocí transformace $p_2 = p_1 - x_1$. Zbývající množství zdroje v druhé etapě procesu $p_2 = 4$. Zjistíme rozhodnutí v druhé etapě procesu x_2 pro hodnotu stavu p_2. Zjištěné optimální rozhodnutí je $x_2 = 3$, proto hodnota stavu $p_3 = p_2 - x_2$. Zbývající počet zdravotních týmů ve třetí etapě je $p_3 = 1$. Zjistíme poslední rozhodnutí $x_3 = 1$ pro hodnotu stavu p_3. Tímto postupem jsme získali optimální rozhodovací strategii zadaného příkladu. Optimální rozhodovací strategie je dána posloupností rozhodnutí {1, 3, 1}.



Obr. 73 Výsledek příkladu distribuce zdravotních týmů vypočtený programovým systémem

Výsledek příkladu distribuce zdravotních týmů, tj. optimální hodnota účelové funkce a optimální rozhodovací strategie, vypočtený v programovém systému, je zobrazen na Obr. 73. Vypočtené výsledné hodnoty se shodují s předchozím vypočteným výsledkem.

6.5 Vytváření výzkumných týmů

Problém vytváření výzkumných týmů je specifičtějším případem obecnější úlohy rozdělení zdrojů. Úloha je zadána následovně, jsou dáni dva vědci a tři výzkumné týmy kam je možné vědce přidělit. Účelové funkce představuje pravděpodobnost selhání výzkumného týmu při výzkumu v závislosti na počtu přidělených vědců. Účelové funkce tří výzkumných týmů jsou zobrazeny v tabulce na Obr. 74. Úkolem úlohy je minimalizovat pravděpodobnost celkového selhání vědeckých týmu při výzkumu. Více informací o problému vytváření výzkumných týmů bylo zmíněno v podkapitole 3.1.5. Numerické hodnoty příkladu jsou převzaty z [6].

$x_1 = x_2 = x_3$	$g_1(x_1)$	$g_2(x_2)$	$g_3(x_3)$
0	0,40	0,60	0,80
1	0,20	0,40	0,50
2	0,15	0,20	0,30

Obr. 74 Zadání úlohy vytváření výzkumných týmů

Pro výpočet optimální hodnoty účelové funkce byly použity vztahy (3.14) pro první etapu procesu a (3.13) pro ostatní etapy procesu. Postup výpočtu optimální hodnoty účelové funkce je následující.

p_3	$f_1(p_3)$	x_3
0	0,80	0
1	0,50	1
2	0,30	2

Obr. 75 Výpočtené optimální hodnoty účelových funkcí pro první etapu

Vypočtené optimální hodnoty účelových funkcí pro první etapu procesu jsou znázorněny v

tabulce Obr. 75, kde optimální hodnota účelové funkce $f_1(p_3) = 0{,}3$ pro $p_3 = 2$ a rozhodnutí $x_3 = 2$. Optimální hodnoty účelových funkcí pro $p_3 < 2$ byly vypočteny z důvodu možného použití v další etapě procesu.

	x_2				
p_2	0	1	2	$f_2(p_2)$	x_2
0	0,48			0,48	0
1	0,30	0,32		0,30	0
2	0,18	0,20	0,16	0,16	2

Obr. 76 Výpočtené optimální hodnoty účelových funkcí pro druhou etapu

Vypočtené optimální hodnoty účelové funkce pro druhou etapu procesu jsou vypsány v tabulce na Obr. 76, kde optimální hodnota účelové funkce $f_2(p_2) = 0{,}16$ pro $p_2 = 2$ a rozhodnutí $x_2 = 2$. Optimální hodnoty účelových funkcí pro stav $p_2 < 2$ byly opět vypočteny pro výpočet optimální hodnoty účelové funkce v další etapě procesu.

	x_1				
p_1	0	1	2	$f_3(p_1)$	x_1
2	0,064	0,060	0,072	0,060	1

Obr. 77 Vypočtená optimální hodnota účelové funkce pro třetí etapu

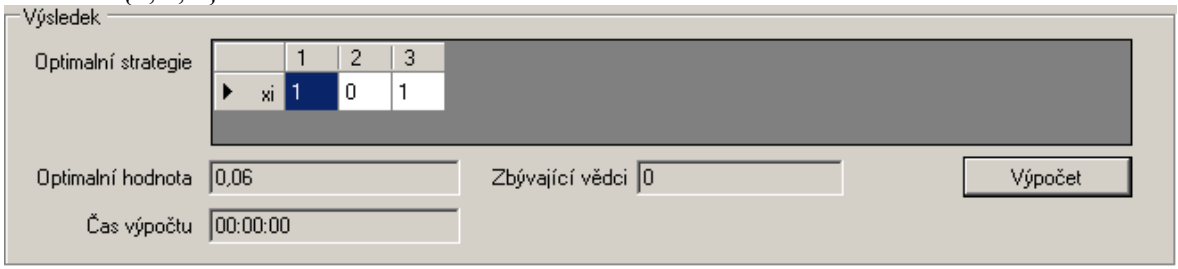
Tímto jsme vypočetli výslednou optimální hodnotu účelové funkce $f_3(p_1) = 0{,}06$. Výpočet optimální hodnoty účelové funkce pro třetí etapu procesu je znázorněn tabulkou na Obr. 77.

Tímto postupem jsme vypočetli optimální hodnotu účelové funkce zadaného příkladu, nyní zbývá vypočítat rozhodovací strategii, kterou lze dosáhnout optimální hodnoty účelové funkce. Optimální strategii vypočítáme pomocí vztahu (3.8). Postup výpočtu optimální strategie je zobrazen tabulkou na Obr. 78.

j	p_j	x_j
1	2	1
2	1	0
3	1	1

Obr. 78 Výpočet optimální strategie

Postup výpočtu optimální strategie je následující: nejdříve do tabulky zapíšeme hodnotu stavu p_1 odpovídající zadanému počtu vědců $p = 2$. Následně opíšeme rozhodnutí $x_1 = 1$ z třetí etapy procesu. Další hodnotu stavu p_2 vypočteme pomocí transformace $p_2 = p_1 - x_1$. Zbývající množství vědců v druhé etapě procesu $p_2 = 1$. Zjistíme rozhodnutí v druhé etapě procesu x_2 pro hodnotu stavu p_2. Zjištěné rozhodnutí $x_2 = 0$, proto hodnota stavu $p_3 = p_2$. Zbývající počet vědců po třetí etapě je $p_3 = 1$. Zjistíme poslední rozhodnutí $x_3 = 1$ pro hodotu stavu p_3. Tímto postupem jsme získali optimální rozhodovací strategii zadaného příkladu. Optimální rozhodovací strategie je dána posloupností rozhodnutí {1, 0, 1}.



Obr. 79 Výsledek příkladu vytváření vědeckých týmů vypočtený programovým systémem

Výsledek příkladu rozdělení zdrojů, tj. optimální hodnota účelové funkce a optimální rozhodovací strategie, vypočtený v programovém systému je zobrazen na Obr. 79. Vypočtené výsledné hodnoty se shodují s předchozím vypočteným výsledkem.

6.6 Obnova strojového parku

Úloha obnovy strojového parku je zadána takto: je dán stroj, který je charekterizován nákupní cenou $c = 10$, stářím na počátku procesu $t = 3$ a ziskovou funkcí $n(t_j)$. Zisková funkce $n(t_j)$ určuje zisk ze stroje v daném období a je závislá na stáří stroje v daném období t_j. Ve většině případů je zisková funkce nerostoucí. Hodnoty ziskové funkce jsou zobrazeny tabulkou na Obr. 80. Dále je zadána délka období $k = 4$, pro které se má stanovit obnovovací strategie. Cílem úlohy je najít optimální obnovovací strategii tj. strategii, která maximalizuje zisk z provozu stroje. Podrobněji je úloha popsána v podkapitole 3.1.6. Numerické hodnoty zadání byly pro kontrolu převzaty z [3].

t_j	$n(t_j)$
0	10
1	9
2	8
3	7
4	6
5	5
6	4
7	3
8	2
9	1

Obr. 80 Zadaná zisková funkce

Při výpočtu příkladu obnovy strojového parku byly použity vztahy (3.18) pro výpočet optimální hodnoty účelové funkce pro první etapu procesu a vztah (3.17) pro výpočet optimální hodnoty procesu pro ostatní etapy procesu.

	x_4			
t_4	0	1	$f_1(t_4)$	x_4
1	9	0	9	0
2	8	0	8	0
3	7	0	7	0
4	6	0	6	0
5	5	0	5	0
6	4	0	4	0
7	3	0	3	0
8	2	0	2	0
9	1	0	1	0

Obr. 81 Výpočet optimální hodnoty účelové funkce pro první etapu

Výpočet optimální hodnoty účelové funkce pro první etapu procesu je znázorněn tabulkou na Obr. 81, kde optimální hodnota účelové funkce $f_1(t_4) = 7$ pro $t_4 = 3$ a rozhodnutí $x_4 = 0$. Optimální hodnoty účelových funkcí pro první etapu byly vypočteny z důvodu použití v další etapě procesu.

	x_3			
t_3	0	1	$f_2(t_3)$	x_3
1	17	9	17	0
2	15	9	15	0
3	13	9	13	0
4	11	9	11	0
5	9	9	9	0; 1
6	7	9	9	1
7	5	9	9	1
8	3	9	9	1

Obr. 82 Výpočet optimální hodnoty účelové funkce pro druhou etapu

Vypočet optimální hodnoty účelové funkce pro druhou etapu procesu je znázorněn tabulkou na Obr. 82, kde optimální hodnota účelové funkce $f_2(t_3) = 13$ pro $t_3 = 3$ a rozhodnutí $x_3 = 0$. Optimální hodnoty účelových funkcí pro druhou etapu byly opět vypočteny pro výpočet optimální hodnoty účelové funkce v další etapě.

	x_2			
t_2	0	1	$f_3(t_2)$	x_2
1	24	17	24	0
2	21	17	21	0
3	18	17	18	0
4	15	17	17	1
5	14	17	17	1
6	13	17	17	1
7	12	17	17	1

Obr. 83 Výpočet optimální hodnoty účelové funkce pro třetí etapu

Výpočet optimální hodnoty účelové funkce pro třetí etapu procesu je zobrazen tabulkou na Obr. 81, kde optimální hodnota účelové funkce $f_3(t_2) = 18$ pro $t_2 = 3$ a rozhodnutí $x_2 = 0$. Optimální hodnoty účelových funkcí pro třetí etapu byly vypočteny z důvodu použití v poslední etapě procesu.

	x_1			
t_1	0	1	$f_4(t_1)$	x_1
3	24	24	24	0; 1

Obr. 84 Výpočet optimální hodnoty účelové funkce pro čtvrtou etapu

Tímto postupem jsme vypočetli výslednou optimální hodnotu účelové funkce $f_4(t_1) = 24$, pro počáteční stáří stroje $t_1 = 3$ a rozhodnutí $x_1 = 0$. Výpočet optimální hodnoty účelové funkce pro poslední etapu procesu je znázorněn tabulkou na Obr. 84.

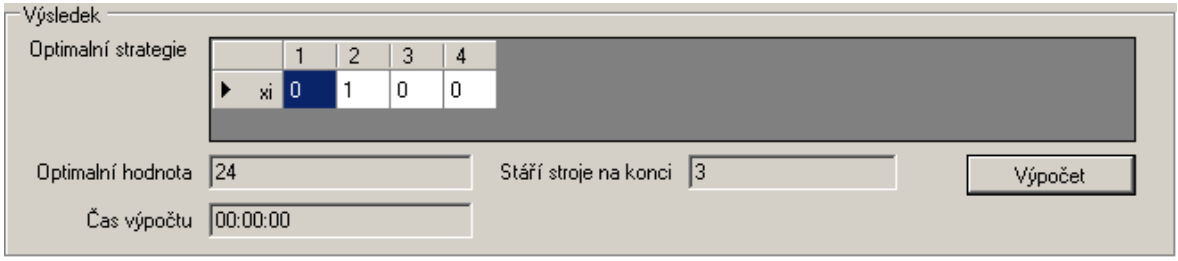
Jestliže už máme vypočtenou výslednou optimální hodnotu účelové funkce, tak zbývá vypočítat optimální rozhodovací strategii, tj. strategii, kterou lze dosáhnout optimální hodnoty účelové funkce. Optimální strategii vypočítáme pomocí vztahu (3.16). Postup výpočtu optimální strategie je zobrazen tabulkou na Obr. 85.

j	t_j	x_j
1	3	0
2	4	1
3	1	0
4	2	0

Obr. 85 Výpočet optimální strategie

Postup výpočtu optimální strategie je následující: nejdříve do tabulky zapíšeme hodnotu stavu t_1 odpovídající zadanému počátečnímu stáří stroje $t = 3$. Následně opíšeme vypočtené rozhodnutí $x_1 = 0$ ze čtvrté etapy. Další hodnotu stáří stroje t_2 vypočteme transformací $t_2 = (1 - x_1)t_1 + 1$. Stáří

stroje v druhé etapě procesu $t_2 = 4$. Zjistíme rozhodnutí v druhé etapě procesu x_2 pro hodnotu stavu t_2. Zjištěné rozhodnutí $x_2 = 1$, proto hodnota stavu $t_3 = (1 - x_1)t_1 + 1 = 1$. Stáří stroje ve třetí etapě je $t_3 = 1$. Zjistíme rozhodnutí $x_3 = 0$ pro hodnotu stavu t_3. Stáří stroje ve čtvrté etapě pro rozhodnutí x_3 je $t_4 = t_3 + 1$. Nakonec zjistíme poslední rozhodnutí x_4 pro stáří stroje $t_4 = 2$. Rozhodnutí pro čtvrtou etapu procesu je $x_4 = 0$. Tímto postupem jsme získali optimální rozhodovací strategii zadaného příkladu. Optimální rozhodovací strategie je dána posloupností rozhodnutí {0, 1, 0, 0}.

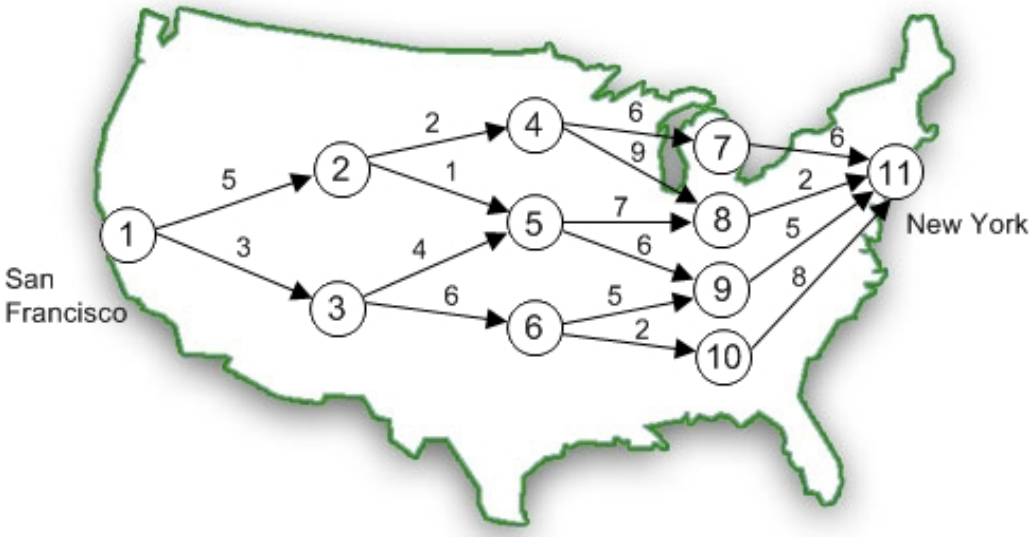


Obr. 86 Výsledek příkladu obnova strojového parku vypočtený programovým systémem

Výsledek příkladu obnova strojového parku, tj. optimální hodnota účelové funkce a optimální rozhodovací strategie, vypočtený v programovém systému, je zobrazen na Obr. 86. Vypočtené výsledné hodnoty se shodují s předchozím vypočteným výsledkem.

6.7 Průchod sítí

Úloha průchodu sítí je zadána následovně: je dán graf, kde uzly grafu jsou města a hrany grafu jsou cesty mezi městy. Hrany grafu jsou ohodnoceny podle počtu dní potřebných k uražení cesty. Teoretické základy problému průchodu sítí byly probrány v podkapitole 3.1.7. Graf zadání úlohy je zobrazen na Obr. 87.



Obr. 87 Grafické zadání úlohy průchod sítí [7]

Úkolem úlohy je najít nejkratší cestu grafem z San Francisca do New Yorku. Nejkratší cestou se rozumí cesta trvající nejmenší počet dní. Tato úloha je blíže popsána v publikaci [7], odkud jsou i převzaty numerické hodnoty zadání. Úloha je v publikaci popsána jako problém dostavníku. Zadání grafu úlohy můžeme převést do tvaru matice sousednosti se kterým umí pracovat programový systém. Zadání grafu ve tvaru matice sousednosti je zobrazeno na Obr. 88.

	1	2	3	4	5	6	7	8	9	10	11
1		5	3								
2				2	1						
3					4	6					
4							6	9			
5								7	6		
6									5	2	
7											6
8											2
9											5
10											8
11											

Obr. 88 Zadání úlohy maticí sousednosti

V matici sousednosti jsou zobrazeny délky cest z jednoho uzlu do druhého. Pokud není uvedena žádná hodnota délky cesty, tak cesta z daného uzlu do uzlu neexistuje. Při výpočtu úlohy průchodu sítí byly použity vztahy (3.20) pro výpočet optimální hodnoty délky cesty přes *n* meziuzlů, (3.21) pro výpočet optimální hodnoty délky cesty přes jeden meziuzel a (3.22) pro výpočet optimální hodnoty délky cesty přes žádný meziuzel. V první etapě budeme hledat nejkratší cestu z počátečního uzlu do koncového přes žádný meziuzel. Výpočet optimální hodnoty délky cesky je zobrazen tabulkou na Obr. 89.

l	1	2	3	4	5	6	7	8	9	10
$u_{i,11}$							6	2	5	8

Obr. 89 Výpočet optimální hodnoty délky cesty přes žádný meziuzel

Při výpočtu nejkratší cesty přes žádný meziuzel nebyla zjištěna optimální hodnota délky cesty z počátečního uzlu, protože neexistuje přímá cesta z počátečního do koncového uzlu. V rámci výpočtu první etapy byly zjištěny hodnoty délky cesty, které budou použity v další etapě. Byly však zjištěny optimální hodnoty délky cesty z uzlů {7, 8, 9, 10} do koncového uzlu. Výpočet optimální hodnoty délky cesty v druhé etapě je zobrazen tabulkou na Obr. 90.

l	1	2	3	4	5	6
$t_{i,7} + u_{7,11}$				6 + 6		
$t_{i,8} + u_{8,11}$				9 + 2	7 + 2	
$t_{i,9} + u_{9,11}$					6 + 5	5 + 5
$t_{i,10} + u_{10,11}$						2 + 8
$u_{i,11}$				11	9	10
j				8	8	9; 1

Obr. 90 Výpočet optimální hodnoty délky cesty přes jeden meziuzel

Při výpočtu nejkratší cesty přes jeden meziuzel nebyla opět zjištěna optimální hodnota délky cesty z počátečního do koncového uzlu. Hodnota nebyla zjištěna proto, protože neexistuje cesta z počátečního do koncového uzlu přes jeden meziuzel. V rámci výpočtu byly ale zjištěny optimální hodnoty délky cesty z uzlů {4, 5, 6} do koncového uzlu. Výpočet optimální hodnoty délky cesty ve třetí etapě je znázorněn tabulkou na Obr. 91.

I	1	2	3
$t_{i,4} + u_{4,11}$		2 + 11	
$t_{i,5} + u_{5,11}$		1 + 9	4 + 9
$t_{i,6} + u_{6,11}$			6 + 10
$u_{i,11}$		10	13
j		5	5

Obr. 91 Výpočet optimální hodnoty délky cesty přes dva meziuzly

Při výpočtu nejkratší cesty přes dva meziuzly nebyla opět zjištěna optimální hondnota délky cesty z počátečního do koncového uzlu. Hodnota nebyla zjištěna proto, protože neexistuje cesta z počátečního do koncového uzlu přes dva meziuzly. V rámci výpočtu byly ale zjištěny optimální hodnoty délky cesty z uzlů {2, 3} do koncového uzlu. Výpočet optimální hodnoty délky cesty ve čtvrté, tj. poslední etapě je znázorněn tabulkou na Obr. 92.

I	1
$t_{i,2} + u_{2,11}$	5 + 10
$t_{i,3} + u_{3,11}$	3 + 13
$u_{i,11}$	15
j	2

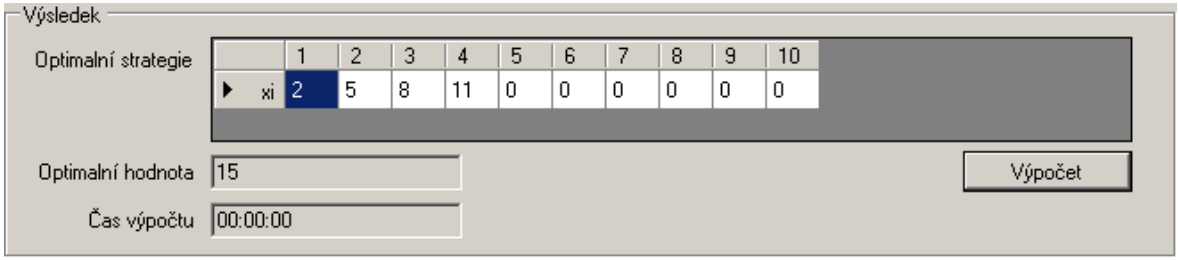
Obr. 92 Výpočet optimální hodnoty délky cesty přes tři meziuzly

Při výpočtu nejkratší cesty přes tři meziuzly byla vypočtena optimální hodnoty délky cesty z počátečního do koncového uzlu. Optimální hodnota délky cesty je $u_{1,11} = 15$. Tímto jsme vypočetli délku nejkratší cesty z počátečního do koncového uzlu, ale ještě zbývá vypočítat optimální strategii pro průchod sítí. Optimální strategii vypočítáme pomocí vztahu $p_{i+1} = j$, tzn. že hodnota nového stavu je rovna hodnotě rozhodnutí. Postup výpočtu optimální strategie je zobrazen tabulkou na Obr. 93.

I	p_i	j
1	1	2
2	2	5
3	5	8
4	8	11

Obr. 93 Výpočet optimální strategie

Při výpočtu optimální strategie se postupuje následovně: hodnota stavu p_1 se rovná číslu počátečního uzlu, tedy $p_1 = 1$. Hodnota prvního optimálního rozhodnutí j je rovna hodnotě rozhodnutí vypočítané v poslední etapě, tedy $j = 2$. Pomocí vztahu $p_2 = j$ můžeme vypočítat hodnotu druhého stavu $p_2 = 2$. Tímto jsme se dostali do uzlu číslo 2 a hledáme nejkratší cestu z uzlu 2 do koncového uzlu 11. Optimální hodnota délky cesty z druhého uzlu je vypočítána v třetí etapě. V třetí etapě bylo vypočítáno pro druhý uzel optimální rozhodnutí j = *5*. Pomocí stejného vztahu bychom opět dostali hodnotu stavu $p_3 = j = 5$. Takto bycho postupovali tak dlouho dokud bychom se nedostali do koncového uzlu. Výpočtem jsme získali optimální strategii jako posloupnost rozhodnutí {2, 5, 8, 11}.



Obr. 94 Výsledek úlohy průchod sítí vypočtený pomocí programového systému

Výsledek příkladu průchodu sítí, tj. optimální hodnota délky cesty a optimální rozhodovací

strategie, vypočtený v programovém systému, je zobrazen na Obr. 94. Vypočtené výsledné hodnoty se shodují s předchozím vypočteným výsledkem. U vypočtené optimální strategie je vidět, že obecně může mít strategie až *N*-1 rozhodnutí, kde *N* je počet uzlů grafu.

6.8 Spolehlivost zařízení

Zadání úlohy je následující: mějme elektronické zařízení se třemi sériově zapojenými prvky. Zařízení je rozděleno do stupňů tak, že každý stupěn zařízení má jeden prvek. Každý z prvnů má různou spolehlivost a je možné spolehlivost daného stupně zařízení zvýšit zapojením paralelních prvků. Dále je prvek zařízení reprezentován svojí cenou. Všechy spolehlivosti a ceny prvků jsou zobrazeny tabulkou na Obr. 95.

j	1	2	3
g_j	0,7	0,5	0,3
c_j	5	3	2

Obr. 95 Zadání úlohy spolehlivosti zařízení

Úkolem je maximalizovat spolehlivost zařízení s ohledem na omezené prostředky. K dispozici jsou finanční prostředky o hodnotě $M = 10$. Teoretické základy problému spolehlivosti zařízení byly uvedeny v rámci podkapitoly 3.1.8.

Při výpočtu úlohy spolehlivosti zařízení byly použity vztahy (3.26) pro výpočet optimální hodnoty účelové funkce pro j-tou etapu, (3.27) pro výpočet optimální hodnoty účelové funkce pro první etapu a (3.25) pro výpočet transformace stavu.

	x_3							
p_3	0	1	2	3	4	5	$f_1(p_3)$	x_3
0	0,3						0,3	0
1	0,3						0,3	0
2	0,3	0,6					0,6	1
3	0,3	0,6					0,6	1
4	0,3	0,6	0,9				0,9	2
5	0,3	0,6	0,9				0,9	2
6	0,3	0,6	0,9	1			1	3
7	0,3	0,6	0,9	1			1	3
8	0,3	0,6	0,9	1	1		1	3; 4
9	0,3	0,6	0,9	1	1		1	3; 4
10	0,3	0,6	0,9	1	1	1	1	3; 4; 5

Obr. 96 Výpočet optimální hodnoty účelové funkce pro první etapu

Postup výpočtu optimální hodnoty účelové funkce pro první etapu je znázorněn tabulkou na Obr. 96. Optimální hodnota účelové funkce pro počáteční hodnotu finančních prostředků je $f_1(p_3) = 1$ s rozhodnutím $x_3 = 3; 4; 5$. Optimální hodnoty účelové funkce byly vypočítány pro další etapu procesu.

p_2	0	1	2	3	$f_2(p_2)$	x_2
0	0,15				0,15	0
1	0,15				0,15	0
2	0,3				0,3	0
3	0,3	0,3			0,3	0; 1
4	0,45	0,3			0,45	0
5	0,45	0,6			0,6	1
6	0,5	0,6	0,3		0,6	1
7	0,5	0,9	0,3		0,9	1
8	0,5	0,9	0,6		0,9	1
9	0,5	1	0,6	0,3	1	1
10	0,5	1	0,9	0,3	1	1

Obr. 97 Výpočet optimální hodnoty účelové funkce pro druhou etapu

Postup výpočtu optimální hodnoty účelové funkce pro druhou etapu je zobrazen v tabulce na Obr. 97. Optimální hodnota účelové funkce pro počáteční hodnotu finančních prostředků je $f_2(p_2) = 1$ s rozhodnutím $x_2 = 1$. Optimální hodnoty účelové funkce pro jiné hodnoty stavů byly vypočítány pro další etapu procesu.

	x_1				
p_1	0	1	2	$f_3(p_1)$	x_1
0	0,105			0,105	0
1	0,105			0,105	0
2	0,21			0,21	0
3	0,21			0,21	0
4	0,315			0,315	0
5	0,42	0,15		0,42	0
6	0,42	0,15		0,42	0
7	0,63	0,3		0,63	0
8	0,63	0,3		0,63	0
9	0,7	0,45		0,7	0
10	0,7	0,6	0,15	0,7	0

Obr. 98 Výpočet optimální hodnoty účelové funkce pro třetí etapu

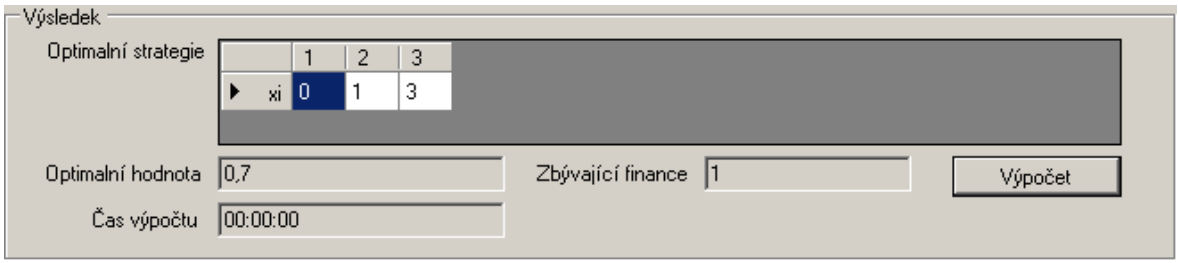
Výpočet optimální hodnoty účelové funkce pro poslední etapu je znázorněn tabulkou na Obr. 98. Vypočtená optimální hodnota účelové funkce je $f_3(p_1) = 0,7$ a je to i výsledná optimální hodnota zadaného příkladu. Rozhodnutí vypočtené v poslední etapě je $x_1 = 0$, tzn. v prvním stupni elektronického zařízení nezavádět žádný paralelní prvek. V poslední etapě stačilo vypočítat optimální hodnotu pouze pro počáteční stav $p_1 = 10$, ale zde je opět ukázáno jak lze vypočítat optimální hodnoty účelových funkci pro úlohy s menší hodnotou počátečního stavu p_1. Tímto jsme spočítali optimální hodnotu a zbývá vypočítat optimální strategii.

j	p_j	x_j
1	10	0
2	10	1
3	7	3

Obr. 99 Výpočet optimální strategie

Výpočet optimální strategie je zobrazen tabulkou na Obr. 99. Při výpočtu optimální strategie se postupuje následovně: hodnota stavu p_1 se rovná množství funančních prostředků, tedy $p_1 = 10$. Hodnota prvního optimálního rozhodnutí x_1 je rovna hodnotě rozhodnutí vypočítané v poslední etapě, tedy $x_1 = 0$. Pomocí vztahu $p_2 = p_1 - x_1$ můžeme vypočítat hodnotu druhého stavu $p_2 = 10$. Další hodnotu rozhodnutí získáme z druhé etapy pro stav p_2. Optimální rozhodnutí zjištěné v druhé etapě je $x_2 = 1$. Dále vypočítáme hodnotu třetího stavu p_3 pomocí vztahu $p_3 = p_2 - x_2 = 7$. Nakonec získáme

hodnotu posledního rozhodnutí $x_3 = 3$ pro hodnotu stavu $p_3 = 7$. Tímto jsme získali optimální strategii jako posloupnost rozhodnutí {0, 1, 3}.



Obr. 100 Výsledek úlohy spolehlivosti zařízení vypočtený programovým systémem

Výsledek vypočtený programovým systémem je zobrazen na Obr. 100. Výsledek se shoduje s předchozím vypočteným výsledkem.

6.9 Využití pracovních sil

Úloha je zadána následovně: firma má zjištěné požadované počty zaměstnanců na následující tři měsíce a chce stanovit počty zaměstnanců na tyto měsíce s ohledem na minimalizaci nákladů. Náklady jsou dvojího druhu: náklady na nabírání a propuštění zaměstnanců a náklady na nepotřebné zaměstnance. Napočátku má firma počet zaměstnanců roven $m_0 = 50$. Požadované počty zaměstnanců jsou dány tabulkou na Obr. 101.

j	1	2	3
m_j	45	40	50

Obr. 101 Zadání úlohy využití pracovních sil

Náklady na nabírání a propuštění jednoho zaměstnance jsou c = 20 peněžních jednotek. Náklady na jednoho nepotřebného zaměstnance jsou d = 300 peněžních jednotek. Úkolem je stanovit optimální hodnotu účelové funkce a optimální rozhodovací strategii. Teoretické základy tohoto typu úlohy byly probrány v rámci podkapitoly 3.1.9. Optimální hodnota účelové funkce pro j-tou etapu se počítá pomocí vztahu (3.30). Pomocí vztahu (3.31) lze vypočítat optimální hodnotu účelové funkce pro první etapu. Výpočet nové hodnoty stavu se provádí transformací pomocí vztahu (3.29).

	x_3		
p_3	50	$f_1(p_3)$	x_3
40	2000	2000	50
41	1620	1620	50
42	1280	1280	50
43	980	980	50
44	720	720	50
45	500	500	50
46	320	320	50
47	180	180	50
48	80	80	50
49	20	20	50
50	0	0	50

Obr. 102 Výpočet optimální hodnoty účelové funkce pro první etapu

Výpočet optimální hodnoty účelové funkce pro první etapu je znázorněn tabulkou na Obr. 102. Výpočet se provádí pouze pro přípustné hodnoty stavů p_3. Přípustné stavy jsou dány intervalem (40 - 50*)*, kde 40 je požadovaný počet zaměstnanců v další etapě a 50 je maximální počet zaměstnanců ve všech období včetně počátečního počtu zaměstnanců. Optimální hodnota účelové funkce pro první etapu je $f_1(p_3) = 0$ pro hodnotu stavu $p_3 = 50$ a rozhodnutí je $x_3 = 50$.

	x_2												
p_2	40	41	42	43	44	45	46	47	48	49	50	$f_2(p_2)$	x_2
45	2500	2240	2060	1960	1940	2000	2140	2360	2660	3040	3500	1940	44
46	2720	2420	2200	2060	2000	2020	2120	2300	2560	2900	3320	2000	44
47	2980	2640	2380	2200	2100	2080	2140	2280	2500	2800	3180	2080	45
48	3280	2900	2600	2380	2240	2180	2200	2300	2480	2740	3080	2180	45
49	3620	3200	2860	2600	2420	2320	2300	2360	2500	2720	3020	2300	46
50	4000	3540	3160	2860	2640	2500	2440	2460	2560	2740	3000	2440	46

Obr. 103 Výpočet optimální hodnoty účelové funkce pro druhou etapu

Výpočet optimální hodnoty účelové funkce pro druhou etapu je zobrazen na Obr. 102. Výpočet se provádí pouze pro přípustné hodnoty stavů p_2. Přípustné stavy jsou dány intervalem (45 - 50), kde 45 je požadovaný počet zaměstnanců v další etapě a 50 je maximální počet zaměstnanců. Rozhodnutí mohou nabývat hodnot z intervalu (40 - 50), kde 40 je požadovaný počet zaměstnanců v této etapě a 50 je maximální počet zaměstnanců. Optimální hodnota účelové funkce pro druhou etapu je $f_2(p_2) = 2440$ pro hodnotu stavu $p_2 = 50$ a rozhodnutí je $x_2 = 46$.

	x_1							
p_1	45	46	47	48	49	50	$f_3(p_1)$	x_1
50	2440	2620	2860	3160	3520	3940	2440	45

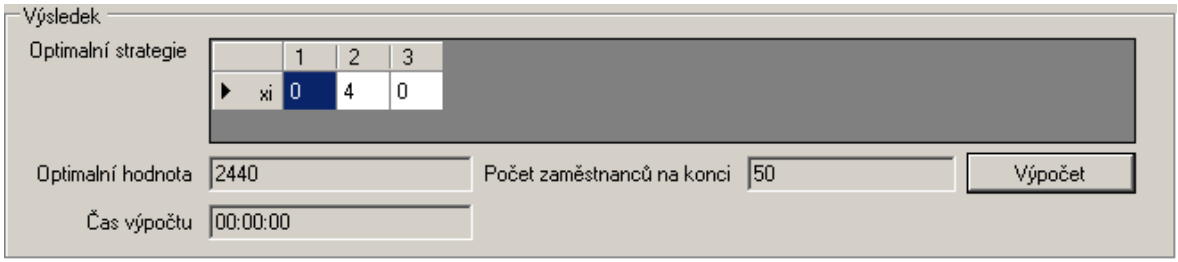
Obr. 104 Výpočet optimální hodnoty účelové funkce pro třetí etapu

Výpočet výsledné optimální hodnoty účelové funkce je zobrazen v tabulce na Obr. 104. Výsledná optimální hodnota účelové funkce je $f_3(p_1) = 2440$ a rozhodnutí je $x_1 = 45$. Jelikož už máme vypočtenou optimální hodnotu účelové funkce, tak můžeme vypočítat optimální rozhovací strategii.

j	p_j	x_j
1	50	45
2	45	44
3	44	50

Obr. 105 Výpočet optimální strategie

Výpočet optimální strategie je zobrazen tabulkou na Obr. 105. Při výpočtu optimální strategie nejdříve zapíšeme do tabulky hodnotu stavu p_1, která je rovna počtu zaměstnanců na počátku m_0, tedy $p_1 = 50$. Hodnota prvního optimálního rozhodnutí x_1 je rovna hodnotě rozhodnutí vypočítané v poslední etapě, tedy $x_1 = 45$. Pomocí vztahu $p_2 = x_1$ můžeme vypočítat hodnotu druhého stavu $p_2 = 45$. Další hodnotu rozhodnutí získáme z druhé etapy pro stav p_2. Optimální rozhodnutí zjištěné v druhé etapě je $x_2 = 44$. Dále vypočítáme hodnotu třetího stavu p_3 pomocí vztahu $p_3 = x_2 = 44$. Nakonec získáme hodnotu posledního rozhodnutí $x_3 = 50$ pro hodnotu stavu $p_3 = 44$. Tímto jsme získali optimální strategii jako posloupnost rozhodnutí {45, 44, 50}.



Obr. 106 Výsledek úlohy využití pracovních sil vypočtený programovým systémem

Výsledek vypočtený programovým systémem je zobrazen na Obr. 106. Vypočtená optimální hodnota účelové funkce je shodná s optimální hodnotou předešle vypočítanou. Optimální strategie se liší od již vypočítané důsledkem jiné reprezentace rozhodnutí. Při předchozím výpočtu jsme

rozhodnutí chápali jako počet pracovníků v daném období, programový systém chápe rozhodnutí jako o kolik více se má najmout zaměstnanců pro dané období, tzn. rozhodnutí v programovém systému $x_1 = 0$ odpovídá požadovanému počtu zaměstnanců 45. Tímto byla v programovém systému dosažena úspornější reprezentace rozhodnutí. Pro přepočet lze použít vztah $n_j = x_j + m_j$, kde n_j je výsledný počet zaměstnanců v j-tém období, x_j je j-té rozhodnutí vypočtené programovým systémem a m_j je požadovaný počet zaměstnanců. Pomocí tohoto vztahu dostaneme výslednou optimální strategii jako posloupnost rozhodnutí {45, 44, 50}, která je shodná se strategií dříve vypočtenou.

6.10 Sklad náhradních dílu

Zadání úlohy je následující: jsou dány tři náhradní díly, kdy každý má svou očekávanou hodnotu poptávky po náhradním dílu n_j, svou velikost a_j a pokutu pokud není díl na skladě π_j. Náhradní díly jsou zobrazeny v tabulce na Obr. 107.

j	1	2	3
π	800	600	1300
n_j	4	2	1
a_j	1	2	2

Obr. 107 Zadání úlohy skladu náhradních dílů

Sklad má kapacitu d = 10 prostorových jednotek. Úkolem úlohy je stanovit takový počet jednotlivých dílů na skladě, který minimalizuje očekávanou hodnotu finančních ztrát všech dílů a zároveň součet všech velikostí náhradních dílů na skladě nesmí být větší než kapacita skladu. Úloha sklad náhradních dílů byla probrána v podkapitole 3.2.1. Numerické hodnoty zadání úlohy byly pro kontrolu převzaty z [3]. Výpočet úlohy se provádí podle vztahů (3.36) pro optimální hodnotu účelové funkce v j-té etapě, (3.37) pro optimální hodnotu účelové funkce pro první etapu a (3.35) pro výpočet transformace stavu.

	x_3							
p_3	0	1	2	3	4	5	$f_1(p_3)$	x_3
0	1300						1300	0
1	1300						1300	0
2	1300	478,24					478,24	1
3	1300	478,24					478,24	1
4	1300	478,24	134,73				134,73	2
5	1300	478,24	134,73				134,73	2
6	1300	478,24	134,73	30,34			30,34	3
7	1300	478,24	134,73	30,34			30,34	3
8	1300	478,24	134,73	30,34	5,65		5,65	4
9	1300	478,24	134,73	30,34	5,65		5,65	4
10	1300	478,24	134,73	30,34	5,65	0,9	0,9	5

Obr. 108 Výpočet optimální hodnoty účelové funkce pro první etapu

Výpočet optimální hodnoty účelové funkce první etapy je znázorněn tabulkou na Obr. 108. Všechny hodnoty účelových funkcí jsou v tabulce zaokrouhleny na dvě desetinná místa. Optimální hodnota účelové funkce $f_1(p_3)$ = 0,9 pro hodnotu stavu p_3 = 10 a rozhodnutí x_3 = 5. Vypočtené optimální hodnoty budou použity pro výpočet v další etapě.

	x_2							
p_2	0	1	2	3	4	5	$f_2(p_2)$	x_2
0	2500						2500	0
1	2500						2500	0
2	1678,24	1981,2					1678,24	0
3	1678,24	1981,2					1678,24	0
4	1334,73	1159,44	1624,8				1159,44	1
5	1334,73	1159,44	1624,8				1159,44	1
6	1230,34	815,93	803,05	1430,81			803,05	2
7	1230,34	815,93	803,05	1430,81			803,05	2
8	1205,65	711,54	459,53	609,05	1345,08		459,53	2
9	1205,65	711,54	459,53	609,05	1345,08		459,53	2
10	1200,9	686,85	355,14	265,54	523,33	1313,49	265,54	3

Obr. 109 Výpočet optimální hodnoty účelové funkce pro druhou etapu

Výpočet optimální hodnoty účelové funkce první etapy je znázorněn tabulkou na Obr. 109. Hodnoty účelových funkcí jsou v tabulce zaokrouhleny na dvě desetinná místa. Optimální hodnota účelové funkce $f_2(p_2) = 265{,}54$ pro hodnotu stavu $p_2 = 10$ a rozhodnutí $x_3 = 3$. Vypočtené optimální hodnoty budou použity v další etapě.

	x_1												
p_1	0	1	2	3	4	5	6	7	8	9	10	$f_3(p_1)$	x_1
10	3465	2874	2147	1881	1428	1487	1315	1746	1705	2509	2503	1315,79	6

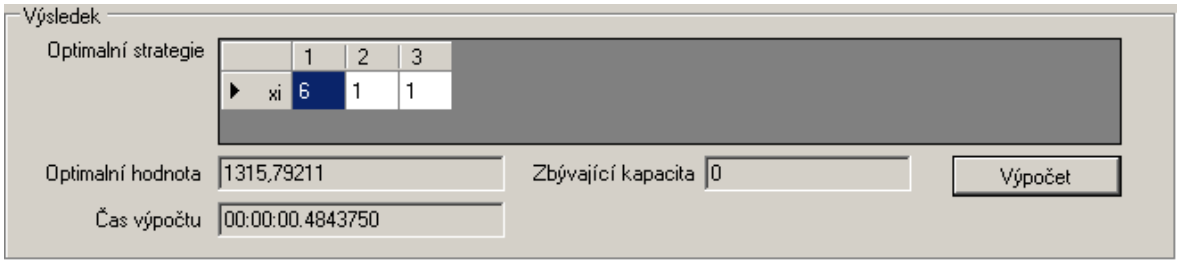
Obr. 110 Výpočet otimální hodnoty účelové funkce pro třetí etapu

Výpočet výsledné optimální hodnoty účelové funkce je zobrazen v tabulce na Obr. 110. Hodnoty účelových funkcí pro různá rozhodnutí x_1 byly z důvodu zmenšení velikosti tabulky zaokrouhleny. Výsledná optimální hodnota účelové funkce zaokrouhlená na dvě desetinná místa je $f_3(p_1) = 1315{,}79$ a rozhodnutí je $x_1 = 6$. Tímto jsme vypočetli optimální hodnotu účelové funkce a nyní zbývá vypočítat optimální rozhodovací strategii.

j	p_j	x_j
1	10	6
2	4	1
3	2	1

Obr. 111 Výpočet optimální strategie

Výpočet optimální strategie je zobrazen tabulkou na Obr. 111. Při výpočtu optimální strategie nejdříve zapíšeme do tabulky hodnotu stavu p_1, která je rovna kapacitě skladu d, tedy $p_1 = 10$. Hodnota prvního optimálního rozhodnutí x_1 je rovna hodnotě rozhodnutí vypočítané v poslední etapě, tedy $x_1 = 6$. Pomocí vztahu $p_2 = p_1 - a_1 x_1$ můžeme vypočítat hodnotu druhého stavu $p_2 = 4$. Další hodnotu rozhodnutí získáme z druhé etapy pro stav p_2. Optimální rozhodnutí zjištěné v druhé etapě je $x_2 = 1$. Dále vypočítáme hodnotu třetího stavu p_3 pomocí vztahu $p_3 = p_2 - a_2 x_2$. Nakonec získáme hodnotu posledního rozhodnutí $x_3 = 1$ pro hodnotu stavu $p_3 = 2$. Tímto jsme získali optimální strategii jako posloupnost rozhodnutí {6, 1, 1}.



Obr. 112 Výsledek úlohy skladu náhradních dílů vypočtený programovým systémem

Výsledek příkladu skladu náhradních dílů, tj. optimální hodnota účelové funkce a optimální rozhodovací strategie, vypočtený v programovém systému, je zobrazen na Obr. 112. Vypočtené výsledné hodnoty se shodují s předchozím vypočteným výsledkem.

6.11 Vyhrávání v Las Vegas

Příklad je zadán takto: je dána hra, ve které je pravděpodobnost výhry $P_W = 0{,}66$ a pravděpodobnost prohry $P_L = 0{,}34$, dále je ve hře možné vsadit jakýkoliv počet žetonů a v případě výhry vsázející obdrží dvojnásobný počet žetonů než vsadil. Počáteční počet žetonů je $C_1 = 3$ a požadovaný počet žetonů po poslední sázce je $C_2 = 4$. Pro získání požadovaného počtu žetonů má vsázející tři sázky. Úkolem je maximalizovat pravděpodobnost získání požadovaného počtu žetonů po třech sázkách. Více informací o úloze vyhrávání v Las Vegas je uvedeno v podkapitole 3.2.2.

Při výpočtu se postupuje podle vztahů (3.41) pro optimální hodnotu účelové funkce v j-té etapě, (3.39) pro optimální hodnotu účelové funkce před první etapou a (3.40) pro výpočet transformace stavu.

	p_4												
	0	1	2	3	4	5	6	7	8	9	10	11	12
$f_0(p_4)$	0	0	0	0	1	1	1	1	1	1	1	1	1

Obr. 113 Vypočtené hodnoty účelových funkcí pro nultou etapu

Vypočtené hodnoty účelových funkcí pro nultou etapu jsou zobrazeny tabulkou na Obr. 113. Hodnoty účelových funkcí pro nultou etapu byly počítány pro hodnoty přípustných stavů p_4 a byly vypočteny pro další etapu výpočtu.

	x_2								
p_2	0	1	2	3	4	5	6	$f_2(p_2)$	x_2
0	0							0	0
1	0	0						0	0
2	0	0	0,66					0,66	2
3	0	0,66	0,66	0,66				0,66	1
4	1	0,66	0,66	0,66	0,66			1	0
5	1	1	0,66	0,66	0,66	0,66		1	0
6	1	1	1	0,66	0,66	0,66	0,66	1	0

Obr. 114 Výpočet optimální hodnoty účelové funkce pro první etapu

Výpočet optimální hodnoty účelové funkce pro první etapu je zobrazen tabulkou na Obr. 114. Optimální hodnota účelové funkce pro první etapu je $f_1(p_2) = 0{,}66$ pro hodnotu stavu $p_2 = 3$ a rozhodnutí $x_2 = 1$. Optimální hodnoty účelových funkcí budou použity v další etapě výpočtu.

	x_1					
p_1	0	1	2	3	$f_2(p_1)$	x_1
3	0,66	0,88	0,66	0,66	0,88	1

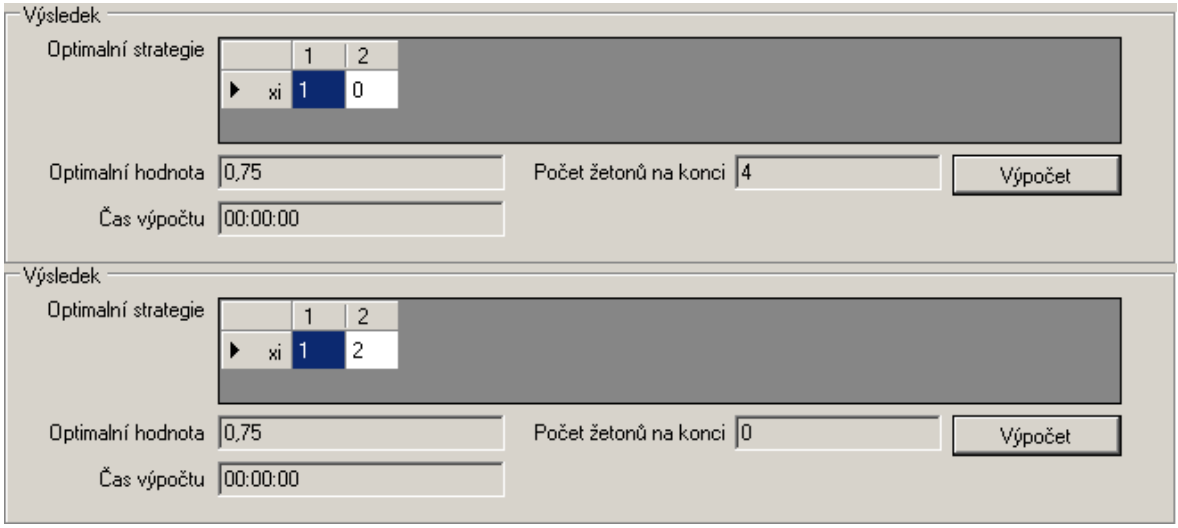
Obr. 115 Výpočet optimální hodnoty účelové funkce pro druhou etapu

Výpočet optimální hodnoty účelové funkce druhé etapy je zobrazen tabulkou na Obr. 115. Optimální hodnota účelové funkce pro druhou etapu je $f_2(p_1) = 0,88$ pro hodnotu stavu $p_1 = 3$ a rozhodnutí $x_1 = 1$. Tímto byla vypočtena výsledná optimální hodnota účelové funkce, protože jde o poslední etapu výpočtu. Optimální strategie není jednoznačně určitelná, protože závisí na působení náhodné proměnné.

j	r_{j-1}	p_j	x_j
1		3	1
2	0	2	2
2	1	4	0

Obr. 116 Výpočet optimální strategie

Výpočet optimální strategie je zobrazen tabulkou na Obr. 116. Při výpočtu optimální strategie nejdříve zapíšeme do tabulky hodnotu stavu p_1, která je rovna počtu žetonů na počátku, tedy $p_1 = 3$. Hodnota prvního optimálního rozhodnutí x_1 je rovna hodnotě rozhodnutí vypočítané v poslední etapě, tedy $x_1 = 1$. Pomocí vztahu $p_2 = p_1 + r_1 x_1 - (1 - r_1) x_1$ můžeme vypočítat hodnotu druhého stavu $p_2 = 2$ pro náhodnou proměnnou $r_1 = 0$ a hodnotu druhého stavu $p_2 = 4$ pro náhodnou proměnnou $r_1 = 1$. Další hodnotu rozhodnutí získáme z druhé etapy pro oba stavy p_2. Optimální rozhodnutí zjištěné v druhé etapě je $x_2 = 2$ pro $p_2 = 2$ a $x_2 = 0$ pro $p_2 = 4$. Tímto jsme získali optimální strategii jako posloupnost rozhodnutí v závislosti na hodnotě stavu p_2 {1, 2}; {1, 0}.



Obr. 117 Výsledek úlohy vyhrávání v Las Vegas vypočtený programovým systémem

Na Obr. 117 je zobrazen výsledek vypočtený programovým systémem. Při výpočtu optimální strategie se využívá simulace náhodné proměnné, proto výsledná strategie nemusí být při každém výpočtu stejná na rozdíl od optimální hodnoty účelové funkce. Výsledek vypočtený programovým systémem je shodný s výsledkem již dříve vypočítaným.

7 ZÁVĚR

Práce se zabývala optimalizací pomocí dynamického programování. V teoretické části práce bylo rozebráno dynamické programování jako přístup k optimalizaci diskrétních víceetapových rozhodovacích procesů a to deterministických i stochastických. Dále v rámci teoretické části práce byly uvedeny některé systémy, zabývající se řešením úloh dynamického programování. Poslední teoretická část práce je teoretický rozhor úloh dynamického programování implementovaných v programovém systému. V rámci rozboru byly sestaveny rovnice pro výpočet jednotlivých typů deterministických i stochastických úloh.

Praktická část práce se zabývá převážně návrhem a vytvořením programového systému. Hlavní uplatnění bude mít programový systém při výuce předmětu Optimalizace. Programový systém byl navržen tak, aby umožňoval snadné doprogramování dalších úloh. Toho je dosaženo tak, že část programu sloužící pro výpočet úloh je univerzální a dokáže počítat jakoukoliv úlohu, která splňuje určitou podmínku. Podmínkou je, že úloha musí být potomkem abstraktní úlohy a musí mít implementované metody pro výpočet dané úlohy. Systém byl vytvořem ve vývojovém prostředí Visual Studio 2010 pomocí programovácího jazyka C#.

V systému byly implementovány vybrané úlohy dynamického programování a to deterministické i stochastické. Dále byla v systému vytvořena univerzální úloha, která umožňuje zadat charakter úlohy rovnicemi. V rámci tvorby systému byly vytvořeny dva výpočetní režimy; režim kalkulátor pro rychlý výpočet a jednoduché zobrazení výsledků a tabulkový režim pro zobrazení postupu řešení formou tabulky. Dalšími funkcemi systému jsou načítaní a ukládaní zadání úloh a ukládaní výsledků a průběhu výpočtu.

Součástí práce bylo i zpracování jednoduchého návodu pro ovládání systému, návod pro řešení úloh a návod pro přidávání nových úloh do systému. Poslední část práce je věnována ověření funkčnosti systému. Funkčnost systému byla ověřena vypočtením příkladů pro každý typ úloh, které byly implementovány v programovém systému a porovnáním výsledků s výsledkem vypočteným pomocí programového systému.

SEZNAM POUŽITÉ LITERATURY

[1] BELLMAN, R. *Dynamic Programming*. Princeton, N. J.: Princeton University Press, 1957.

[2] KLAPKA, J. DVOŘÁK, J. *Úvod do operační a systémové analysy*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2002.

[3] KUDLÁČEK, V. KLAPKA, J. *Aplikovaná matematika IV dynamické programování a operační analýza*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechnická, 1972.

[4] LAŠČIAK, A. a kol. *Dynamické modely*. Bratislava, Alfa/SNTL, 1985.

[5] TAHA, H. *Operations Research: An Introduction*. New Jersey, Pearson Education, Inc., 2003.

[6] HILLIER, F. LIEBERMAN, G. *Introduction to Mathematical Programming*. New York, McGraw-Hill, Inc., 1990.

[7] TURBAN, E. MEREDITH, J. *Fundamentals of management science*. Boston, Irwin, Inc., 1991.

[8] ZAŇKA, T. *Problémy a algoritmy* [online]. [cit. 2011-4-21]. Dostupné z: <http://taz.mystik.cz/Skola/PAA/batoh>

[9] ROLFE, T. *Eastern Washington University* [online]. [cit. 2011-4-21]. Dostupné z: <http://penguin.ewu.edu/~trolfe/>

[10] ČERMÁK, O. *OndřejČermák.info*. [online]. [cit. 2011-4-21]. Dostupné z: <http://ondrejcermak.info/programovani/lua-x36paa-3-uloha-vetve-a-hranice-dynamicke-programovani/>

[11] JENSEN, P. *The University of Texas at Austin*. [online]. [cit. 2011-4-22]. Dostupné z: <http://www.me.utexas.edu/~jensen/ORMM/computation/>

[12] WEINER, V. *Řešení problému obchodního cestujícího pomocí PC*. [online]. [cit. 2011-4-22]. Dostupné z: <http://vlastikw.110mb.com/skola/cvutfd/cvutfd.php>

[13] WEINER, V. *Řešení problému obchodního cestujícího pomocí PC*. Praha: České vysoké učení technické v Praze, Fakulta dopravní, 2008.

[14] TWEDDLE , A. *Reusable dynamic programming with C# generics*. [online]. [cit. 2011-4-22]. Dostupné z: <http://www.codeproject.com/KB/cs/DynamicProgrammingGeneric.aspx>

[15] KÁTAI, Z. CSIKI, A. *Automated dynamic programming*. Marosvásárhely: Sapientia Hungarian University of Transylvania, Department of Mathematics and Informatics, 2009.

[16] LEW, A. MAUCH, H. *Dynamic programming a computational tool*. Berlin: Springer, 2007.

[17] SCOTT ALLEN, K. *An Eval Function for C# using JScript.NET (JavaScript)*. [online]. [cit. 2011-5-14]. Dostupné z: <http://odetocode.com/code/80.aspx>

[18] JACQUES, M. *An extensible math expression parser with plug-ins*. [online]. [cit.2011-5-14]. Dostupné z: <http://www.codeproject.com/KB/recipes/MathieuMathParser.aspx>