VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta strojního inženýrství

Distribuované optimalizační problémy

HABILITAČNÍ PRÁCE

Ing. Jan Roupec, Ph.D.

Brno, březen 2016.

OBSAH

1 ÚVOD

V průběhu předchozí stovky let došlo k několika převratným událostem, které nevratně změnily způsob života lidí na Zemi a v podstatě celý svět jako takový. Mnoho těchto událostí má svůj původ v rozvoji vědy a techniky, značná část těchto přelomových událostí a změn byla vyvolána vznikem, vývojem a masovým používáním výpočetní techniky. Masivní zlevnění počítačů, které následovalo zejména po rozšíření využití mikroprocesorů, pak způsobilo rozšíření počítačů do řady dříve nemyslitelných oblastí, a také vývoj zcela nových technických zařízení (a technických oborů a také nových profesí), které by bez mikropočítačů nemohly existovat. Jednou z nejdůležitějších oblastí byly digitální komunikace, kde za dosavadní vyvrcholení lze považovat prakticky všudypřítomnou možnost konektivity k síti Internet. Tato (dnes již) samozřejmost výrazně ovlivňuje způsob provádění mnoha činností. S ohledem na možnost snadné realizace vzájemné komunikace nejrůznějších zařízení je současným trendem vývoj distribuovaných (a přitom často hierarchicky uspořádaných) systémů. Až čas ukáže, zda tento vývoj bude pro lidstvo představovat čistý přínos či zda s sebou nese i nějaká negativa. V každém případě se ale v současné době silně zvyšuje rozhodování za pomoci počítačové podpory. Často se dokonce jedná o rozhodnutí, které za uživatele učinil výrobce či programátor a vliv uživatele na nastalou situaci je nepatrný. Pro podporu „správných rozhodnutí“ (autor si uvědomuje problematičnost tohoto pojmu) je nutné mít k dispozici dostatek informací, znát varianty rozhodnutí a mít schopnost (a možnost) předvídat, jaké důsledky bude volba každé varianty mít. Tato rozhodnutí se mohou obecně vztahovat k libovolným oblastem a mít nejrůznější charakter. Za výběr z variant lze v tomto smyslu považovat např. i volbu za daných okolností nejlepší hodnoty nějaké veličiny. Je zřejmé, že se ideálně jedná o výběr nejlepší varianty ze všech možností za daných podmínek, touto problematikou se zabývá matematická disciplína zvaná optimalizace.

Oblast pro aplikaci optimalizace může být prakticky libovolná. V počátcích se jednalo zejména o hledání optimálního nastavení výrobních procesů nebo hledání ekonomicky nejziskovějších parametrů vztahujících se k výrobnímu plánu, investiční strategii apod. Přitom s ohledem na dřívější omezené výpočetní možnosti počítačů se jednalo o poměrně úzce vymezené jednoduché problémy a úlohy nevelkého rozsahu. Tak jako v jiných oblastech, i v optimalizaci představoval pokrok v oblasti výpočetní techniky rozšiřování aplikačních oblastí. Kromě možností zpracovávat úlohy většího rozsahu (posuzováno z hlediska množství zpracovávaných dat) se obor optimalizace vyvíjel a vyvíjí také ve směru hledání matematického popisu a zejména metod řešení nových a svým charakterem složitějších problémů. Tam, kde vhodná metoda řešení (dosud?) chybí, jsou využívány netradiční přístupy, např. heuristiky, jejichž použití se s rostoucí výpočetní kapacitou počítačů prosazuje stále více, neboť řešení bývají dosahována v uspokojivém čase a s uspokojivou spolehlivostí (heuristiky neposkytují záruku nalezení optimálního řešení, viz kapitola 3).

V souladu s všeobecným trendem k používání distribuovaných přístupů se tento přístup uplatňuje i v optimalizaci. K tomu existuje např. teoretický koncept označovaný jako *Distributed optimization programmes* (DOP) [38]. DOP je koncipován především jako zajímavý modelovací nástroj. Tato práce popisuje několik aplikačních úloh řešených s využitím přístupu DOP a na základě získaných zkušeností zavádí nový přístup k softwarové realizaci DOP úloh. Návrh OTP (*Optimization Transfer Protocol*) v kapitole 6 nepřináší nové optimalizační metody, ale nový originální způsob spolupráce jednotlivých optimalizačních uzlů úlohy řešené pomocí DOP přístupu. OTP umožňuje konzistentně zúročit ideje distribuované optimalizace. Předpokládá se realizace rozsáhlého softwaru pro optimalizaci založeného na OTP.

Je nutno podotknout, že pro DOP neexistuje zavedená česká terminologie, v této práci jsou proto uváděny i původní anglické termíny. Teoretická východiska jak optimalizace jako takové, tak heuristik a principů DOP jsou v práci popsána jen v míře nutné pro pochopení textu.

2 ZÁKLADY OPTIMALIZACE

Jako **optimalizace** se označuje matematická disciplína, která se zabývá snahou o nalezení takových hodnot nezávisle proměnných, pro které daná tzv. **účelová funkce** nabývá maximální či minimální hodnotu [21]. Optimalizace bývá někdy zařazována také do informatiky. Jedná se o disciplínu dosti obsáhlou, monografie věnované optimalizaci jako celku nebo dokonce i některé její části mívají značný rozsah. V této kapitole je podán pouze velmi stručný přehled zaměřený zejména na ty oblasti, které budou používány v další části textu. Tato kapitola vychází z různých pramenů. Bohužel je rozšířeným zvykem, že různí autoři používají místy odlišnou symboliku. V dalším textu byla maximální snaha věnována sjednocení této symboliky tak, aby byla konzistentní v rámci všech kapitol této práce.

Hledání maximální (příp. minimální) hodnoty účelové funkce $z_{max} = f(x_{max})$ je problémem hledání extrému funkce. Máme-li funkci f proměnné x a bod $x_0 \in D(f)$, kde $D(f)$ označuje definiční obor funkce f, pak funkce f má v bodě x_0 **lokální maximum**, jestliže existuje ryzí okolí $\bar{O}(x_0)$ takové, že platí $f(x_0) \geq f(x)$ pro všechna $x \in \bar{O}(x_0)$. Pokud je nerovnost ostrá, tedy $f(x_0) > f(x)$, pak má funkce f v bodě x_0 **ostré lokální maximum**. Platí-li opačné nerovnosti, pak se jedná o **lokální minimum** resp. **ostré lokální minimum**. Globální maximum (minimum) je potom největší (nejmenší) ze všech lokálních maxim (minim) přes celý definiční obor funkce.

Funkce má tedy v bodě x_0 ostré lokální maximum (minimum), jestliže v nějakém ryzím okolí bodu x_0 nabývá funkce pouze nižších (vyšších) hodnot než $f(x_0)$. Toto okolí by mělo celé ležet v definičním oboru funkce f (v případě omezení by potom uvedené mělo platit pro průnik okolí a definičního oboru funkce). Protože při optimalizaci obvykle existují omezení daná reálným problémem (např. nelze vyrábět záporné nebo nekonečně velké množství výrobků), bývá definována tzv. **množina přípustných řešení** jako podmnožina definičního oboru účelové funkce. Při hledání optimálního řešení potom hledáme **globální extrém funkce** na dané kompaktní množině ve smyslu *Weierstrassovy věty* „Spojitá funkce je na uzavřeném intervalu ohraničená a nabývá zde svého absolutního maxima a absolutního minima“. Uvažujeme-li tedy speciálně funkci $f(x)$, která je spojitá na intervalu $\langle a, b \rangle$, pak tato funkce může na tomto intervalu nabýt absolutního extrému buď v bodě, kde má tato funkce příslušný lokální extrém, nebo v některém krajním bodě intervalu $\langle a, b \rangle$.

Optimalizační úloha obecně může mít na neprázdné množině přípustných řešení právě jedno optimální řešení, nebo více řešení se stejnou hodnotou účelové funkce (dokonce nekonečně mnoho), nebo může být hodnota účelové funkce na množině přípustných hodnot neomezená (v takovém případě říkáme, že úloha nemá žádné optimální řešení).

Ačkoliv optimalizace, jak již bylo uvedeno výše, představuje hledání extrému funkce, často nelze vzhledem ke tvaru matematického modelu k řešení optimalizačních úloh použít metody klasické matematické analýzy založené na diferenciálním a integrálním počtu. Proto byly vyvinuty speciální techniky pro řešení různých typů úloh. Obecně lze optimalizační metody dělit na exaktní a přibližné (aproximativní). Exaktní metody

poskytují algoritmy pro přesné (zaručené) nalezení optimálního řešení specifického typu úlohy. Mnoho z nich spadá do oblasti tzv. **matematického programování**[1]. Některé z těchto metod budou zmíněny dále v této kapitole. Pro úlohy velkého rozsahu nebo pro typy úloh, kde odpovídající exaktní metoda neexistuje, se používají přibližné metody. Tyto nalezení globálně optimálního řešení negarantují. Do této kategorie spadají např. heuristické metody založené někdy na evolučních algoritmech (typicky např. genetické algoritmy, viz kapitola 3), nebo přístupy založené na algoritmech umělé inteligence (např. neuronové sítě) [21].

Vzhledem k tomu, že problém hledání minima lze jednoduše převést na hledání maxima prostou změnou znaménka účelové funkce, nebudou v dalším textu případy hledání maxima a minima posuzovány zvlášť.

Dle charakteru řešené úlohy a odpovídajícího tvaru použitého matematického modelu lze úlohy a odpovídající techniky řešení dělit do jistých kategorií, z hlediska dalšího textu je nejdůležitější dělení na úlohy *deterministické* a *stochastické*. Deterministické modely jsou takové modely, ve kterých se vyskytují pouze deterministické, tedy pevně dané, veličiny a vztahy. Naproti tomu v modelech stochastických se vyskytuje alespoň jedna veličina, která je náhodnou proměnnou, přičemž rozdělení pravděpodobnosti všech náhodných proměnných v modelu je obvykle známé. Podle typu problému (a s tím souvisejícího způsobu řešení) lze hovořit o různých odvětvích matematického programování, jedná se např. o:

- lineární programování,
- kvadratické programování,
- nelineární programování,
- konvexní programování,
- celočíselné programování,
- parametrické programování,
- vícekriteriální programování,
- stochastické programování,
- dynamické programování, atd.

S ohledem na zaměření práce bude dále pozornost věnována zejména rozsáhlým úlohám. Složitost řešených praktických úloh neustále vzrůstá; to je dáno rozšiřováním uplatnění optimalizačních přístupů do stále nových a nových oblastí. Toto je umožněno pokrokem ve výpočetní technice, který dovoluje jednak rychleji řešit stále rozsáhlejší úlohy, jednak také používat i nové přístupy, které byly v minulosti buď nerealizovatelné a/nebo nebylo jejich použití pro reálné problémy efektivní (typicky např. heuristické a simulační přístupy). U složitosti optimalizačních úloh lze rozlišovat její dvě stránky – kvantitativní a kvalitativní. Kvantitativní složitost spočívá v rozsahu úlohy daném velikostí

[1] Slovo programování se v tomto významu nevztahuje k programování jakožto tvorbě počítačového softwaru.

dat (počet proměnných, parametrů, omezení). Kvalitativní stránka složitosti se potom vztahuje ke struktuře úlohy.

Úlohy velkého rozsahu často vznikají spojováním několika původně nezávislých menších úloh v prostoru nebo v čase. Tyto úlohy mohou být řešeny jako celek použitím běžných metod a algoritmů, aniž by bylo využito jejich struktury k usnadnění řešení. Tento přístup „řešení hrubou silou“ je založen na extenzivním přístupu, tedy využití rostoucí výpočetní rychlosti a paměťové kapacity počítačů. Dalšího pokroku lze dosáhnout například paralelizací výpočtu tam, kde je to možné (typicky např. maticové operace) a kde jsou k dispozici odpovídající technické prostředky. Ačkoliv se technické limity neustále posouvají, je výhodné při řešení využívat i speciální strukturu rozsáhlých úloh.

2.1 Deterministická optimalizace

Jak již bylo uvedeno, matematické programování je vědní odvětví, které se zabývá problematikou nalezení optimálního řešení účelové funkce při existenci omezujících podmínek vyjádřených ve formě rovností či nerovností, které určují množinu přípustných řešení v prostoru o konečném počtu dimenzí. Obecná formulace **matematického programu** pro případ hledání minima má tvar:

$$? \in \arg\min_{\mathbf{x}} \left\{ f(\mathbf{x}) \,\middle|\, \mathbf{x} \in C \right\}, \tag{2.1}$$

kde C je množina přípustných řešení, $C \subseteq \mathbb{R}^n$, $n \in \mathbb{N}$, funkce $f(\mathbf{x})$, přesněji $f: C \to \mathbb{R}$, je účelová funkce a $\mathbf{x} \in C$ je rozhodovací proměnná (vektor) [38]. Úkolem je nalézt alespoň jedno optimální řešení $\mathbf{x}_{min}$ z množiny přípustných řešení *C*, které minimalizuje účelovou funkci $f(\mathbf{x})$. Symbol argmin značí množinu optimálních řešení (minimalizátorů). V (2.1) je znak ? (otazník) společně s operátorem ∈ použit pro odlišení symbolického zápisu (2.1) např. od situace, kdy jsou hledána všechna optimální řešení. Vztah (2.1) lze považovat za výchozí specifikaci matematického programu (*mathematical program* – MP).

Pomocí matematických programů bývají modelovány také rozhodovací problémy [21]. Matematické programy často obsahují důležité na problému závislé konstanty, tyto tvoří deterministické parametry matematického programu. Tyto parametry lze zakomponovat přímo do definice matematického programu, potom dostaneme:

$$? \in \arg\min_{\mathbf{x}} \left\{ f(\mathbf{x}, \mathbf{a}) \,\middle|\, \mathbf{x} \in C(\mathbf{a}) \right\}, \tag{2.2}$$

kde $\mathbf{a}$ je konstantní parametr, $\mathbf{a} \in \mathbb{R}^K$, $K \in \mathbb{N}$. Model (2.2) se označuje jako parametrický matematický program (*parametric mathematical program* – PMP).

Pro řešení optimalizačních problémů existuje velmi mnoho metod, z nichž některé jsou velmi sofistikované, specializované, složité a jejich vysvětlení a popis jsou velmi obsáhlé. Množství existujících metod je zdůvodnitelné rozmanitostí optimalizačních úloh a také

forem, ve kterých jsou popisovány. Ať už je ale složitost algoritmů i úloh jakákoliv, lze nalézt obecný postup společný celé řadě algoritmů. Schéma tohoto postupu lze popsat následovně [11]:

1. Získání počátečního odhadu řešení $\mathbf{x}_0 \in C$.
2. Pro $k = 0,1,2,\ldots$ opakovat následující kroky:
   a. Pokud je $\mathbf{x}_k$ optimální, ukončení algoritmu.
   b. Stanovení dalšího odhadu řešení $\mathbf{x}_{k+1} \in C$.

Tento postup se na první pohled jeví být příliš obecným a nedostatečným, neboť nestanoví, jakým způsobem se mají provádět jednotlivé kroky. Nejen, že neobsahuje způsob stanovení odhadu řešení (ať už počátečního či dalšího), ale ani není uvedeno, jak lze detekovat, že nalezené řešení je optimální. To je mimochodem dosti vážný problém, neboť s výjimkou testovacích úloh není optimální řešení předem známo, takže formulace vhodné podmínky ukončení je zejména u heuristických algoritmů obtížná. Některé další poznámky na toto téma lze nalézt v kapitole 3. Výše popsaný algoritmus ale přece jen obsahuje některé důležité informace. Je zřejmé, že řešení optimalizační úlohy bude mít iterační charakter, tzn. nelze použít řešení typu „dosazení do vzorce“. Např. i známá a „proslulá“ simplexová metoda (viz odstavec 2.1.1) přesně odpovídá tomuto schématu, ovšem samozřejmě detailně precizuje jednotlivé kroky. Dále z algoritmu plyne, že test optimality řešení a způsob přechodu na další odhad řešení v dalším kroku jsou dvě oddělené činnosti, i když mezi nimi může být jistá souvislost – výsledek testu například může sloužit jako vodítko pro určení dalšího řešení. Některé známé metody řešení pro řešení jednotlivých typů úloh budou zmíněny v dalším textu.

2.1.1 Lineární programy

Lineární program (*linear mathematical program* – LMP) je matematický program, kde účelová funkce i výrazy v omezeních jsou lineární kombinací prvků vektoru **x**. Lineární program tedy lze specifikovat jako:

$$? \in \arg\min_{\mathbf{x}} \left\{ \mathbf{c}^{\top}\mathbf{x} \,\middle|\, \mathbf{A}\mathbf{x} \lozenge \mathbf{b},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u} \right\}, \tag{2.3}$$

kde:

$\mathbf{c} = (c_j) \in \mathbb{R}^n,\ j \in J$, jsou koeficienty účelové funkce, J je množina indexů proměnných, obvykle se volí $J = \{1, \ldots, n\}$.

$\mathbf{b} = (b_i) \in \mathbb{R}^m$, $\boldsymbol{A} = (a_{ij}) \in \mathbb{R}^{m \times n}$, $i \in I, j \in J$, jsou koeficienty omezujících podmínek, I je množina indexů pro omezující podmínky, obvykle $I = (1, \ldots, m)$.

$\mathbf{l} = (l_j) \in \mathbb{R}^n$, $\mathbf{u} = (u_j) \in \mathbb{R}^n$, $j \in J$ jsou vektory horních a dolních mezí pro **x** (obvykle je pro jednotlivé prvky vektoru **x** dolní mez rovna 0 a horní mez ∞, meze také bývají často zahrnuty již přímo v omezeních).

$\lozenge$ je symbol reprezentující vektor relačních operátorů, $\lozenge \in \{\le, \ge, =\}^m$.

Vzhledem k nejčastějšímu významu v optimalizačních úlohách jsou koeficienty omezujících podmínek a_{ij} obvykle nazývány strukturálními koeficienty, prvky vektoru **b** jsou nazývány kapacitními limity a prvky vektoru **c** cenovými koeficienty resp. cenami.

S ohledem na metody řešení úloh lineárního programování je užitečné rozlišovat mezi třemi tvary obecné formulace úlohy lineárního programování, které vychází z obecného formulace (2.3). Pokud a_{ij}, b_i, c_j $(i = 1,2,\dots,m; j = 1,2,\dots,n)$ jsou daná reálná čísla a $I_1 \subset I = \{1,2,\dots,m\}$, $J_1 \subset J = \{1,2,\dots,m\}$, lze účelovou funkci zapsat ve tvaru

$$z = \sum_{j=1}^{n} c_j x_j \ . \tag{2.4}$$

a množinu přípustných řešení jako soustavu lineárních rovnic a nerovností

$$\begin{aligned} &\sum_{j=1}^{n} a_{ij} x_j \le b_i && (i \in I_1) \\ &\sum_{j=1}^{n} a_{ij} x_j = b_i && (i \in I - I_1) \\ &x_j \ge 0 && (j \in J_1). \end{aligned} \tag{2.5}$$

Pokud platí, že $I_1 \neq \emptyset, I_1 \neq I$ nebo $J_1 \neq J$, vztahy (2.4) a (2.5) popisují minimalizační *úlohu lineárního programování ve smíšeném tvaru*. Pokud $I_1 = \emptyset$ a $J_1 = J$, pak se jedná o *úlohu lineárního programování v rovnicovém (standardním) tvaru*; v tomto případě bude množina přípustných řešení popsána jednodušeji, a sice:

$$\begin{aligned} &\sum_{j=1}^{n} a_{ij} x_j = b_i && (i = 1,2,\dots,m) \\ &x_j \ge 0 && (j = 1,2,\dots,n). \end{aligned} \tag{2.6}$$

Konečně, pokud bude množina přípustných řešení popsána pouze soustavou nerovnic, tedy $I_1 = I$, $J_1 = J$, pak se jedná o *úlohu lineárního programování ve tvaru nerovnic*. Každou úlohu lineárního programování ve smíšeném tvaru nebo ve tvaru nerovností lze převést na rovnicový tvar zavedením tzv. přídatných proměnných. Tyto mají pochopitelně v účelové funkci nulové koeficienty, takže hodnotu účelové funkce ani polohu jejího extrému neovlivní. Ačkoliv jsou tyto proměnné zavedené formálně za účelem usnadnění výpočtu, mohou mít v konkrétních úlohách i reálný význam (např. rezervy ve zdrojích surovin apod.).

Množinu přípustných řešení úlohy lineárního programování ve standardním tvaru lze specifikovat jako

$$C_{LP} = \left\{ \mathbf{x} \in \mathbb{R}^n \,\middle|\, \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \ge \mathbf{0} \right\}. \tag{2.7}$$

Optimální řešení minimalizační úlohy lineárního programování $\mathbf{x}_{min} \in C_{LP}$ je takové přípustné řešení, pro které platí

$$\mathbf{c}^{\mathsf{T}}\mathbf{x}_{min} \leq \mathbf{c}^{\mathsf{T}}\mathbf{x} \;\; \forall \mathbf{x} \in C_{LP}\,. \tag{2.8}$$

Množina přípustných řešení úlohy lineárního programování (ve tvaru nerovností, smíšeném nebo rovnicovém) je konvexní polyedrická množina. Množina všech optimálních řešení úlohy lineárního programování je rovněž konvexní polyedrická množina [36]. Pro úvahy o řešení, a také pro vlastní postup řešení, je důležitý pojem *krajní bod množiny*. Nechť $S \subset \mathbb{R}^n$ je libovolná množina. Bod $\mathbf{s} \in S$ je krajním bodem množiny S, jestliže neexistují body $\mathbf{x}, \mathbf{y} \in S$ a číslo $\alpha \in (0,1)$ tak, že $\mathbf{x} \neq \mathbf{y}$ a $\mathbf{s} = \alpha\mathbf{x} + (1-\alpha)\mathbf{y}$.

Dalším důležitým pojmem je *základní řešení* úlohy lineárního programování. Přípustné řešení $\mathbf{x} \in C_{LP} = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ je základním řešením úlohy lineárního programování, jestliže jsou sloupce matice $\mathbf{A}$ s indexy odpovídajícími nenulovým složkám vektoru $\mathbf{x}$ lineárně nezávislé. Z výše uvedeného lze dovodit důležitý závěr, a sice že bod $\mathbf{x} \in C_{LP} = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ je krajním bodem množiny C_{LP} právě tehdy, je-li základním řešením této úlohy.

Pro řešení úlohy lineárního programování má klíčový význam *základní věta lineárního programování*. Tato věta praví, že pro úlohu lineárního programování

$$? = \arg\min_{\mathbf{x}} \left\{ \mathbf{c}^{\mathsf{T}}\mathbf{x} \,\middle|\, \mathbf{x} \in C_{LP} = \left\{ \mathbf{x} \in \mathbb{R}^n \,\middle|\, \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \right\} \right\} \tag{2.9}$$

a její množinu optimálních řešení

$$C_{LP}^{*} = \left\{ \mathbf{x}^{*} \in C_{LP} \,\middle|\, \mathbf{c}^{\mathsf{T}}\mathbf{x}^{*} = \min_{\mathbf{x} \in C_{LP}} \mathbf{c}^{\mathsf{T}}\mathbf{x} \right\} \tag{2.10}$$

platí jedna ze tří možností:

a) $C_{LP} = \emptyset$,

b) $C_{LP} \neq \emptyset \;\wedge \inf_{\mathbf{x} \in C_{LP}} \mathbf{c}^{\mathsf{T}}\mathbf{x} = +\infty$, $C_{LP}^{*} = \emptyset$,

c) $C_{LP}^{*} \neq \emptyset$.

Dále platí, že pokud je množina přípustných řešení neprázdná, tedy $C_{LP} \neq \emptyset$, pak existuje základní přípustné řešení. Pokud je neprázdná množina optimálních řešení, tedy $C_{LP}^{*} \neq \emptyset$, pak existuje základní optimální řešení. Pro optimální řešení tedy ze základní věty lineárního programování plyne, že pokud existuje, patří mezi základní řešení.

Uvedeného závěru využívá známá *simplexová metoda* vyvinutá Dantzigem[2] a poprvé publikovaná v roce 1947. Tato metoda pro řešení úlohy lineárního programování nabízí způsob procházení krajních bodů množiny přípustných řešení (tedy procházení základních řešení úlohy lineárního programování). Tím se z obecně nekonečné množiny přípustných řešení hledání omezí na konečný (a obvykle vcelku nízký) počet krajních bodů. V úvodu je nutné zjistit nějaké základní řešení. V případě minimalizace se dále vždy přechází na ten sousední krajní bod, ve kterém je hodnota účelové funkce menší. Není-li takový bod, pak

[2] George Dantzig, americký matematik, 1914–2005.

buď bylo nalezeno optimální řešení, nebo je některá hrana vycházející z tohoto krajního bodu neomezená, a pak optimální řešení neexistuje vzhledem k neomezenosti hodnoty účelové funkce na množině přípustných řešení. Algoritmus je formulován speciálně se zřetelem pro výpočet prováděný pomocí počítače, ruční výpočet pro úlohu obsahující více proměnných je sice možný, ale časově náročný. I když je princip algoritmu poměrně jednoduchý, jeho detailní popis je poměrně rozsáhlý a pro účely tohoto textu není nezbytný. Podrobnosti lze nalézt např. v [11].

Lineární program lze modifikovat dalšími podmínkami kladenými na prvky vektoru **x** (tzn. omezením množiny přípustných řešení C), tyto modifikace jsou užitečné pro řešení některých speciálních typů úloh. Vztah (2.3) definující speciální lineární program lze zapsat ve tvaru:

$$? \in \arg\min_{\mathbf{x}} \left\{ \mathbf{c}^{\top}\mathbf{x} \,\middle|\, \mathbf{A}\mathbf{x} \lozenge \mathbf{b},\ \mathbf{x} \in D \right\}, \tag{2.11}$$

kde $D \subset \mathbb{R}^n$ je diskrétní množina. Pokud platí, že $D \subset \mathbb{Z}^n$ ($\mathbb{Z}$ jako obvykle označuje množinu celých čísel), jedná se o **úplně celočíselný lineární program** (*integer linear program* – ILP). Pokud $D \subset \mathbb{Z}^l \times \mathbb{R}^{n-l}$, jedná se o **smíšený celočíselný lineární program** (*mixed integer linear program* – MILP). Pokud $D = \{0,1\}^n$, jedná se o tzv. **binární programování** (někdy též **nula-jedničkové programování**).

Požadavek na celočíselnost vyvstává v řadě praktických úloh. Někdy je tomu tak proto, že úloha pracuje s počty nedělitelných objektů (osoby, bity, …). Těmto úlohám se říká úlohy s nedělitelnostmi. V těchto případech se často používají běžné metody lineárního programování a výsledek se nakonec upravuje na celá čísla, a výsledky jsou vcelku uspokojivé. Celočíselné programování však umožňuje řešit nejen problémy s nedělitelnostmi, ale také řadu komplikovaných problémů, které nelze efektivně řešit. Jedná se zejména o kombinatorické problémy, v nichž jde o nalezení optimálního řešení z konečné (i když obvykle velmi rozsáhlé) množiny přípustných řešení[3]. Jedná se např. o problém obchodního cestujícího nebo rozvrhovací problémy.

2.1.2 Síťové úlohy

Lineární programy bývají často definovány na sítích. Takové programy mají mnoho speciálních vlastností, které např. umožňují definovat modifikovanou síťovou simplexovou metodu [11].

Obecný tvar síťové optimalizační úlohy pro minimalizaci nákladů toku v síti lze zapsat jako úlohu LP např. v podobě:

[3] I když u klasické „spojité“ úlohy je množina přípustných řešení teoreticky nekonečná, je nutné si uvědomit, že pokud k řešení použijeme počítač (což je kromě triviálních případů v podstatě vždy), pak vždy vybíráme řešení z konečné množiny. Počítače pro uložení jedné číselné hodnoty používají konečně velké paměťové místo, takže nemohou pracovat s rozlišením na nekonečně mnoho platných cifer ani s neomezeným rozsahem. To ovšem v žádném případě neznamená, že by všechny úlohy měly být považovány za celočíselné, jde pouze o problém omezené numerické přesnosti výpočtu.

$$? \in \arg\min_{\mathbf{x}} \left\{ \mathbf{c}^{\top}\mathbf{x} \,\middle|\, \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{l} \le \mathbf{x} \le \mathbf{u} \right\}, \tag{2.12}$$

kde **l** a **u** jsou vektory dolních a horních mezí proměnné **x**. Ačkoliv (2.12) se na první pohled neliší od obecné úlohy LP, je popis síťové úlohy speciální vzhledem k charakteru a datovému popisu síťové úlohy. Matematický program je definován na síti, síť je tvořena orientovaným grafem, který má m uzlů a n hran. Prvky vektoru **x** představují toky hranami. Protože hrana spojuje dva uzly, bývá zvykem pro snazší orientaci indexovat prvky **x** pomocí dvojice indexů (ovšem **x** pochopitelně stále zůstává vektorem). Prvek $x_{i,k}$ popisuje tok hranou vedoucí z uzlu *i* do uzlu *k*. Pro tento text indexujeme uzly a hrany samostatně a vazba mezi nimi je určena nepřímo pomocí dále uvedené incidenční matice **A**. Prvky vektoru **c** bývají indexovány stejně jako prvky vektoru **x**, tedy prvek $c_{i,k}$ resp. c_j představuje cenu za jednotkový tok hranou *j* směřující z uzlu *i* do uzlu *k*. Matice **A** o rozměrech $m \times n$ je v podstatě incidenční maticí orientovaného grafu, řádky představují uzly a sloupce hrany, je-li prvek $a_{ij} = -1$, pak to znamená, že hrana *j* vychází z uzlu *i* (hranou z uzlu něco odtéká), pokud je $a_{ij} = +1$, pak hrana *j* vstupuje do uzlu *i* (hranou něco vtéká do uzlu *i*, hrana končí v uzlu *i*), nulová hodnota znamená, že hrana *j* nemá žádnou incidenci s uzlem *i*. Prvky vektoru **b** představují tok mezi uzlem a vnějším prostředím, nulová hodnota znamená, že uzel je čistě tranzitní, kladná hodnota znamená odtok z uzlu ze sítě do vnějšího prostředí a záporná hodnota přítok do uzlu z vnějšího prostředí. Soustava rovnic $\mathbf{A}\mathbf{x} = \mathbf{b}$ potom vyjadřuje podmínky, že pro každý uzel musí být součet všech přítoků a odtoků roven nule (analogie Kirchhoffových zákonů).

Síťové úlohy jsou velmi důležité. Síť může představovat skutečnou potrubní síť, síť elektrického vedení, tok výrobní linkou, datovou síť; může ale také být čistou abstrakcí. Síťové úlohy tak mají velmi široké uplatnění např. při řešení dopravních problémů, v logistice, směrování dat atd. Kromě již zmíněné modifikované síťové úlohy se při řešení uplatňují algoritmy z oblasti teorie grafů a pro rozsáhlé úlohy též heuristické metody.

2.1.3 Nelineární programy

Nelineární program (*nonlinear program* – NLP) může být specifikován obdobným způsobem jako program lineární:

$$? \in \arg\min_{\mathbf{x}} \left\{ f(\mathbf{x}) \,\middle|\, \mathbf{g}(\mathbf{x}) \lozenge \mathbf{0},\ \mathbf{x} \in X \right\}, \tag{2.13}$$

kde $\mathbf{g}: \mathbb{R}^n \to \mathbb{R}^m$ a $X \subset \mathbb{R}^n$, přičemž kterákoliv funkce (účelová funkce, funkce z omezení) může být nelineární. Nelineární program lze pochopitelně zapsat i jako parametrický matematický program se symbolickým zápisem totožným s (2.2):

$$? \in \arg\min_{\mathbf{x}} \left\{ f(\mathbf{x}, \mathbf{a}) \,\middle|\, \mathbf{x} \in C(\mathbf{a}) \right\}, \tag{2.14}$$

kde $C(\mathbf{a}) = \{\mathbf{x} \mid \mathbf{g}(\mathbf{x}, \mathbf{a}) \lozenge \mathbf{0}, \mathbf{x} \in X\}$, funkce **g** je obecně nelineární.

Podobně jako u lineárního programu, i u programu nelineárního lze zaměněním množiny *X* za množinu *D* získat celočíselný nelineární program (*integer nonlinear program* – INLP).

2.1.4 Dekomponovatelné a zahnízděné matematické programy

Matematický program může být zapsán jako dvoukrokový dekomponovaný (složený) program (*two-step decomposed program*, 2DP) ve tvaru:

$$? = \min_{\mathbf{x}_1}\left\{ \min_{\mathbf{x}_2}\left\{ f(\mathbf{x}_1,\mathbf{x}_2) \middle|\, \mathbf{x}_2 \in C_2(\mathbf{x}_1)\right\} \middle|\, \mathbf{x}_1 \in C_1 \right\}, \tag{2.15}$$

kde

$C_1 \subseteq \mathbb{R}^{n_1}, \forall \mathbf{x_1} \in C_1\colon C_2(\mathbf{x_1}) \subseteq \mathbb{R}^{n_2}$,
$\mathbf{x} = (\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}})^{\mathrm{T}}$,
$C = \{\mathbf{x} \middle| \mathbf{x} = (\mathbf{x}_1^{\mathrm{T}}, \mathbf{x}_2^{\mathrm{T}})^{\mathrm{T}}, \mathbf{x}_1 \in C_1, \mathbf{x}_2 \in C_2(\mathbf{x}_1)\}$,
$n = n_1 + n_2$.

Zvláštním zajímavým případem je situace, kdy účelovou funkci lze rozdělit na dvě funkce:

$$? = \min_{\mathbf{x}_1}\left\{ f\left(\mathbf{x}_1, \min_{\mathbf{x}_2}\left\{ f_2(\mathbf{x}_2) \middle|\, \mathbf{x}_2 \in C_2(\mathbf{x}_1)\right\}\right) \middle|\, \mathbf{x}_1 \in C_1 \right\}, \tag{2.16}$$

kde $f_1\colon C_1 \times \mathbb{R}^{n_1} \to \mathbb{R}$, $f_2\colon \mathbb{R}^{n_2} \to \mathbb{R}$, $f(\mathbf{x}) = f_1\big(\mathbf{x}_1, f_2(\mathbf{x}_2)\big)$. Nezbytnou podmínkou dekomponovatelnosti je, aby funkce f_1 byla ryze rostoucí podle druhého argumentu. Vnitřní program v (2.16) je parametrický matematický program (2.2).

2.2 Stochastické programování

V oblasti modelování a řešení stochastických úloh lze vyjít z parametrického matematického programu (2.2). Nahradíme-li některé konstantní parametry náhodnými proměnnými, získáme stochastický program:

$$? \in \arg\min_{\mathbf{x}}\left\{ f(\mathbf{x},\boldsymbol{\xi}) \middle|\, \mathbf{x} \in C(\boldsymbol{\xi})\right\}, \tag{2.17}$$

kde $\boldsymbol{\xi}\colon \Omega \to \mathbb{R}^K$ je náhodný vektor z pravděpodobnostního prostoru $(\Omega, \mathcal{F}, P)$, a jeho rozdělení pravděpodobnosti nezávisí na rozhodnutí $\mathbf{x}$. Funkce $f\colon \mathbb{R}^n \times \Omega \to \mathbb{R}$ je měřitelná pro každé $\mathbf{x} \in \mathbb{R}^n$. Častý je zápis úlohy stochastického programování ve tvaru

$$? \in \arg\min_{\mathbf{x}}\left\{ f(\mathbf{x},\boldsymbol{\xi}) \middle|\, \mathbf{g}(\mathbf{x},\boldsymbol{\xi}) \lozenge \mathbf{0}, \mathbf{x} \in X\right\}, \tag{2.18}$$

kde funkce $\mathbf{g}\colon \mathbb{R}^n \times \Omega \to \mathbb{R}^m$ je měřitelná pro každé $\mathbf{x} \in \mathbb{R}^n$. Řešením této úlohy se rozumí řešení deterministického ekvivalentu, který je definován tak, aby byla korektně odstraněna

náhodnost z původní úlohy. Při konstrukci deterministického ekvivalentu je klíčové rozhodnutí, co bude považováno za přípustné řešení (tedy deterministická reformulace omezení) a co bude považováno za optimální řešení.

Množinu přípustných řešení deterministického ekvivalentu C_{DE} lze pro úlohu (2.18) zkonstruovat například s využitím středních hodnot $E(\boldsymbol{\xi})$, a to buď ve tvaru

$$C_{DE} = \left\{\mathbf{x} \in X \,\middle|\, \mathbf{g}(\mathbf{x}, E(\boldsymbol{\xi})) \lozenge \mathbf{0}\right\}, \tag{2.19}$$

nebo ve tvaru

$$C_{DE} = \left\{\mathbf{x} \in X \,\middle|\, E_{\boldsymbol{\xi}}\, \mathbf{g}(\mathbf{x}, \boldsymbol{\xi}) \lozenge \mathbf{0}\right\}, \tag{2.20}$$

index *DE* v obou případech značí deterministický ekvivalent. Jinou možností je modelovat neurčitost veličinou s diskrétním rozdělením pravděpodobnosti a s konečným počtem realizací zvaných *scénáře*, index $s \in S$ označuje scénář:

$$C_{DE} = \bigcap_{\boldsymbol{\xi}^s \in S} \left\{\mathbf{x} \in X \,\middle|\, \mathbf{g}(\mathbf{x}, \boldsymbol{\xi}^s) \lozenge \mathbf{0}\right\}. \tag{2.21}$$

Při reformulaci účelové funkce lze použít střední hodnotu náhodného vektoru $\boldsymbol{\xi}$

$$f_{DE}(\mathbf{x}) = f(\mathbf{x}, E(\boldsymbol{\xi})), \tag{2.22}$$

střední hodnotu originální účelové funkce

$$f_{DE}(\mathbf{x}) = E_{\boldsymbol{\xi}}(f(\mathbf{x}, \boldsymbol{\xi})), \tag{2.23}$$

případně kombinaci střední hodnoty a rozptylu funkce $f(\mathbf{x}, \boldsymbol{\xi})$

$$f_{DE}(\mathbf{x}) = (1-\lambda) E_{\boldsymbol{\xi}}(\mathbf{x}, \boldsymbol{\xi}) + \lambda D_{\boldsymbol{\xi}}(f(\mathbf{x}, \boldsymbol{\xi})), \tag{2.24}$$

kde $\lambda \in \langle 0,1 \rangle$. Další možnosti deterministického ekvivalentu lze nalézt v [38].

V analýze úlohy stochastického programování je z principu vždy přítomna neurčitost. Otázkou ale je, kdy bude muset být provedeno rozhodnutí, tedy stanovení optimálního řešení. Buď se tak musí stát dříve, než je možné určit momentální hodnotu náhodného parametru, nebo lze rozhodnutí odložit a provést ho až po zjištění aktuální hodnoty náhodného parametru, tedy po pozorování $\boldsymbol{\xi}^S$. První uvedený případ charakterizuje tzv. *here-and-now* přístup, v případě druhém se jedná o přístup *wait-and-see* [41]. U here-and-now přístupu (dále HN) musí být hodnota rozhodnutí $\mathbf{x}$ vždy stejná bez ohledu na budoucí realizace $\boldsymbol{\xi}$. Tento přístup je ve stochastickém programování obvyklejší, neboť je dán převládajícím charakterem úloh.

Přístup wait-and-see (dále WS) lze dle [22] charakterizovat jako „analýza v podmínkách nejistoty, rozhodnutí v podmínkách jistoty“. V tomto případě lze rozhodnutí modifikovat podle skutečné hodnoty $\boldsymbol{\xi}$, takže rozhodnutí $\mathbf{x}$ je funkcí $\mathbf{x}(\boldsymbol{\xi})$. To v důsledku znamená, že hodnota účelové funkce $f(\mathbf{x}(\boldsymbol{\xi}), \boldsymbol{\xi})$ je náhodná hodnota. Tento přístup je důležitý zejména pro úlohy dlouhodobého plánování, kde lze rozhodnutí operativně

přizpůsobovat nastalé situaci. S využitím uvedeného lze WS deterministickou reformulaci úlohy (2.17) (tzv. WS program) zapsat ve tvaru:

$$? \in \arg\min_{\mathbf{x}(\boldsymbol{\xi})} \left\{ f(\mathbf{x}(\boldsymbol{\xi}), \boldsymbol{\xi}) \middle| \, \mathbf{x}(\boldsymbol{\xi}) \in C(\boldsymbol{\xi}) \right\}, \tag{2.25}$$

kde $\mathbf{x}(\boldsymbol{\xi})$ je měřitelnou funkcí $\mathbf{x}$: $\mathbb{R}^k \rightarrow \mathbb{R}^n$. V tomto případě lze skutečně (2.25) považovat za speciální případ parametrického programu, dokonce existuje indukované rozdělení pravděpodobnosti $\mathbf{x}(\boldsymbol{\xi})$. Pro zdůraznění faktu, že není počítáno pouze jedno řešení $\mathbf{x}_{min}$, lze WS program zapsat také ve tvaru:

$$\forall x \in \Xi : ? \in \arg\min_{\mathbf{x}(\boldsymbol{\xi})} \left\{ f(\mathbf{x}(\boldsymbol{\xi}), \boldsymbol{\xi}) \middle| \, \mathbf{x}(\boldsymbol{\xi}) \in C(\boldsymbol{\xi}) \right\}, \tag{2.26}$$

Množina všech optimálních řešení pro WS program potom bude

$$X_{min}^{WS}(\boldsymbol{\xi}) = \operatorname{argmin}_{\mathbf{x}(\boldsymbol{\xi})} \left\{ f(\mathbf{x}(\boldsymbol{\xi}), \boldsymbol{\xi}) \middle| \, \mathbf{x}(\boldsymbol{\xi}) \in C(\boldsymbol{\xi}) \right\}. \tag{2.27}$$

Další podrobnosti k WS nejsou pro účely tohoto textu nezbytné, lze je nalézt např. v [38].

Jak již bylo uvedeno, je v mnoha praktických situacích nutné přijmout rozhodnutí dříve, než je známé pozorování, tedy používat přístup here-and-now. Stochastické programování se také primárně zabývá HN přístupem. U HN přístupu nelze přijmout rozhodnutí $\mathbf{x}$ na základě pozorování $\boldsymbol{\xi}^S$, ale je nutné přijmout rozhodnutí, které bude stejné bez ohledu na to, jakou hodnotu následně nabude náhodný parametr $\boldsymbol{\xi}$. Aby bylo možné vůbec výpočet provést, je nutné využít znalost, která je k dispozici, např. znalosti budoucích realizací $\boldsymbol{\xi}^S$. Situace popsaná hodnotou $\boldsymbol{\xi}^S$ poté může, ale také nemusí nastat, jedná se tedy pouze o jednu z mnoha možností, tzv. individuální scénář. Pro něj lze aplikovat WS přístup na vybranou množinu scénářů, které se s ohledem na řešený problém jeví být jako důležité. Mezi WS řešeními pak lze vybrat podle zvoleného kritéria to nejvýhodnější pro všechny scénáře jako HN řešení. Tento přístup je často používán pro řešení složitých problémů zejména v ekonomii, uplatňuje se ale i v logistice a při řešení složitých technických problémů (např. [39]). Pro původní úlohu stochastického programování (2.17) lze její HN deterministický ekvivalent pro daný individuální scénář zapsat ve tvaru

$$? \in \operatorname{argmin}_{\mathbf{x}} \left\{ f(\mathbf{x}, \boldsymbol{\xi}^S) \middle| \, \mathbf{x} \in C(\boldsymbol{\xi}^S) \right\}, \tag{2.28}$$

kde $\boldsymbol{\xi}^S \in \Xi$ je specifikovaný individuální scénář. Aplikace HN přístupu je použita a více rozpracována v kapitole 5.2.

3 HEURISTICKÉ METODY, GENETICKÉ ALGORITMY

Jak již bylo uvedeno v předchozí kapitole, znamená řešení optimalizační úlohy hledání globálního extrému funkce na dané oblasti. V ideálním případě lze toto řešit vyšetřením průběhu funkce všeobecně známými postupy matematické analýzy. Bohužel, většina praktických úloh tímto postupem řešitelná není s ohledem na formulaci účelové funkce [5, 10]. Zde se potom otevírá pole působnosti pro operační analýzu a optimalizaci jako obor aplikované matematiky.

Ačkoliv je optimalizace značně propracovaná disciplína, detailní obecně použitelné algoritmy (obvykle iterativního charakteru), kde stačí jen formálně správně a vhodně popsat konkrétní úlohu a pak už jen „dosadit vstupní data a spustit výpočet“, existují jen pro omezené okruhy a typy úloh. Typickým představitelem tohoto typu algoritmů je simplexová metoda pro řešení úloh lineárního programování [4, 36]. Pro širokou škálu úloh potom sice je znám princip řešení, ale neexistuje obecně použitelný algoritmus. Tento je nutno v podstatě sestavovat speciálně pro každou úlohu s ohledem na její strukturu a specifika (toto platí např. pro dynamické programování, viz [22]). Konečně existují i úlohy, u kterých sice lze o každé hodnotě rozhodovacího vektoru rozhodnout, zda se jedná o přípustné řešení, a vyčíslit odpovídající hodnotu účelové funkce, ale použití výše uvedených postupů není možné. Metodou, která by zaručeně vedla k řešení, by v tomto případě mohlo být tzv. řešení „hrubou silou“, tzn. postupně generovat všechna přípustná řešení, pro každé vypočítat hodnotu účelové funkce a porovnáním stanovit optimální řešení (metoda *úplné enumerace*). Je zřejmé, že s výjimkou úloh velmi malého rozsahu se jedná o metodu značně neefektivní. Častou námitkou zde bývá nekonečnost množiny přípustných řešení (nemusí platit pro celočíselné a neplatí pro kombinatorické úlohy). Tento problém je ale pouze teoretický, při řešení na počítači je množina všech různých přípustných řešení, ze které je řešení skutečně vybíráno, vždy konečná s ohledem na limitovanou velikost paměťového místa pro uložení hodnoty prvku rozhodovacího vektoru (např. jsou-li prvky rozhodovacího vektoru chápány jako číselné hodnoty, je logicky omezen jejich počet platných cifer). Protože požadujeme nejen nalezení optimálního řešení, ale rovněž jeho nalezení co nejefektivnějším způsobem, tj. s použitím dostupných prostředků v co nejkratším čase (nebo alespoň s ohledem na charakter řešeného problému v „rozumném“ čase), jeví se metoda úplné enumerace pro praktické použití jako nevhodná, viz např. [37].

Cílem tedy je nalezení vhodné strategie procházení množiny přípustných řešení tak, aby bylo nutné vyhodnocovat hodnotu účelové funkce pro co nejméně přípustných řešení, a přitom aby bylo nalezeno co nejlepší (v ideálním případě optimální) řešení. Ať je zvolena strategie jakákoliv, není obvykle nalezení optimálního řešení obecně garantováno a zejména je obtížná (až nemožná) detekce situace, kdy optimální řešení již bylo dosaženo (blíže viz odstavec 3.3.6). Nalezené nejlepší řešení je proto obvykle označováno jako suboptimální (někdy též kvazioptimální).

V případě mimořádně komplikovaných a mimořádně výpočetně náročných úloh se někdy v nouzi volí poněkud nestandardní přístup, kdy je náhodně vygenerován určitý počet přípustných řešení a z nich je vybráno řešení s nejlepší hodnotou účelové funkce (tzv. náhodné prohledávání, *random search*). I když v praxi může být tento přístup někdy akceptovatelný, jedná se o postup značně nesofistikovaný a pravděpodobnost nalezení řešení blízkého optimálnímu je dosti nízká. Tento postup je ale východiskem tzv. *heuristických metod*. Obecně se pojem heuristika[4] používá pro zkusmé řešení problémů, pro něž není znám algoritmus nebo přesnější metoda. Takto získané řešení je často pouze přibližné, což je v souladu s výše uvedeným. Často se hovoří o *metaheuristikách* – tento pojem zdůrazňuje, že se jedná o obecnou metodologii, nikoliv konkrétní algoritmus pro konkrétní typ úlohy (tzv. *problem specific heuristics*) [9]. Metaheuristiky mají obvykle iterační charakter. Výjimkou jsou tzv. *hladové metody*, které začínají s prázdným řešením a v každém kroku je přiřazena hodnota jednoho prvku rozhodovacího vektoru až do nalezení úplného řešení. U iterativních metod je jejich hlavní charakteristikou strategie přechodu mezi řešeními v jednotlivých iteracích. Metody přitom mohou v každé iteraci pracovat s jedním řešením nebo s více řešeními (*population-based metaheuristics*). Kromě toho se mohou dále lišit i v jiných aspektech – inspirace přírodními procesy (evoluční algoritmy, společenstva – mravenci, včelstva, fyzika – simulované žíhání), používání resp. nepoužívání paměti již prohledaných oblastí, deterministické nebo stochastické metody apod. [59]

Mezi stochastické metaheuristiky pracující s více řešeními v každé iteraci patří *genetické algoritmy*. S ohledem na to, že právě tyto algoritmy jsou použity k řešením úloh v kapitole 5 (a také proto, že se jimi autor dlouhodobě zabývá), je právě jejich detailnímu popisu věnována převážná část této kapitoly.

3.1 Východiska evolučních algoritmů

Umělé evoluční systémy, mezi které genetické algoritmy patří, jsou inspirovány přírodními procesy. Evoluční teorie vysvětlují původ a proměny živých obyvatel Země v průběhu vývoje až do současnosti jako přirozený proces [6]. Začátky vědeckých diskusí o přírodní evoluci trvají již více než dvě stě let [45, 31]. Možnost přírodní evoluce naznačil francouzský přírodovědec Buffon[5], první evoluční teorii potom vytvořil jeho žák Lamarck[6]. Lamarck byl prvním, kdo myšlenku evoluce podpořil logickými argumenty a kdo formuloval první hypotézy týkající se mechanismů evolučních změn. Evoluci vysvětloval schopností změn živých organismů, a to postupně po mnoho generací zděděním struktury, která se stává větší a lépe vyvinutá jako výsledek trvalého používání, nebo naopak se stává

[4] z řečtiny, heuriskó = nalézt, objevit
[5] George-Louis Leclerc, Comte de Buffon, francouzský matematik a přírodovědec, 1707–1788
[6] Jean-Baptiste Pierre Antoine de Monet, rytíř de Lamarck, francouzský přírodovědec, 1744–1829

zmenšenou jako výsledek nepoužívání. Podle dnešních poznatků většina evolučních změn tímto mechanismem produkována nebyla.

Podle Darwinovy[7] teorie založené na přírodním výběru má vznik většího počtu jedinců, než je okolní prostředí schopné uživit, za následek vznik konkurence a „boj o přežití". Pokud ve skupině existují dědičné odchylky, potom v rámci dané populace přežijí jedinci s výhodnějšími odchylkami a budou se dál rozmnožovat na úkor jedinců s méně vhodnými vlastnostmi. Díky tomu přežijí nejsilnější jedinci, takže daný biologický druh se postupně adaptuje na okolí. Změnám životního prostředí se tak biologické druhy postupně přizpůsobí. Darwin si uvědomil, že přirozeným (přírodním) výběrem se nedá vysvětlit vše, takže kromě přírodního výběru zavedl teorii o vlivu používání a nečinnosti a dědičnost vlastností získaných transmutací druhů. Je však zřejmé, že obě tyto původně Lamarckovy myšlenky nezapadají do teorie přírodního výběru a Darwinovi stoupenci, kteří považovali nedostatečné teoretické vysvětlení dědičnosti a odchylek za nejslabší článek teorie přírodního výběru, je ve velké většině případů odmítali.

Chybějící článek v Darwinově teorii doplnil Mendel[8], jehož práce byla znovu objevena na přelomu 19. a 20. století po téměř čtyřiceti letech opomíjení. (Udává se, že sám Darwin obdržel jeden výtisk Mendelovy publikace, odložil ji však do knihovny a nikdy se do ní nepodíval [12].) Mendelova teorie se obvykle podává ve formě tzv. Mendelových zákonů: zákona o uniformitě míšenců, zákona o štěpení znaků a zákona volné kombinovatelnosti.

Mendelova genetika se vztahuje k vzorům dědičnosti organismů se sexuální reprodukcí. Nositelem dědičných vlastností jsou tzv. *chromozomy*. U živých organismů je nositelem informace o dědičném znaku *gen*, který obsahuje *alely*. Pro každý znak existují dvě párové alely. Při křížení získává gen potomka po jedné alele od každého z rodičů. Výsledný gen potomka potom obsahuje buď obě alely stejné (vzniká *homozygot*), nebo křížením získal alely různé (*heterozygot*). Chromozomy, které obsahují genetické informace v podobě párových alel, se nazývají *diploidní chromozomy*, chromozomy, které tuto vlastnost nemají, jsou *haploidní chromozomy*.

Souhrn genetických informací obsažených v chromozomech organismu se nazývá *genotyp*, zatímco vnější projev znaků a vlastností organismu se nazývá *fenotyp*. Často lze pozorovat podobný či dokonce identický vzhled (fenotyp) jedinců, kteří mají různý genotyp. Existují alely tří typů: *dominantní*, *recesivní* a *kodominantní*. Dominance v genetické terminologii znamená schopnost jedné alelické formy určovat fenotyp heterozygotního jedince. Recesivní alela je potom taková alela, jejíž výskyt sám o sobě fenotyp ovlivnit nedokáže, její projev může být zastíněn dominantní alelou. O kodominantních alelách se potom hovoří v případě, kdy současný výskyt dvou dominantních alel vede k fenotypickému projevu odlišnému než samostatný výskyt jedné z nich (např. krevní skupina AB je fenotypickým projevem současného výskytu dominantní alely pro krevní skupinu A a dominantní alely pro krevní skupinu B). Není důležité, zda dominantní alela

[7] Charles Robert Darwin, britský přírodovědec, 1809–1882
[8] Johann Gregor Mendel, přírodovědec, opat Augustiniánského kláštera v Brně, 1822–1884

pochází od matky nebo otce. Dědičnost ale může být pohlavně vázaná – zděděný znak se pak projevuje pouze ve fenotypu jedinců určitého pohlaví.

Všechny buňky v těle živého organismu mají stejný počet chromozomů, tento počet je konstantní u všech organismů stejného druhu. Většina organismů má sudý počet chromozomů srovnaných v párech (diploidní uspořádání). Výjimkou jsou pohlavní buňky, které mají poloviční počet chromozomů (jsou haploidní). Uvedeným mechanismem se při zachování relativní stability fenotypu druhu zajišťuje uchování dostatečné rozmanitosti genetické informace takové, aby byl umožněn další vývoj. Sexuální rekombinace pak vytváří nepřeberné množství různých genotypických kombinací, které zvyšují evoluční potenciál populace. Vzhledem k tomu, že sexuální reprodukce zvyšuje rozmanitost genetického materiálu neseného potomky totožných rodičů, zvyšuje tím i šanci, že alespoň někteří z nich budou úspěšní i v proměnlivém a často nepředvídatelném prostředí. Sexuální reprodukce nemá ale vliv na rychlost evolučních změn [32, 33, 34].

K činitelům evoluce lze kromě samotného genetického křížení (tzn. výběru, který gen je zděděn od kterého rodiče) řadit také mechanismus výběru rodičů, mutaci, migraci a přírodní výběr. V přírodě lze nalézt různé sociální (rozmnožovací) systémy – monogamie, promiskuita, harémy, stáda, hejna atd., které můžeme chápat jako kolbiště střetu zájmů samců a samic [2] a které představují odlišné strategie výběru rodičů. Přírodní strategie výběru partnerů pro rozmnožování se projevuje vzorem chování samců („věrný“ vs. „záletník“) a samic („zdrženlivá“ vs. „nevázaná“) a jednotkou, ve které daný druh převážně žije (osamělí jedinci, páry, tlupy, smečky, stáda, kolonie, …). Důležitou roli hraje též sociální uspořádání v jednotce, které má obvykle souvislost s velikostí skupiny. Populace mnoha druhů jsou rozděleny do mnoha subpopulací, které jsou alespoň částečně vzájemně izolovány. Tyto subpopulace mohou přežívat v odlišných podmínkách a těmto podmínkám se alespoň částečně přizpůsobit. V důsledku migrace pak může občas docházet k přenosu genetických informací mezi subpopulacemi.

Důležitou roli hraje v procesu evoluce délka života, stárnutí a smrt. Existují tři kauzální teorie stárnutí: teorie prospěchu druhu, teorie životního tempa a evoluční teorie stárnutí [2, 35, 63]. Podle *teorie prospěchu druhu* je pro druh výhodné, když se přizpůsobí svému prostředí. Prostředí se však neustále mění, zejména vlivem jiných organismů, které jsou pod neustálým vývojovým tlakem a musí se adaptovat co nejlépe. Každý druh musí držet krok s evolučními změnami jiných organismů, a proto musí vznikat nové mutace a nové kombinace genů. Tyto kombinace se mohou objevit a uplatnit pouze tehdy, když jedna generace vymře a je nahrazena druhou. Podle některých autorů je tato teorie z několika důvodů mylná [2, 6]. *Teorie životního tempa* předpokládá, že stárnutí je závislé na rychlosti žití. Vycházela z porovnání vztahů metabolismu, tělesné váhy (velikosti zvířat), tělesné teploty a délky života. Větší zvířata obvykle žijí déle než malá, ale přímá závislost mezi délkou života a poměrnou rychlostí metabolismu neexistuje. Podle *evoluční teorie stárnutí* přírodní výběr nemá vliv na nevýhodné vlastnosti (např. sklony k chorobám), které se

projevují až ve věku vyšším než reprodukčním. Navíc se některé geny mohou v nižším věku projevovat pozitivně, ve vyšším negativně (typicky např. pohlavní hormony).

3.2 Princip genetických algoritmů

Genetické algoritmy (GA) jsou řazeny mezi stochastické heuristické optimalizační metody [23, 24]. GA jsou inspirovány adaptačními a evolučními mechanismy živých organismů nastíněnými v odstavci 3.1, ovšem nekopírují tyto přírodní procesy zcela přesně. Největší uplatnění nacházejí u řešení problémů vícerozměrné optimalizace, pro které není známé (nebo je velmi složité) analytické řešení a kde není známa vhodná efektivní numerická metoda.

Obecná formulace optimalizační úlohy může být vyjádřena např. vztahem

$$\mathbf{x}_{opt} \in \arg\operatorname{opt}\left\{ f(\mathbf{x}) \,\middle|\, \mathbf{x} \in C \right\} \tag{3.1}$$

kde f je účelová funkce, C je množina přípustných řešení, $\mathbf{x}$ je přípustné řešení a $\mathbf{x}_{opt}$ je hledané optimální řešení. Jak již bylo uvedeno, důležitá je strategie procházení množiny C. GA používá pro generování možných řešení strategii inspirovanou přírodním darwinistickým vývojem. Vychází se z metody přírodního výběru, kdy největší šanci na přežití a předání svého genetického materiálu dalším generacím mají jedinci nejlépe uzpůsobení daným podmínkám. Kromě toho GA zohledňují ještě další v přírodě existující mechanismus – mutaci, kterým se usnadňuje možnost adaptace na měnící se podmínky a který rovněž omezuje nebezpečí degenerace (z hlediska optimalizace lze pro degeneraci vidět obdobu v uváznutí v lokálním extrému).

GA mají iterační charakter. V jednotlivých iteracích se nepracuje s izolovaným řešením, ale s tzv. *populací*. Pro GA stejně jako pro všechny evolucí inspirované algoritmy je charakteristické „populační“ prohledávání prostoru možných řešení. GA pracuje v každé iteraci s několika (obecně mnoha, není výjimkou populace o velikosti stovek jedinců) řešeními obsaženými v populaci a snaží se pomocí genetických operací s těmito řešeními zajistit výskyt stále lepších řešení v další iteraci na základě předpokladu, že kombinací dobrých vlastností dvou jedinců se mohou „rodit“ jedinci, kteří převýší své předky [61].

Populace je tvořena jistým vhodným (po dobu běhu algoritmu zpravidla neměnným) počtem prvků (jedinců), obsahem těchto prvků jsou řešení uložená v *chromozomech*. Pro každého jedince obsaženého v populaci je vypočítána odpovídající hodnota účelové funkce. Generování populace pro další iteraci se provádí na základě stávající populace genetickými operacemi, které se nazývají *křížení* a *mutace*. V operaci křížení se chromozom nové populace získává kombinací dvou chromozomů stávající populace. Řešení vstupující do této operace jsou stanovována náhodně, ovšem tak, aby pravděpodobnost použití každého člena populace v operaci křížení byla úměrná kvalitě jeho hodnoty účelové funkce. Na takto

získanou populaci se aplikuje operace mutace a postup pokračuje další iterací. Iniciální populaci pro první iteraci lze vygenerovat např. náhodně.

Genetické algoritmy jsou zjednodušením biologických mechanismů naznačených v 3.1 a jejich řešení je možné (s výjimkou triviálních ukázkových příkladů) pouze s využitím počítače. To je jedním z důvodů, proč pro GA není ani po letech ustálen formální matematický popis ani odpovídající symbolika. Popis algoritmů bývá v literatuře uváděn slovně (případně včetně ukázek konkrétní implementace v nějakém programovacím jazyce). Je ovšem třeba podotknout, že zavedení symboliky má pro GA pouze omezený význam pro specifikaci některých základních konceptů nebo upřesnění některých dílčích kroků – s ohledem na rozmanitost možností implementačních detailů a odchylek (viz 3.3) neexistuje jednotná úplná symbolika, která by umožňovala popsat činnost genetického algoritmu do všech detailů (do podoby, ze které by šlo přímo vycházet při realizaci implementace GA v programovacím jazyce). Symbolika používaná v této kapitole vychází z pramenů [24] a [61] a byla sjednocena v [49] a [52].

Genetický algoritmus je stochastický heuristický algoritmus obsahující následující operátory a parametry:

$$GA = (N, P, f, \Theta, \Omega, \Psi, \tau) \tag{3.2}$$

kde P je populace obsahující N prvků (jedinců, individuí): $P = \{S_1, S_2, \dots, S_N\}$. Každý prvek $S_i, i = 1, \dots, N$ je řetězec (nebo množina) celých čísel pevné délky n reprezentující řešení problému, tedy obecně platí $S_i \in Z^n$.

Symbol f označuje *účelovou funkci* (*fitness function*), která přiřazuje každému prvku reálné číslo:

$$f : S_i \mapsto R, \quad i = 1, \dots, N. \tag{3.3}$$

Θ je operátor *výběru* (*selekce*) *rodičovských prvků* (*parent selection operator*), který vybere u prvků z P:

$$\Theta : P \to (P_1, P_2, \dots, P_u). \tag{3.4}$$

Ω je množina genetických operátorů, zahrnující operátor *křížení* (*crossover*) Ω_c, operátor *mutace* (*mutation*) Ω_m a případně další problémově nebo implementačně specifické operátory, které všechny dohromady generují z u *rodičů* celkem v *potomků* (*offsprings, children*):

$$\Omega = \{\Omega_c, \Omega_m, \dots\} : \{P_1, \dots, P_u\} \mapsto \{O_1, \dots, O_v\}. \tag{3.5}$$

Ψ je operátor *redukce* (*deletion*), která odstraní v vybraných prvků z aktuální populace $P(t)$, kde t je identifikace iterace. Poté je v potomků přidáno do nové populace $P(t+1)$:

$$P(t+1) = P(t) - \Psi(P(t)) \cup \{O_1 \dots, O_v\}. \tag{3.6}$$

τ je kritérium ukončení:

$$\tau : P(t) \to \{true, false\}. \tag{3.7}$$

Operátor výběru rodičů Θ a genetické operátory Ω mají pravděpodobnostní charakter, operátor redukce Ψ je zpravidla deterministický.

Jedním ze základních konceptů GA je vymezení pojmu *chromozom*. Podle provedení GA může být chromozom jediným obsahem jedince S_i, nebo může každý jedinec obsahovat více chromozomů; kromě chromozomu (chromozomů) mohou být v některých implementacích GA v prvcích populace obsaženy i další informace. Chromozom $\mathbf{x}$ je abstraktní matematický objekt, který v reálné aplikaci může představovat vektor hodnot (reálných nebo celých čísel), popis grafové struktury, řetězec symbolů apod. Chromozomy jsou složeny z *genů*, gen lze chápat jako prvek vektoru $\mathbf{x}$.

Předpokládáme zpracování na počítači, takže chromozom bude fyzicky reprezentován posloupností bitů. Tato posloupnost bude nutně mít konečnou délku, takže množina X všech přípustných chromozomů bude konečná.

$$X = \{\mathbf{x}, \mathbf{x}', \mathbf{x}'', \ldots\} \tag{3.8}$$

Pokud prvek populace S_i obsahuje právě jediný chromozom, lze účelovou funkci (3.3) přepsat do tvaru:

$$f : X \to R \tag{3.9}$$

a řešení celého problému lze chápat jako optimalizační úlohu (hledání globálního minima) účelové funkce f:

$$\mathbf{x}_{opt} \in \arg\min_{\bar{x} \in X} f(\mathbf{x}) \tag{3.10}$$

Cílem evoluce je vytvořit chromozom, který se co nejvíce blíží (nebo je dokonce roven) optimálnímu chromozomu $\mathbf{x}_{opt}$.

Pravděpodobnostní charakter operátorů křížení a mutace je realizován pomocí dvou pravděpodobností P_c a P_m. Aplikace operátoru mutace na chromozom musí být realizována tak, aby platilo [61]

$$\lim_{P_m \to 0} \Omega_m(\mathbf{x}) = \mathbf{x} \ . \tag{3.11}$$

Pravděpodobnost křížení P_c udává pro každý chromozom (obecně pro každý prvek populace), s jakou pravděpodobností bude operátorem Θ vybrán k reprodukčnímu procesu právě tento chromozom. Hodnota P_c je tedy pro každý chromozom jiná a její hodnota záleží na hodnotě účelové funkce pro každý chromozom, přesněji na *přínosu* (*fitness*) každého chromozomu k úspěšnosti celé populace. Přínos F chromozomu pro úspěšnost populace P je nezáporné reálné číslo určené mapovací funkcí F:

$$F : P \to R^+ \tag{3.12}$$

která splňuje podmínku:

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \Rightarrow F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \tag{3.13}$$

Vztah (3.13) platí samozřejmě pro hledání minima. Závislost mezi hodnotou účelové funkce a přínosem obvykle bývá, ale nemusí nutně být, lineární. Normalizovaný přínos je potom určen jako

$$F'(\mathbf{x}) = \frac{F(\mathbf{x})}{\sum_{\tilde{\mathbf{x}} \in P} F(\tilde{\mathbf{x}})}. \tag{3.14}$$

Pochopitelně platí

$$\forall \mathbf{x} \in P: \quad 0 \leq F'(\mathbf{x}) \leq 1 \tag{3.15a}$$

$$\sum_{x \in P} F'(\mathbf{x}) = 1 \tag{3.15b}$$

Normalizovaný přínos bývá často interpretován přímo jako pravděpodobnost výběru každého chromozomu pro křížení:

$$P_c(\mathbf{x}) = F'(\mathbf{x}) \tag{3.16}$$

a tím i míra zastoupení jím nesené genetické informace v další generaci (v populaci v dalším iteračním kroku GA). Aplikace hodnoty P_c je realizována implementací selekčního operátoru Θ.

Základní schéma běhu genetického algoritmu je potom následující:

1. Vygenerování počáteční populace (obvykle náhodně).
2. Ohodnocení jednotlivých členů populace pomocí účelové funkce.
3. Selekce určeného počtu rodičů a vygenerování stanoveného počtu potomků aplikací operátoru křížení na vybrané rodiče, ohodnocení potomků.
4. Sestavení nové populace (redukcí stávající populace a začleněním potomků).
5. Mutace a nový výpočet hodnot účelové funkce pro mutované jedince.
6. Není-li splněna podmínka ukončení, opakovat od kroku 3.
7. Za výsledek je považován jedinec s nejlepší hodnotou účelové funkce.

Je zřejmé, že uvedené schéma je (stejně jako zavedení jednotlivých operátorů v předchozím textu) pouze ideové. Pro konkrétní realizaci GA je třeba vyřešit celou řadu implementačních detailů.

3.3 Implementace GA

Navzdory jednoduchému principu jsou návrh a realizace GA pro úspěšné praktické použití překvapivě komplikované. GA má mnoho parametrů a možností, jejichž vhodné nastavení je silně závislé na charakteru řešeného problému. Rozmanitost je zde velmi vysoká. Na

jedné straně se jedná o stanovení hodnot kvantitativních parametrů, jako je např. velikost populace nebo pravděpodobnost mutace. Na druhé straně je nutné řešit poměrně komplikovaná rozhodnutí týkající se datových struktur (různý charakter problému vyžaduje různý způsob kódování hodnot do chromozomů) i algoritmických řešení. Samozřejmě existuje celá řada známých řešení a doporučení, která jsou stručně shrnuta např. v [58], ale přesto je úspěšná a efektivní implementace GA pro daný účel programátorskou výzvou. GA tak tedy v reálu představuje spíše filozofii přístupu k řešení problému a rodinu metod, než jeden konkrétní pevně daný algoritmus. Zejména toto je důvodem, proč je obtížné porovnávat a hodnotit výkonnost různých implementací GA, proč je až na triviální případy nemožné popsat GA symbolicky tak, aby to bylo užitečné a odpovídalo realitě, a konečně, proč jsou výsledky jednotlivých autorů jen obtížně reprodukovatelné, pokud nebyla detailně popsána implementace GA včetně datových struktur. Jeden z možných přístupů k porovnávání GA lze nalézt v [20].

V této kapitole jsou uvedeny obecné principy týkající se možností implementace jednotlivých detailů GA. Klasifikace jednotlivých vlastností se způsobem členění drží klasického přístupu (např. [61, 52, 49]).

3.3.1 Chromozomy

Datová reprezentace chromozomu je jednou ze základních vlastností implementace GA. Formulace optimalizační úlohy (3.1) spolu se vztahem (3.8) určuje, že řešení je hledáno z konečné množiny vektorů parametrů (množina přípustných řešení), které nabývají diskrétních hodnot. Vektory parametrů jsou uspořádány do řetězců v chromozomech.

Chromozomy bývají chápány jako složené z genů. Význam genu není zcela pevně definován. Některými autory [61] je gen chápán jako prvek vektoru parametrů, tedy chromozom je složen z tolika genů, kolik prvků má vektor řešení. Jiná možnost je chápat gen jako základní stavební jednotku chromozomu bez ohledu na hranice prvků vektoru řešení, např. [49, 34, 55]. V těchto publikacích je gen jednotkou, která reprezentuje jeden bit, přitom samotný gen může být fyzicky tvořen jedním nebo několika bity. Toto pojetí je rozšířením klasického Hollandova genetického algoritmu [14]. Dekódování jednotlivých prvků vektoru řešení z chromozomu na základě hodnot jednotlivých genů je potom záležitostí až výpočtu účelové funkce.

Velký význam je nutné přikládat použitému kódování údajů. Vzhledem k uložení dat v počítači jsou hodnoty genů tvořeny posloupností bitů. Jedná-li se o číselné údaje, jeví se jako zdánlivě nejschůdnější použití přirozeného dvojkového kódu. Přirozený dvojkový kód má ale pro aplikace genetických operátorů některé nevýhodné vlastnosti plynoucí ze skutečnosti, že blízké (dokonce sousední) hodnoty se mohou lišit ve velkém počtu bitů, hovoří se o tzv. *Hammingově bariéře*. Např. uvažujeme-li chromozom délky osm bitů, tak pokud např. optimálnímu řešení odpovídá chromozomu $\alpha = (01000000)$, GA evolucí dospěl k nejlepšímu dosud nalezenému řešení $\beta = (00111111)$ a tato hodnota chromozomu v populaci převládá, obvykle se nepodaří genetickými operátory křížení

a mutace dospět ke správnému řešení (genetický operátor nedokáže provést transformaci chromozomu β na chromozom α). Příčinou je to, že hodnoty reprezentované chromozomy α a β jsou sice blízké (dokonce sousední), ale jejich Hammingova vzdálenost (počet lišících se bitů) je velká. Proto se v tomto případě obvykle používá *Grayův kód* [24]. Je třeba připomenout, že použití Grayova kódu může činit potíže v situacích, kdy bitové pole chromozomu má význam čísla v pohyblivé řádové čárce ve vnitřním formátu počítače. Obecně je při použití Grayova kódu nutné použít „mezitransformaci", která umožní převést vnitřní hodnotu chromozomu na hodnotu, která je chromozomem reprezentována a která je potřebná pro výpočet účelové funkce. Ačkoliv realizace této transformace není obtížná, může způsobit zvýšení výpočetní náročnosti a prodloužení doby výpočtu.

Jak bylo uvedeno v kapitole 3.1, u živých organismů je nositelem informace o dědičném znaku gen, který obsahuje alely. Pro každý znak existují dvě párové alely a každou alelu získává potomek od jednoho z rodičů, hovoří se o diploidních a haploidních chromozomech. Jedna z variant implementace popsaného jevu je uvedena v [30], podrobné porovnání s jinými variantami GA lze nalézt v [49]. Diploidní chromozom je realizován jako uspořádaná dvojice dvou bitových řetězců shodné délky *n*:

$$\mathbf{x} = (\mathbf{x}^1, \mathbf{x}^2) \in \left(\{0,1\}^n\right)^2, \tag{3.17}$$

kde $\mathbf{x}^1$, $\mathbf{x}^2$ jsou subchromozomy. Při výpočtu účelové funkce se náhodně zvolí subchromozom, pro který je hodnota skutečně vyčíslena. Pravděpodobnost výběru každého subchromozomu pro výpočet je 0,5. Operátor mutace se aplikuje separátně na každý subchromozom

$$\mathbf{x}' = \Omega_m(\mathbf{x}) = \left(\Omega_m(\mathbf{x}^1), \Omega_m(\mathbf{x}^2)\right) \tag{3.18}$$

Při křížení se z každého vstupního chromozomu náhodně vybere jeden subchromozom, křížení se aplikuje na vybraný pár subchromozomů, nevybrané subchromozomy se pouze beze změny kopírují.

Diploidních chromozomů se týká další vlastnost živých, a sice *dominance* a *recesivita*. Pokud k projevení dané vlastnosti stačí, aby byla alespoň jedna alela jistého typu (heterozygot se v dané vlastností shoduje s některým z homozygotů), je tato alela úplně dominantní a její párová alela je recesivní. Vlastnost se potom neprojevuje pouze u homozygotů, kteří mají obě alely recesivní. Realizace uvedeného principu vede k variantě GA, kde každý bitový řetězec obsahuje navíc ještě příznak, zda je dominantní nebo recesivní. Při výpočtu hodnoty účelové funkce má přednost dominantní subchromozom, jsou-li v chromozomu obsaženy oba subchromozomy recesivní nebo oba subchromozomy dominantní, určí se subchromozom pro vyčíslení účelové funkce náhodně (každý subchromozom má pravděpodobnost vstupu do výpočtu rovnu hodnotě 0,5). Operace mutace se opět aplikuje nezávisle na každý subchromozom zvlášť, pouze je nutné zajistit, aby mutací nebylo možné změnit příznak dominantnosti subchromozomu. Operace křížení probíhá podobně jako u obyčejných diploidních chromozomů, ovšem pro křížení je

přednostně použit dominantní subchromozom [47]. Dalším přiblížením k přírodním mechanismům je nevázat příznak dominance k celému chromozomu, ale přímo k jednotlivým alelám nebo genům.

V přeneseném slova smyslu lze pojmout haploiditu a diploiditu i jiným způsobem, a sice rozhodnout, že za haploidní budou považovány takové chromozomy, ve kterých jednomu bitu reprezentované hodnoty odpovídá právě jeden bit uložený v chromozomu, za diploidní (obecně se někdy hovoří o *polyploiditě* nebo *multiploiditě*) potom takové chromozomy, které jeden bit hodnoty kódují redundantně do více bitů chromozomu. GA používající toto multiploidní schéma chromozomů vykazují velmi zajímavé vlastnosti blížící se klasickým diploidním GA, výhodou ovšem je, že manipulace s chromozomy zůstávají stejně jednoduché jako u haploidních chromozomů.

Velmi zajímavý způsob zacházení s multiploidními chromozomy zavedl Ryan [55]. V podstatě se jedná o použití polyploidních chromozomů bez dominance. V této variantě je umožněno rodičům stejného fenotypu vytvářet potomky různého fenotypu. Chromozom je tvořen geny, každý gen navenek představuje jeden bit hodnoty reprezentované chromozomem. Gen samotný však obsahuje více než jeden bit (je použito redundantní kódování). Hodnoty bitů uvnitř genu jsou mapovány na vnější hodnotu genu (0 nebo 1) pomocí zvláštní mapovací funkce tak, že hranice mezi hodnotou 0 a 1 není ostrá, ale existuje jistá *„stínová“ zóna* (*shade zone*), ve které se hodnota nesená genem určuje náhodně. Situace je znázorněna na obr. 3.1.

Obr. 3.1: *Příklad struktury genu při použití stínů*

Text v tomto odstavci se věnoval převážně kódování dat v chromozomech na nejnižší úrovni, tj. na úrovni bitů. Na této úrovni lze ještě formulovat obecně platná doporučení do značné míry nezávislá na řešené úloze. Charakter řešené úlohy však hraje při implementaci GA pro konkrétní aplikaci podstatnou roli. Jiná je situace, kdy rozhodovací vektor má pouze reálné prvky, a zcela jiná pro řešení rozvrhovacích úloh. Kromě struktury a kódování chromozomu se pro tyto charakterem různé úlohy liší i parametry GA a také realizace některých operací. Podrobnosti lze nalézt např. v [52], [58] nebo [62].

3.3.2 Populace

Při volbě velikosti populace N je třeba volit kompromis mezi dvěma protichůdnými požadavky, a to jsou *rozmanitost* (diversity) a *rychlost konvergence* (rate of convergence). Je zřejmé, že při malé populaci bude počet jedinců s různým genotypem v populaci (genofond) malý a nemusí obsahovat informace, které povedou k nalezení globálního extrému. V tomto případě bude sice GA konvergovat rychle, ovšem často pouze k lokálnímu extrému. Nalezení extrému globálního je silně nepravděpodobné, neboť rekombinace chromozomů obsažených v populaci budou dávat jen málo rozmanitá řešení, která se budou pohybovat v okolí již nalezeného lokálního extrému. Možné vymanění z této „lokální pasti" může zajistit pouze operátor mutace, ovšem „správná" posloupnost mutací je dosti nepravděpodobná. Důsledkem malé diverzity v populaci je situace, kdy po určitém (obvykle malém) počtu iterací jsou všechny chromozomy v populaci stejné. Tento jev se označuje jako *degenerace* nebo jako *předčasná konvergence*. Originální řešení tohoto problému bude popsáno v odstavci 3.3.7.

Požadavek rozmanitosti vede tedy k volbě co největších velikostí populace. Důsledkem rozsáhlých populací je pochopitelně větší časová náročnost výpočtu, neboť k provedení jedné iterace bude nutné provést větší množství operací. Rovněž počet iterací nutný k nalezení globálního extrému je obvykle větší než počet iterací, po kterých dojde k degeneraci u GA s neúměrně malou velikostí populace (zde je ale nevhodné srovnávat počet iterací nutných pro úspěšný a neúspěšný běh GA).

Hypoteticky může nastat i případ, kdy velikost populace vzhledem k počtu možných různých řešení (množina C sice může být nekonečná, množina X je ale vždy konečná) je tak velká, že se správně řešení s vysokou pravděpodobností objeví už v iniciální populaci. Obecně to sice nelze považovat za závadu (cíl, kterým je nalezení řešení, byl dosažen), ovšem jinak se zcela jistě jedná o nevhodně zvolený prostředek pro daný problém, nehledě k tomu, že může nastat problém s podmínkou ukončení GA, neboť v průběhu jednotlivých iterací nebude docházet k žádnému zlepšování řešení.

Goldberg teoreticky dokázal [8], že pro binárně kódované řetězce optimální velikost populace exponenciálně roste s délkou řetězce. Pro většinu praktických problémů by to v důsledku znamenalo zpracovávat velmi rozsáhlé populace. Praktické zkušenosti však ukazují, že pro většinu problémů postačuje volba $N \in \langle 50{,}200 \rangle$ [1, 50].

Pro volbu počáteční populace se používají dva základní přístupy: buď se počáteční populace vygeneruje náhodně, nebo se získá jako množina známých vyhovujících řešení získaných pomocí jiné (např. heuristické) metody. V případě, že prvotní populace je generována náhodně, je výhodné, aby byla dodržena co největší rozmanitost jedinců. To lze zajistit například nepřipuštěním vstupu více totožných jedinců do počáteční populace.

Některé aplikace nepřipouštějí, aby všechny existující kombinace genů tvořily přípustná řešení. Problém se zajištěním přípustnosti všech řešení obsažených v populaci se projevuje i u realizace operátorů křížení a mutace, tam se ovšem obvykle řeší znevýhodněním těchto jedinců mimořádně nepříznivou hodnotou účelové funkce (penalizace).

Při generování počáteční populace je však vhodné zajistit, aby populace nepřípustná řešení vůbec neobsahovala. Důvodem je snaha zajištění co největšího množství „kvalitního“ genetického materiálu pro další evoluci.

GA, které používají v počáteční populaci řešení získaná jinou metodou, se nazývají *hybridní* GA. Pokud se v počáteční populaci používají výhradně známá dostatečně kvalitní řešení, hovoří se někdy o použití GA k doladění parametrů (*parameters tuning*). Někdy může být výhodné pro zajištění „rozběhnutí“ GA dodat do iniciální populace alespoň několik známých přibližných řešení, v jejichž okolí předpokládáme (nebo si přejeme, to nemusí být totéž) výskyt globálního extrému nebo alespoň dalších lepších řešení. I když předpoklad o poloze řešení nemusí být splněn, lze tímto způsobem někdy příznivě ovlivnit konvergenci GA. Příspěvek o vlivu inicializace počáteční populace lze nalézt např. v [27].

3.3.3 Účelová funkce

Nejjednodušším způsobem stanovení účelové funkce GA je použít přímo účelovou funkci řešeného problému. V zásadě je to možné, protože nečiní praktické potíže realizovat GA pro maximalizační i minimalizační úlohu. Někdy je ale vhodné pro účely GA tuto účelovou funkci modifikovat.

GA představuje pouze nástroj na prohledávání stavového prostoru a je poměrně obtížné zařídit, aby byla generována pouze řešení splňující jisté podmínky. Informaci o splnění těchto podmínek je vhodné zařadit také do účelové funkce. Operací křížení nebo mutace může totiž obecně vzniknout řešení, které nepatří do množiny přípustných řešení (nebo je sice přípustné, hodnotu účelové funkce lze vyčíslit, tato hodnota se dokonce může pohybovat v „rozumném“ rozsahu, ovšem řešení je z nějakých důvodů nežádoucí). Náprava tohoto jevu přímo v realizaci příslušného operátoru obvykle silně komplikuje implementaci, a tak bývá výhodnější tato nepřípustná (nebo jinak nežádoucí) řešení do populace zařadit, ovšem uměle jim přiřadit silně nevýhodnou hodnotu účelové funkce (tzv. penalizace). Toto řešení je samozřejmě vhodné pouze v případě, že vznik nepřípustných řešení není příliš častý, jinak by mohlo v extrémním případě dojít k zaplnění populace samými nepřípustnými řešeními. Za jistých okolností může být chybou vyřazovat automaticky z populace všechna v průběhu řešení vzniklá nepřípustná řešení, neboť vzhledem k charakteru genetických operátorů může být potomkem dvou nepřípustných řešení být řešení přípustné (nebo dokonce optimální).

3.3.4 Výběr rodičů, křížení a mutace

Mechanismus výběru rodičů hraje v GA jednu z klíčových rolí. Zdánlivě by mělo být nejvýhodnější pro reprodukci vybírat jedince s co nejlepší hodnotou účelové funkce. To je ovšem pravda jen do jisté míry – tímto přístupem se snižuje variabilita genetického materiálu a zvyšuje se riziku uváznutí GA v lokálním extrému. Podobně jako při volbě malé velikosti populace má v tomto případě GA sklon k rychlé degeneraci.

Nejčastěji používané strategie jsou:

1. *Proporcionální výběr* (*proportionate selection*) je založen na výpočtu normalizovaného přínosu (3.14), který určuje pravděpodobnost výběru pro reprodukci (3.16). Konkrétní realizace výběru se provádí buď pomocí losování analogického s „ruletou“ (každému jedinci je určena výseč rulety o velikosti odpovídající normalizovanému přínosu, celé „kolo“ rulety má rozsah $\langle 0,1 \rangle$, generuje se náhodné číslo z intervalu $\langle 0,1 \rangle$ a vybere se jedinec, kterému přísluší odpovídající výseč), nebo se vygeneruje pomocná populace o počtu N^* (obvykle platí $N^* > N$) taková, ve které jsou jedinci zastoupeni poměrně v počtu odpovídajícím jejich normalizovanému přínosu, a vylosování jedince se provádí generováním náhodného čísla v rozsahu $\langle 1, N^* \rangle$. Proporcionální výběr může být nevýhodný v případě, kdy se v populaci vyskytuje několik málo jedinců s vysokým přínosem, ovšem nepředstavujících optimální řešení. V dalších generacích převládnou jejich genetické vlastnosti a zvýší se nebezpečí uváznutí v lokálním extrému. Kromě toho tato metoda komplikuje programovou realizaci GA a v důsledku tak zpomaluje běh GA.
2. Při použití *uspořádaného výběru* (*ranking selection*) se jedinci v populaci seřadí vzestupně podle přínosu a výběrové pravděpodobnosti jedinců jsou odvozeny od jejich pořadí v takto setříděné populaci. V tomto případě nezáleží pravděpodobnost výběru jedince na velikosti přínosu, ale pouze na jeho pořadí v uspořádané populaci. Pokud je jedinec s nejlepším přínosem umístěn na poslední pozici v populaci a naopak, pak pro výběrové pravděpodobnosti platí:

$$P(S_i) = \frac{2i}{N(N+1)}, \; i = 1, \ldots, N \tag{3.19}$$

   Jako nevýhoda bývá uváděna nutnost výpočtu pravděpodobností výběru všech jedinců [61]. Tato metoda selekce však může být programově realizována velmi jednoduše bez výpočtu pravděpodobností např. tak, že při výběru i-tého rodiče se bude výběr provádět generováním náhodného čísla z intervalu $\langle i, N \rangle$; pokud je počet vybíraných jedinců (nebo párů jedinců) roven velikosti populace, jsou pravděpodobnosti podle (3.19) zachovány. Modifikací tohoto způsobu je strategie, kdy jako první rodič se postupně vybírají chromozomy v pořadí od „nejhoršího” člena populace směrem k „nejlepšímu“ členu populace, druhý rodič se generuje náhodným výběrem ze všech chromozomů „lepších“ než první rodič [26, 47, 46].
3. *Turnajový výběr* (*tournament selection*) je velmi často používaný způsob výběru. Každý rodič se vybírá tak, že se náhodně vybere k jedinců a z nich se vybere jedinec s nejlepší hodnotou účelové funkce. Praktické vlastnosti jsou srovnatelné s uspořádaným výběrem, pro výběrové pravděpodobnosti platí za předpokladu vzestupného uspořádání populace vztah:

$$P(S_i) = \frac{i^k - (i-1)}{N^k},\ i = 1,\ldots,N \tag{3.20}$$

Jako největší výhoda bývá pro turnajový výběr uváděna efektivnost (např. [61, 58]), neboť není nutno ani počítat výběrové pravděpodobnosti, ani generovat pomocnou populaci, ani setřiďovat populaci podle účelové funkce. Použije-li se však výše popsaná modifikace uspořádaného výběru, pak tyto výhody přestávají platit a implementačně nejjednodušší je modifikovaný uspořádaný výběr.

Způsob realizace operátoru **křížení** je pro chování GA velmi důležitý. Někdy je operátor křížení považován za nejvýznamnější operátor prohledávání [61]. Operátor křížení vytváří nového jedince (potomka) kombinací segmentů vybraných rodičovských prvků. Běžně jsou uváděny následující způsoby realizace operátoru křížení:

1. *Jednobodové křížení* (*one-point crossover*) se používá v klasických variantách GA. Princip spočívá v náhodném určení bodu křížení a výměně segmentů dvou rodičů. Tímto způsobem vznikají dvě různé varianty potomků, obvykle se ale dále pracuje pouze s jednou z nich.
2. *Vícebodové křížení* (*multi-point crossover*) lze chápat jako zobecnění jednobodového křížení použitím více než jednoho bodu křížení.
3. Při *uniformním křížení* (*uniform crossover*) je každý gen potomka získán kopírováním odpovídajícího genu jednoho nebo druhého rodiče, použití genů obou rodičů má stejnou pravděpodobnost, výběr se provádí podle binárního čísla náhodně vygenerovaného z alternativního rozdělení. Namísto genů se uniformní křížení může týkat přímo jednotlivých bitů.

Volba způsobu křížení a výběr bodu křížení je nutno stanovit s ohledem na strukturu chromozomu (rozhodující je charakter dat uložený v chromozomu). Právě z charakteru aplikace a tvaru chromozomu vyplyne, zda lze bod křížení stanovit na zcela libovolném místě nebo např. pouze na hranicích jednotlivých genů.

Praktické zkušenosti ukazují, že při použití multiploidních chromozomů realizovaných pomocí stínů (viz. odst. 3.3.1) dává velmi dobré výsledky uniformní křížení [34].

Výše uvedené platí pro úlohy, kde prvky rozhodovacího vektoru jsou číselné hodnoty. Je-li GA použit např. pro řešení jiného typu úlohy, pak je nutné respektovat charakter této úlohy i v operaci křížení, např. každé řešení úlohy typu obchodní cestující musí respektovat požadavek, aby bylo každé místo navštíveno pouze jednou apod. Proto musí být jak struktura chromozomu, tak i operace křížení (a mutace) realizována s ohledem k řešené úloze [58].

Operátor **mutace** (mutation operator) s jistou malou pravděpodobností modifikuje každý gen chromozomu. Hlavním cílem mutace je snaha zabránit degeneraci (algoritmus nalezne lokální extrém a jen velmi málo chromozomů v populaci obsahuje genetický materiál, který by algoritmu umožnil tento lokální extrém opustit, v krajním případě nemusí takové informace obsahovat žádný chromozom).

V bitové reprezentaci bývá mutace konkrétně realizována invertováním příslušného bitu. Existují však i jiné možnosti realizace tohoto operátoru. Například je možné měnit celou souvislou skupinu bitů (do jisté míry může být i toto způsob překonání Hammingovy bariéry). Pro některé aplikace, kde hodnota každého genu představuje vícebitovou hodnotu jakožto jeden z prvků vektoru řešení, může být výhodné nerealizovat mutaci jako bitovou operaci, ale jako změnu hodnoty genu o jistou náhodnou („malou“) hodnotu [26], limit pro tuto hodnotu je v tomto případě druhým parametrem mutace. I tato modifikace operátoru mutace může mít za následek snadnější překonání Hammingovy bariéry.

Dolní mez pravděpodobnosti mutace bývá obvykle udávána hodnotou $1/n$, kde n je délka chromozomu (měřená v bitech nebo v genech). Horní mez se obvykle neudává. Platí, že při větší pravděpodobnosti mutace vykazuje GA lepší schopnost vyváznout z pasti lokálního extrému. Na druhé straně je ale nutno podotknout, že při příliš velké hodnotě pravděpodobnosti mutace mění GA svůj charakter, mutace se stává hlavní hybnou silou evoluce nových řešení a význam křížení naopak klesá. Tím se mění charakter prohledávání stavového prostoru a přibližuje se např. strategii *slepého prohledávání*.

Zajímavé výsledky dává GA s proměnnou pravděpodobností (příp. i limitem) mutace – pokud se po jistý počet iterací nedaří zlepšit dosavadní nalezené řešení, je možné zvýšením hodnot parametrů mutace docílit zrychlení hledání řešení (po „rozběhnutí“ algoritmu lze doporučit parametry mutace upravit na původní hodnoty) [28].

3.3.5 Změna populace

Změnou populace (*replacement scheme*) se rozumí strategie, která je používána pro vytvoření populace pro další iterační krok GA. V této nové populaci se mohou vyskytovat jak jedinci vygenerovaní operátorem křížení, tak i jedinci převzatí z minulého iteračního kroku. V literatuře bývají uváděny dvě základní strategie:

1. *Generační výměna* (*generational replacement*) spočívá v náhradě všech prvků populace novými jedinci vzniklými operací křížení. To znamená, že křížením musí být v každém kroku vygenerován počet jedinců rovný velikosti populace.
2. *Inkrementální náhrada* (*steady-state replacement, incremental replacement*): Křížením se vygeneruje pouze několik nových prvků (méně než je velikost populace), tyto prvky nahradí odpovídající počet prvků ve stávající populaci. K výběru nahrazených (zrušených) prvků se používají následující strategie:
   a) *Náhrada náhodně vybraného prvku* (*randomly chosen member replacement*) – tato strategie je používána pouze zřídka, vlastnosti GA se v tomto případě výrazně neodlišují od GA používajících generační výměnu (mohou ale vykazovat pomalejší konvergenci vyjádřenou počtem iterací, výpočetní doba jedné iterace je ale zase naopak kratší).
   b) *Náhrada nejhoršího prvku* (*least-fit member replacement*).
   c) *Náhrada náhodně vybraného prvku s podprůměrnou hodnotou účelové funkce*.

U strategie změny populace (zejména u inkrementálních náhrad) bývá někdy požadována podmínka, aby do populace nebyl zařazován prvek shodný s prvkem, který je v populaci již obsažen [61]. Opět se jedná o snahu zabránit degeneraci populace.

Strategie použitá pro změnu populace významným způsobem ovlivňuje chování GA. Předně může odpovědět na otázku, zda GA se při konvergenci chovají jako *monotonní* [23]. Je zřejmé, že neexistuje žádná záruka, že řešení získaná operátory křížení a mutace nebudou vykazovat horší hodnoty účelové funkce, než měla řešení v minulém iteračním kroku (rodičovská populace). Z toho důvodu při použití generační výměny ani náhrady náhodně vybraného prvku nelze monotonnost konvergence zaručit.

Jednou z možností, jak zajistit monotonnost a přitom používat libovolnou strategii změny populace je použití tzv. *elity* [23]. Elita je představována nejlepším doposud nalezeným řešením (jedním nebo několika různými). Elita beze změny přechází do další iterace. Složení elity se postupně mění se zlepšováním nalezeného řešení. Smyslem zavedení elity je vcelku samozřejmý názor, že není účelné zbavovat se nejlepšího dosud nalezeného řešení.

Dále pak může strategie změny populace silně ovlivnit i schopnost GA vyváznout z lokálního extrému a chování GA při řešení problémů, jejichž parametry jsou proměnné v čase. Z těchto hledisek je výrazným zlepšením dosavadně používaných strategií změny populace využívající operátor – *smrt* (*death*) [48, 49]. Tento operátor bude popsán v odstavci 3.3.7.

3.3.6 Kritérium ukončení

Za nejvhodnější podmínku ukončení běhu GA by bylo možné považovat dosažení řešení nebo alespoň přiblížení se k řešení s požadovanou přesností. Tento přístup lze ale používat pouze při testování algoritmů. Při použití GA k řešení praktických úloh je řešení neznámé, takže jeho dosažení explicitně testovat nelze. Jako podmínku ukončení běhu GA lze v analogii k iteračním metodám numerické matematiky používat:

a) porovnání výsledků generovaných několika posledními iteracemi (mezigenerační zlepšení kriteriální funkce)
b) maximální počet iterací (generací) GA

Obě uvedené varianty však vzhledem ke stochastické povaze GA a k charakteru chování GA nemusejí vyhovovat [47].

Zkušenosti ukazují, že zlepšování řešení obvykle neprobíhá v každé iteraci. Pozorovaná stagnace mezigeneračního vývoje nejlepší nalezené hodnoty kriteriální funkce potom může znamenat jednu z následujících situací:

i) Jedná se o přirozený dočasný jev, GA bude úspěšně pokračovat v řešení.
ii) GA uváznul v lokálním extrému, pravděpodobně došlo k degeneraci, další úspěšný vývoj je ohrožen.
iii) GA nalezl nejlepší možné řešení, další zlepšování již nemůže nastat.

Je zřejmé, že v prvních dvou popsaných situacích nebylo řešení nalezeno. Použití podmínky a) jako kritéria ukončení tedy naráží na nemožnost klasifikace situace iii).

Prvořadým cílem kritéria ukončení podle varianty b) je zajistit vůbec nějaké ukončení algoritmu, aby nemohlo docházet k nekonečným smyčkám. Tato základní myšlenka může být rozšířena o hledání „typického počtu iterací“ potřebného pro nalezení globálního extrému. Bude-li tento počet nalezen, lze jej použít jako přídavné kritérium k podmínce formulované podle bodu b). Je zřejmé, že typický počet iterací se bude lišit podle konkrétního problému a varianty GA. Použijme analogii. Je známé, že problémy lineárního programování jsou řešitelné v polynomiálním čase (např. [4]). Rovněž je známé, že v teoreticky nejhorším možném případě vyžaduje Dantzigova simplexová metoda exponenciální počet iterací. Experimenty ale ukázaly, že většina praktických úloh vyžaduje k získání řešení významně nižší počet iterací. Nabízí se tedy pokusit se odvodit přídavnou podmínku ukončení ze statistické charakteristiky potřebného počtu iterací pro typický problém a implementaci GA.

K řešení problému stejné struktury a složitosti může stejný GA vyžadovat různý počet iterací. Počet iterací závisí na konkrétních datech a na řízení algoritmu. Počet iterací může být považován za diskrétní náhodnou proměnnou η. Pokud by byl algoritmus řízen deterministicky, bylo by možné získat informace o η analýzou dostatečného množství dat. Neurčitost by byla důsledkem neznalosti deterministického popisu závislosti počtu iterací na konkrétní specifikaci problému typu GA.

Jak již bylo uvedeno, pro GA jakožto stochastický heuristický algoritmus platí, že opakované spouštění stejné varianty GA pro řešení téhož problému může k nalezení řešení vyžadovat provedení různého počtu iterací. Tato druhá příčina „náhodné doby života algoritmu“ spočívá v častém použití generátoru náhodných čísel jakožto přímého zdroje neurčitosti, který způsobuje neopakovatelnost výsledků. Řízení algoritmu se stává stochastickým a požadovaný typický počet iterací musí být odvozen od náhodné proměnné η. Lze zvolit pravděpodobnost α a pak určit potřebný počet iterací n vztahem:

$$\min\left\{n \in N \mid P(\eta > n) \leq \alpha\right\} \tag{3.21}$$

Akceptovatelným řešení je $n = \text{round}(\eta_{1-\alpha})$, kde $\eta_{1-\alpha}$ je $1-\alpha$ kvantil pravděpodobnostního rozložení η. Jako podmínku ukončení GA lze použít dosažení n iterací. Další krok nyní musí směřovat k identifikaci neznámého rozložení η.

Považujeme-li stochastický heuristický algoritmus za komplexně řízený náhodný proces [37], lze předpokládat, že některá z teoretických rozdělení (např. normální nebo Weibullovo) budou pro tento účel vhodná. Pak by se problém identifikace η redukoval po úspěšném Pearsonovu χ^2 testu na zjištění parametrů tohoto rozdělení. Tato úvaha však není vždy použitelná. Předpoklad normálního rozdělení není obecně splněn. Počet iterací nemůže být popsán pomocí lineární kombinace primárních zdrojů neurčitosti. Také analogie počtu iterací GA do nalezení řešení (jakožto doby života GA) s problematikou životnosti selhává, neboť GA nemůže ukončit svůj život po první „nehodě“, tedy nalezení

prvního lokálního extrému, ale je snaha v běhu GA pokračovat po jisté „revitalizaci“. Obvykle nezbývá než pracovat s empirickým rozdělením. Toto rozdělení lze obdržet experimentálně realizací dostatečně velkého počtu běhů GA pro daný problém, za základ poslouží získané histogramy počtu potřebných iterací pro získání výsledku. Z rozdělení η lze potom zjistit počet iterací n pro zvolené α. Podrobnější popis naznačené metody lze nalézt v [47].

Naznačený přístup lze použít i opačně, a to k „doladění“ parametrů GA případně k určení varianty GA vhodné pro řešení daného problému. V tomto případě je možné určit rozdělení η pro různé varianty GA, preferována bude pochopitelně varianta, která vykazuje nejnižší hodnotu n pro zvolené α a nejmenší rozptyl.

3.3.7 Operátor *smrt*

Operátor smrt je zvláštním způsobem realizace operátoru redukce Ψ. Jeho zavedení bylo původně inspirováno ryze programátorským pohledem na implementaci GA. Největším problémem používání GA pro řešení praktických problémů se ukazuje být degenerace. U složitých účelových funkcí s mnoha lokálními extrémy se ukazovala být manipulace se stávajícími „klasickými“ parametry GA jako nedostatečná pro zajištění konvergence ke globálnímu extrému. Rozborem populace bylo zjištěno, že GA má v závislosti na složení iniciální populace sklon zachytit se v lokálním extrému, a následně jsou z populace velmi rychle vytěsněny prvky s odlišným genetickým materiálem. Vymanění se z lokální pasti je při použití genetických operátorů křížení a mutace velmi komplikované a zdlouhavé. V horším případě (bohužel velmi častém) se GA z lokálního extrému vymanit v „rozumném“ počtu iterací nedokáže. Jako řešení této situace navrhují někteří autoři restartování algoritmu (např. [28]). Další z cest je zajistit rozmanitost v populaci např. použitím proměnlivé pravděpodobnosti mutace. Uvedená řešení jsou ovšem nasazována až jako reakce na nastalý fakt zachycení algoritmu v lokálním extrému a neschopnost tento extrém se stávajícím složením populace opustit. Obtížná je již samotná detekce této situace. Jak již bylo uvedeno v odstavci 3.3.6, zlepšování řešení obvykle neprobíhá v každé iteraci a pozorovaná stagnace mezigeneračního vývoje nejlepší nalezené hodnoty kriteriální funkce může, ale nutně nemusí, signalizovat že řešení bylo nalezeno, nebo že došlo k degeneraci. Je zřejmé, že detekovat degeneraci, kdy jedině má smysl zasahovat, je bez znalosti alespoň přibližné polohy hledaného řešení a bez analýzy rozmanitosti populace obtížné. Lze předpokládat, že úplný restart GA je velmi radikální řešení, které bude mít v každém případě za následek ztrátu genetické informace nashromážděné v populaci. Současně je zde riziko, že restart je předčasný, neboť se jednalo o situaci i) (viz 3.3.6) a po několika málo iteracích by došlo k nalezení lepšího řešení. Restart tedy může být přínosem, ale nikdy není jisté, zda nezpůsobil naopak zbytečný nárůst doby řešení.

Možnou variantou by bylo sledovat rozmanitost populace a v případě příznaků degenerace reagovat dodáním „čerstvé“ genetické informace (vybraná část populace by byla nahrazena novými náhodně vygenerovanými jedinci). Implementačně by se ale jednalo

o nepříliš schůdnou variantu, rozmanitost by musela být kvantifikována, kvantifikace by byla problémově závislá, vyhodnocování rozmanitosti by zpomalovalo běh algoritmu, bylo by nutné stanovit strategii, které jedince nahrazovat novými, atd.

Dosavadní úvahy se zabývaly pouze statickými problémy, u kterých se poloha globálního extrému nemění v čase. Při řešení problémů dynamických je kromě dostatečně rychlé konvergence GA důležitým parametrem i schopnost adaptace GA, tedy schopnost co nejrychleji zareagovat na změnu polohy globálního extrému. Intuitivně lze vyvodit, že kromě způsobu realizace genetických operátorů zde bude úspěšnost GA silně závislá na udržení co největší rozmanitosti populace (v tomto případě dokonce i v ustáleném stavu po nalezení řešení), aby jedinci obsažení v populaci co nejlépe pokryli celý stavový prostor (množinu přípustných řešení).

Řešení uvedených problémů nabízí zavedení nového operátoru **smrt** do strategie změny populace. Operátor předpokládá zavedení **omezené doby života** jedinců v populaci. Zjednodušeně řečeno přináší tento operátor kontinuální částečný restart GA. Činnost operátoru smrt lze charakterizovat následovně:

1. Je zavedena globální hodnota představující *limit doby života*.
2. Každý jedinec obsahuje čítač, představující *věk* jedince. Na začátku života jedince (vznik nového jedince použitím generátoru náhodných čísel nebo vznik potomka operací křížení) je věk nastaven na hodnotu 0.
3. Populace (resp. každý její jedinec) *stárne*, operace stárnutí je realizována inkrementací věku každého jedince v každé iteraci. Jedinci, jejichž věk přesáhne globální limit doby života, umírají, jsou z populace odstraněni a nahrazeni novými náhodně vygenerovanými jedinci (s věkem 0).
4. Věk jedince nemá vliv na výběr rodičů pro křížení ani na mutaci.
5. Samotný čítač věku nepodléhá křížení ani mutaci, jeho hodnota je měněna výhradně způsobem dle bodů 1 a 2.
6. Pro zajištění monotonnosti si GA zapamatovává nejlepší dosud nalezené řešení. Jedná se o obdobu používání elity, ovšem s tím rozdílem, že se jedinec, který je nositelem dosud nejlepšího známého řešení, nemusí vůbec vyskytovat v populaci (mohl být odstraněn pro překročení věkového limitu); dosavadní nejlepší známé řešení se tedy obecně nemusí účastnit reprodukce.

Popsaný mechanismus a vlastnosti GA využívajícího operátor smrt lze nalézt např. v [52, 49, 34, 33, 32].

3.3.8 Sexuální reprodukce

GA se sexuální reprodukcí využívá další analogii živé přírody. V přírodou používané sexuální reprodukci vzniká diploidní potomek spojením haploidních pohlavních buněk diploidních rodičů. Pro napodobení této strategie v genetickém algoritmu je třeba uvážit, jaké modifikace klasického GA může modelování sexuální reprodukce přinést. Prvním

důležitým rysem je existence diploidních chromozomů a související problematika dominance a recesivity (viz odst. 3.3.1). Druhým rysem je fakt, že v populaci nejsou všichni jedinci rovnocenní z hlediska účasti v reprodukčním procesu. Existují jedinci dvou typů (pohlaví), reprodukce je možná pouze odvozením potomka od rodičů, z nichž každý je jiného pohlaví. Pohlaví potomka je určováno děděním, pravděpodobnost vzniku potomka každého pohlaví je 0,5.

V GA, který modeluje sexuální reprodukci, je populace rozdělena do dvou stejně početných částí, jedna část představuje samčí a druhá samičí část populace. Příznak příslušnosti k pohlaví je uložen ve speciálním genu. Tento „pohlavní" gen podléhá křížení stejně jako ostatní části chromozomu, hodnota vzniklá křížením určuje, do které části populace bude jedinec zařazen. Pohlavní gen nepodléhá mutaci. Operátor výběru rodičů Θ je modifikován tak, aby potomek byl vytvářen křížením dvou rodičů, kde každý rodič je z jiné části populace. Pro výběr rodičů z každé části populace potom může být použita buď některá ze standardních strategií popsaných v odst. 3.3.4, nebo lze vycházet z analogií s některým z existujících sociálních společenství. Je třeba zdůraznit, že použitá výběrová strategie může být v každé části populace jiná. Tak lze modelovat např. promiskuitní nebo zdrženlivé chování, stáda vedená dominantním samcem, párový systém apod. Uvedené strategie jsou popsány např. v [34, 33, 32].

Na rozdíl od přírodních procesů lze v GA populaci dělit do více než dvou rodičovských subpopulací, tzn. lze pracovat s více než dvěma pohlavími. Podrobnosti lze najít v [64], Výsledky jsou zajímavé, nicméně významné zlepšení vlastností GA pozorováno nebylo.

3.3.9 Paralelní genetické algoritmy

Populační způsob prohledávání stavového prostoru obsahuje již ve svém principu paralelnost, kterou lze využít, pokud je výpočet prováděn s využitím odpovídajícího hw a sw vybavení. Nabízí se zejména paralelní ohodnocování jednotlivých členů populace, při vhodné programové realizaci si lze představit i paralelní provádění operátoru reprodukce.

Kromě zmíněného typu paralelismu, který se týká programové realizace algoritmu, lze uvažovat i o paralelním běhu několika GA. Vzhledem k tomu, že průběh práce algoritmu závisí na prvotní populaci a že chování GA je silně ovlivněno celou řadou implementačních detailů, byl vyzkoušen i tzv. *víceúrovňový distribuovaný GA* [29]. V této variantě je spuštěno současně několik GA (s různou počáteční populací, s různě nastavenými parametry, s různou implementací). Tyto GA jsou hierarchicky uspořádány na nadřízené a podřízené algoritmy obecně ve více vrstvách. Distribuované GA zavádějí další operátor – *migrace*. Migrací se rozumí předání vybraného (obvykle nejlepšího) chromozomu v populaci nadřízenému algoritmu. Nadřízený algoritmus může nabídnuté řešení zařadit do svojí populace buď bezpodmínečně, nebo pouze v případě, že splňuje jistou podmínku „kvality". Za řešení úlohy se považuje řešení dosažené algoritmem hierarchicky nejvyšší úrovně. U víceúrovňových distribuovaných algoritmů se s výhodou zavádí restart podřízených algoritmů v případě, že po jistý stanovený počet iterací nedosáhnou zlepšení řešení,

případně nenabídnou GA vyšší struktury žádné řešení nebo žádné jimi nabídnuté řešení není přijato. Porovnání časové náročnosti výpočtu resp. počtu iterací potřebných k nalezení řešení „obyčejným" a distribuovaným GA (počet iterací se u víceúrovňových distribuovaných GA stanovuje podle počtu iterací provedených algoritmem v nejvyšší hierarchické úrovni) výrazně hovoří ve prospěch tohoto uspořádání. Jako významný lze označit vliv uspořádání hierarchické struktury – počet vrstev, počet algoritmů v jednotlivých vrstvách. Zajímavé je pozorování přínosů různě implementovaných GA – ukázalo se, že počet přijatých příspěvků do vyšších vrstev není závislý na úspěšnosti resp. výkonnosti GA při sólovém nasazení na řešení daného problému. Podobně jako většina zde zmiňovaných vlastností GA nebyl ani tento jev nijak kvantifikován.

Pro dosažení výsledků popsaných v [29] byl víceúrovňový distribuovaný algoritmus realizován tak, že každý algoritmus (včetně algoritmu nejvyšší vrstvy) byl spuštěný na zvláštním počítači, předávání dat mezi vrstvami (migrace) probíhalo s použitím počítačové sítě. Toto poměrně komplikované uspořádání bylo zvoleno vzhledem k nízké výpočetní rychlosti tehdy dostupných počítačů, a lze je považovat spíše za nouzové. Za nejvhodnější lze v tomto případě považovat použití paralelního počítače.

4 DISTRIBUOVANÉ OPTIMALIZAČNÍ PROGRAMY

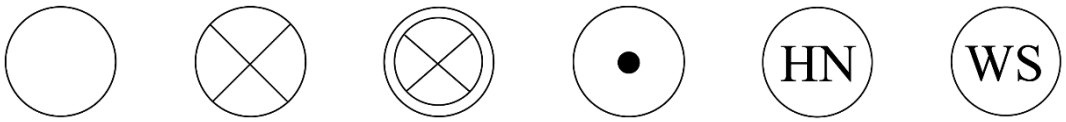
Řešení složitých reálných optimalizačních úloh vyžaduje používání komplikovaných matematických modelů. Pro účely zápisu modelu úlohy pro konkrétní softwarový produkt je vhodné mít k dispozici nástroje, které umožní jednoduché zacházení s modelem a jeho modifikaci v průběhu řešení. Pro řešení určitých typů úloh může být vhodným nástrojem dekompozice (viz např. 2.1.4), ovšem podobné přístupy, používané pro rozmanité typy optimalizačních úloh, jsou obvykle využívané izolovaně případ od případu. V [38] je představen jednotící rámec založený na grafické reprezentaci a objektovém přístupu, který je nazvaný Distribuované optimalizační programy (*Distributed optimization programmes* – DOP). Jak je v pramenu [38] uvedeno, má se jednat o výraznou pomoc zejména ve stádiu modelování (manipulace s matematickými programy) a méně již ve stádiu samotného řešení. Protože se ale i při řešení často používá distribuovaný přístup, dekompozice a paralelní výpočty, je vhodné uvažovat o propojení obou fází a aplikovat DOP i na fázi řešení. Způsob, jak toho lze dosáhnout, je navržen v kapitole 6. Následující text v této kapitole je zpracován dle [38, 43, 44, 45] v rozsahu nutném pro srozumitelnost dalšího textu.

4.1 Základní syntaktické prvky

V [39] je uvedeno, že statické programy mohou být uvažovány jako optimalizační elementy s určitou strukturou

$$\mathcal{O} = (\mathrm{C}, \mathrm{F}, \mathrm{G}) \tag{4.1}$$

kde symboly C, F, G definují interní strukturu optimalizačního prvku. Konkrétně symbol C definuje rozhodovací proměnnou $\mathbf{x}$ a množinu přípustných řešení C, $\mathbf{x} \in C$. Symbol F představuje evaluační pravidlo pro $\mathbf{x}$ (např. $f(\mathbf{x})$). Symbol G specifikuje cíl rozhodnutí $\mathbf{x}$, pokud existuje (např. $? \in \operatorname{argmin}\{\dots\}$).



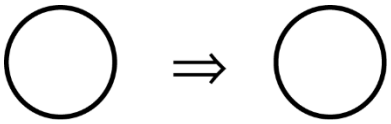
a) *b)* *c)* *d)* *e)* *f)*

Obr. 4.1 – *Příklady symbolů optimalizačních elementů*
a) deterministický program, b) náhodný element, c) základní matematický program stochastické úlohy, d) deterministický ekvivalent stochastické úlohy, e) here-and-now úloha, f) wait-and-see úloha

Pro vizualizaci struktury optimalizační úlohy a jejích elementů byla vyvinuta jednoduchá grafická reprezentace, kde optimalizační elementy jsou symbolizovány uzlem grafu; použitý grafický symbol odlišuje různé typy optimalizačních elementů (deterministický

program, obecný stochastický program, deterministický ekvivalent stochastického programu, …). Příklady grafické podoby některých typů uzlů jsou na obrázku 4.1. Náhodný element na obr. 4.1b představuje zdroj náhodných hodnot při modelování stochastických úloh; např. náhodný element $\mathcal{R}$ může být specifikován jako $\mathrm{C}: \boldsymbol{\xi} \sim P \in \Pi$ (náhodný vektor $\boldsymbol{\xi}$ s rozdělením pravděpodobnosti P z množiny rozložení pravděpodobnosti Π), F může být definováno např. jako očekávaná hodnota, cíl G v tomto případě nebude specifikován. Symbol na obrázku 4.1c reprezentuje základní matematický program stochastické úlohy. Pro deterministický ekvivalent stochastické úlohy je obecně používán symbol dle obr. 4.1d; tento univerzální symbol je při reálném použití modifikován tak, že namísto tečky je uvnitř uzlu vhodná zkratka vztahující se k typu zvoleného deterministického ekvivalentu (např. here-and-now přístup – obr. 4.1e nebo wait-and-see přístup – obr. 4.1f). Pro větší přehlednost je v [38] rovněž používána konvence, kdy se uvnitř uzlu jako popis používá odkaz na číslo rovnice ve zmíněné publikaci, tato konvence je ovšem obtížně přenositelná do publikací jiných.

S grafy reprezentujícími matematické programy (modely optimalizačních úloh) lze provádět syntaktické transformace (dle [38] *element trace*), kdy jednotlivé elementy mohou být dekomponovány na několik elementů s jistými vazbami (to je ekvivalentem dekompozice úloh matematického programování) nebo mohou být transformovány na element jiný (stejného nebo různého typu). Pro tuto transformaci je používán operátor $\Longrightarrow$. Pro identifikaci typů uzlů je v dalším textu používáno označení vycházející ze zkratek anglických názvů, tedy DP = deterministic programme, UP = underlying programme (základní matematický program stochastické úlohy), DE = deterministic equivalent.



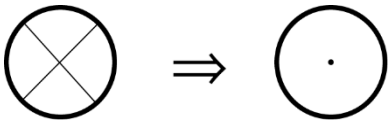
Obr. 4.2: *Transformace DP/DP*

Pro transformaci, kdy deterministický program je transformován na jiný ekvivalentní deterministický program, je použito označení DP/DP. Může se jednat např. o změnu standardního tvaru lineárního programu (nahrazení maximalizace minimalizací, převod podmínek ve tvaru rovnic do tvaru nerovnic, …), přechod k duální úloze apod. Příklad grafického znázornění této transformace je na obrázku 4.2.

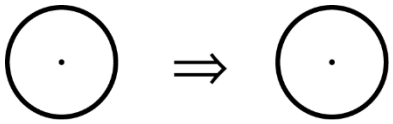
Obr. 4.3: *Transformace DP/UP*

Transformace DP/UP propojují deterministické programy se základními programy stochastických úloh. V tomto kroku je nutné zejména specifikovat, které parametry jsou

náhodné, a určit jejich rozdělení pravděpodobnosti. Transformace je graficky znázorněna na obrázku 4.3. Určení vhodného deterministického ekvivalentu stochastické úlohy a náhrada deterministického ekvivalentu jiným (např. z výpočetních důvodů) je náplní transformací UP/DE a DE/DE (obrázky 4.4 a 4.5).



Obr. 4.4: *Transformace UP/DE*



Obr. 4.5: *Transformace DE/DE*

Pokud je deterministický ekvivalent nahrazen deterministickým programem, potom výsledkem transformace je deterministický program (transformace DE/DP, obr. 4.6). Transformace mohou být zřetězeny do transformačních řetězců, typicky např. DP/UP/DE/DP.

Obr. 4.6: *Transformace DE/DP*

4.2 Vazby mezi prvky DOP

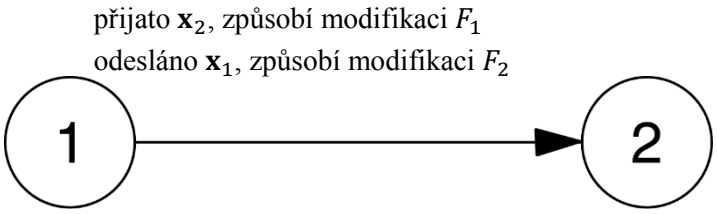
Optimalizační elementy byly v předchozím textu považovány za samostatné programy, kde jedinými souvislostmi mezi elementy byly výše zmíněné transformace. Obecně lze ale uvažovat i jiné vazby mezi jednotlivými programy. Lze si představit existenci dvou programů, kde každý z nich generuje optimální řešení a toto řešení může ovlivňovat druhý program (obr. 4.7), a to buď jeho množinu přípustných řešení, nebo jeho účelovou funkci (tyto dva případy by nemusely být rozlišovány, neboť změna množiny přípustných řešení se může promítnout do účelové funkce formou významné penalizace nepřípustných řešení, jejich odlišné posuzování je ale často vhodné z hlediska modelování). Ukázky prakticky řešených případů takto vzájemně závislých programů jsou v kapitole 5.

Mezi dvěma programy (elementy) reprezentovanými symboly programy $\mathcal{O}_1$, $\mathcal{O}_2$, pokud $\mathcal{O}_1$ reprezentuje nadřízený uzel („master“) a $\mathcal{O}_2$ reprezentuje podřízený uzel („slave“), může docházet k těmto variantám vzájemného vztahu [38]:

1. $\mathcal{O}_1$ a $\mathcal{O}_2$ se nijak neovlivňují, tzn., že v grafu nejsou spojeny žádnou hranou.
2. $\mathcal{O}_1$ modifikuje strukturu $\mathcal{O}_2$ ovlivňováním C_2 a F_2. Současně existuje „zpětná vazba“ modifikací F_1 vyvolaného $\mathcal{O}_2$. V grafu se pro znázornění tohoto vztahu používá jednoduchá šipka směřující od $\mathcal{O}_1$ k $\mathcal{O}_2$. Tato situace je znázorněna na obr. 4.7.

3. $\mathcal{O}_1$ ovlivňuje $\mathcal{O}_2$ stejně jako v předchozí variantě, v opačném směru ale k žádnému ovlivnění nedochází (řešení $\mathcal{O}_2$ nijak neovlivňuje uzel $\mathcal{O}_1$). V grafu se tento vztah znázorňuje pomocí dvojité šipky směřující od $\mathcal{O}_1$ k $\mathcal{O}_2$. Tento vztah je označován jako „silná vazba“.
4. $\mathcal{O}_1$ nijak neovlivňuje $\mathcal{O}_2$, ale musí čekat na výsledek $\mathcal{O}_2$, neboť tento výsledek modifikuje F_1. Tento vztah se v grafu označuje čárkovanou šipkou a nazývá se „slabá vazba“.

Popsané vztahy mohou být i obousměrné (symetrické), pak se značí buď dvěma hranami mezi uzly opatřenými šipkami o opačné orientaci, nebo jednou hranou bez šipky. Zajímavou otázkou také je časová synchronizace elementů, tedy jak jsou vazby mezi elementy uspořádány v čase. V případě jednosměrné vazby je výsledné rozhodnutí učiněno s ohledem na známé informace „master“ elementů a předjímá řešení uzlů podřízených, u obousměrné vazby je situace podstatně složitější a nelze ji pro libovolný případ obecně popsat.

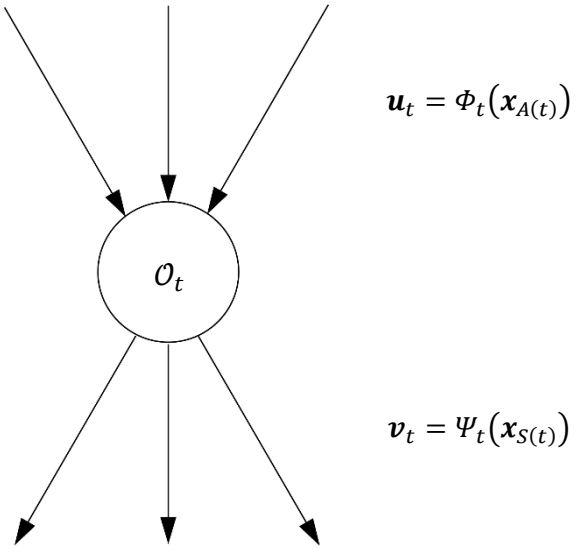


Obr. 4.7: *Vazba mezi elementy [38]*

V praktické realizaci může docházet k tomu, že data zaslaná jedním uzlem nemusí být obecně přímo využitelná jiným uzlem, neboť v cílovém uzlu ve svojí původní podobě nemají smysl. Jednotlivé uzly totiž mohly vzniknout dekompozicí původní úlohy (původního uzlu) pomocí syntaktických transformací. Řešení předané jinému uzlu tak někdy musí být upraveno pomocí odpovídající transformace (např. kdyby ve výše uvedené situaci 3 by optimální řešení nalezené uzlem 1 $\mathbf{x}_1^{opt}$ nemohlo být přímo dosazeno za C_2). Z toho důvodu jsou formálně zavedeny ještě tzv. „transformační elementy“. Tyto transformační elementy jsou potom použity k propojení interagujících uzlů. Všechny tři uvedené typy elementů, tedy optimalizační, náhodné a transformační, jsou souhrnně označovány jako DOP elementy. Při konkrétní softwarové realizaci jsou často transformační funkce přímo součástí příslušných uzlů, pro dodržení obecnosti grafického popisu je však nutné transformační uzly zavést.

Transformační elementy používají transformační funkce Φ a Ψ. Pro napojení prvku $\mathcal{O}_t$ na ostatní prvky je nutné specifikovat hodnoty $\mathbf{u}_t$ (vstup do prvku $\mathcal{O}_t$) a $\mathbf{v}_t$ (výstup z prvku $\mathcal{O}_t$). Je-li $A(t)$ množina indexů DOP elementů, které jsou bezprostředně předcházející elementu $\mathcal{O}_t$, pak $\mathbf{u}_t = \Phi_t(\mathbf{x}_{A(t)}) = \Phi_t((\mathbf{x}_a)_{a\in A(t)})$, kde $\mathbf{x}_{A(t)}$ jsou parametry nebo proměnnými elementů bezprostředně předcházejících uzlu $\mathcal{O}_t$ a Φ_t je odpovídající

vektorová nebo skalární funkce. Vstup do elementu $\mathcal{O}_t$ vznikne transformací výstupů elementů bezprostředně předcházejících. Obdobně jsou pro elementy bezprostředně následující elementu $\mathcal{O}_t$ použity indexy z množiny $S(t)$ a pak pro výstupy elementu $\mathcal{O}_t$ platí $\mathbf{v}_t = \Psi_t(\mathbf{x}_{S(t)}) = \Psi_t((\mathbf{x}_s)_{s\in S(t)})$. Výstup z uzlu $\mathcal{O}_t$ je tedy předán do bezprostředně následujících uzlů po provedení transformace popsané funkcí Ψ_t. Situace je znázorněna na obrázku 4.8.



Obr. 4.8: *Interakce DOP elementu [38]*

S využitím uvedených úvah a definičního vztahu (4.1) může být optimalizační element, který je prvkem propojené struktury elementů, jejichž indexy jsou označeny $t \in \tau$, popsán vztahem

$$\mathcal{O}_t = (\mathrm{C}_t, \mathrm{F}_t, \mathrm{G}_t, \Phi_t, \Psi_t). \tag{4.2}$$

Pro dané τ se množina uzlů $\{\mathcal{O}_t\}_{t\epsilon\tau}$ nazývá **distribuovaný optimalizační program** (DOP). Jako *uzavřený DOP* (closed DOP, CDOP) se označuje program, kde $\forall t \in \tau$ platí, že parametry $\mathbf{u}_t$ a $\mathbf{v}_t$ jsou specifikovány odpovídajícími funkcemi Φ_t a Ψ_t. V případě, že některé parametry $\mathbf{u}_t$ a $\mathbf{v}_t$ nejsou plně specifikovány odpovídajícími funkcemi Φ_t a Ψ_t, používá se termín *otevřený DOP* (open DOP, ODOP). ODOP je chápán jako reprezentant jistých neurčitostí. Tyto neurčitosti mohou být modelovány vložením náhodných elementů. Tím lze ODOP převést na CDOP. Při vkládání náhodných elementů je nutno při modelování vycházet z typu úlohy, podrobnosti včetně odpovídajících transformací jsou uvedeny v [38].

5 VYBRANÉ DISTRIBUOVANÉ OPTIMALIZAČNÍ PROBLÉMY

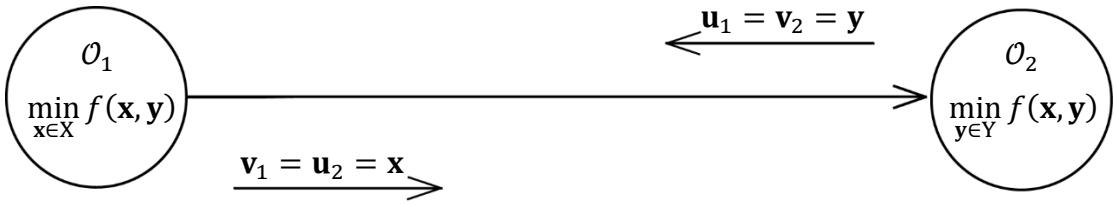
V této kapitole je popsáno několik reálně řešených případů DOP, na kterých byla ověřována použitelnost principů DOP a jejichž realizace potom sloužila jako východisko pro návrh OTP popsaný v kapitole 6. Ve všech případech je v jednom uzlu použit genetický algoritmus, v uzlu druhém je pak použit buď rovněž genetický algoritmus nebo síťová úloha řešená pomocí programu GAMS [66].

5.1 Vhnízděné[9] genetické algoritmy

Popis této a řešení této úlohy vychází z [53]. Mějme následující obecnou minmax úlohu:

$$? \in \min_{\mathbf{x}\in X} \max_{\mathbf{y}\in Y} \{f(\mathbf{x},\mathbf{y})\}, \tag{5.1}$$

kde f je kriteriální funkce, **x** a **y** jsou rozhodovací proměnné (vektory) z prostoru X resp. Y. V tomto případě uvažujeme dva optimalizační elementy. Vnitřní element $\mathcal{O}_2$ představuje maximalizační úlohu, která by měla být řešena pro fixní hodnoty **x** nastavené Ψ, a vnější minimalizační úloha $\mathcal{O}_1$, která by měla být řešena pro získané optimální hodnoty **y**, které jsou předány z prvku $\mathcal{O}_2$ řešícího maximalizační úlohu transformací Φ. Dle klasifikace vazeb mezi elementy uvedené v kapitole 4.2 se zde jedná o situaci 2, takže v grafickém znázornění budou uzly propojeny plnou čarou se šipkou směřující od $\mathcal{O}_1$ k $\mathcal{O}_2$, situace je zachycena na obr. 5.1. Protože je účelová funkce pro oba uzly totožná a výpočet v uzlu se liší pouze tím, podle které proměnné se optimalizuje a která proměnná je považována za fixní parametr, je zřejmé, že transformace ve smyslu kapitoly 4.2 nebude nutná, a tedy výsledek $\mathcal{O}_2$ bude možné přímo použít v $\mathcal{O}_1$ a naopak. S použitím symboliky dle 4.2 tedy platí $\mathbf{u_1} = \mathbf{v}_2 = \mathbf{y}$ a analogicky $\mathbf{u}_2 = \mathbf{v}_1 = \mathbf{x}$.



Obr. 5.1: *DOP model úlohy*

[9] Odpovídající anglický termín je „nested". Toto slovo se v oblasti optimalizace do češtiny obvykle překládá slovem „zahnízděný". V oblasti programování je obvyklejší slovo „vhnízděný" (např. vhnízděné komentáře v jazyce C, vhnízděné cykly) nebo dokonce „hnízděný" (hnízděné tabulky v SQL). Lépe znějící a významově výstižnější slovo „vnořený" je naproti tomu používáno méně (např. vnořené funkce) než uvedená v češtině jinak téměř nepoužívaná slova, která jsou ale přesnějším překladem. Protože autor se cítí být především programátorem, je v tomto textu používáno programátorsky obvyklé slovo „vhnízděný".

K řešení byly využity dva *vhnízděné* GA, základní myšlenka byla inspirována prací [3]. Byla sestavena tabulka se třemi sloupci a n řádky. První a druhý sloupec obsahují jako prvky vektory **x** a **y**, třetí sloupec obsahuje odpovídající hodnotu kriteriální funkce. V této pomocné datové struktuře jsou tedy po řádcích uloženy $(\mathbf{x}_i, \mathbf{y}_i, f(\mathbf{x}_i, \mathbf{y}_i))$ reprezentující nejlepší momentálně známá řešení. Běh dvou spolupracujících genetických algoritmů lze popsat následovně:

1. Inicializace
   a. Vygenerování iniciálních hodnot $\mathbf{x}_i^0, i \in \{1, \ldots n\}$ (náhodně). Dolní index reprezentuje řádek matice, horní index číslo iterace.
   b. Pomocí GA určeného ve struktuře DOP k výpočtu **y** (vnitřní GA) se vypočítá $\mathbf{y}_i^1, i \in \{1, \ldots n\}$ tak, aby platilo:

$$f(\mathbf{x}_i^0, \mathbf{y}_i^1) = \max_{\mathbf{y} \in Y}(f(\mathbf{x}_i^0, \mathbf{y})). \tag{5.2}$$

   c. Setřídění řádků pomocné tabulky podle třetího sloupce vzestupně. První řádek tabulky po setřídění představuje nejlepší doposud známé řešení, toto dílčí řešení je uloženo do další pomocné datové struktury.
   d. Položí se $k = 1$ (k je číslo iterace).
2. Hlavní cyklus
   a. Pomocí GA určeného ve struktuře DOP k výpočtu **x** se vypočítá $\mathbf{x}_i^k, i \in \{1, \ldots n\}$ tak, aby platilo:

$$f(\mathbf{x}_i^k, \mathbf{y}_i^k) = \min_{\mathbf{x} \in X}(f(\mathbf{x}, \mathbf{y}_i^k)). \tag{5.3}$$

   b. Pomocí GA určeného ve struktuře DOP k výpočtu **y** (vnitřní GA) se vypočítá $\mathbf{y}_i^{k+1}, i \in \{1, \ldots n\}$ tak, aby platilo:

$$f(\mathbf{x}_i^k, \mathbf{y}_i^{k+1}) = \max_{\mathbf{y} \in Y}(f(\mathbf{x}_i^k, \mathbf{y})) \tag{5.4}$$

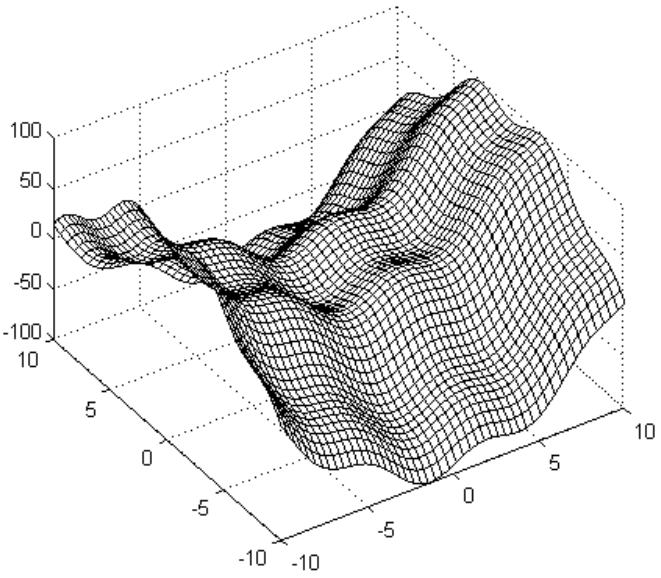
   c. Setřídění pomocné tabulky jako v kroku 1.c. Je-li získané nejlepší řešení lepší než doposud známé nejlepší řešení, je toto nové dosud známé nejlepší řešení uloženo do pomocné datové struktury.
   d. Není-li dosažen maximální počet iterací, inkrementuje se číslo iterace k a provede se návrat ke kroku 2.a, jinak ukončení algoritmu.

Popsaný algoritmus byl v [53] testován na dvou funkcích, pro možnost grafického znázornění a lepší představu o reálnosti obdrženého řešení se v obou případech jednalo o funkce pouze dvou proměnných. První testovací funkce byla popsána vztahem:

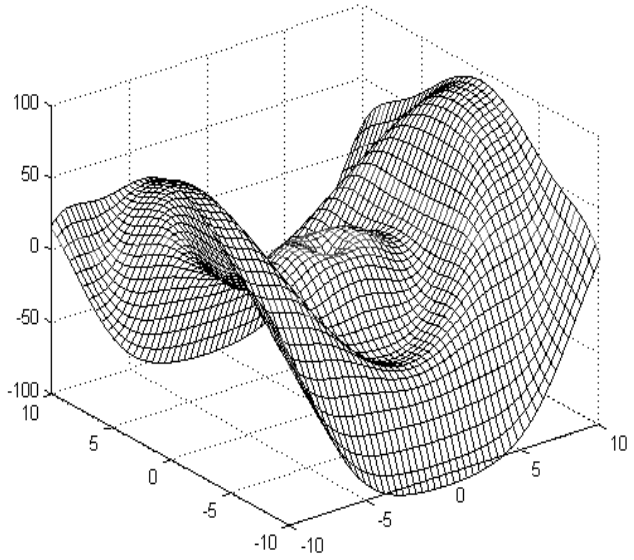
$$f_1(x, y) = x^2 - y^2 + 10\sin x - 10\cos y\,, \tag{5.5}$$

graf této funkce je na obrázku 5.2. Jako řešení byly získány hodnoty:

$$f_1(x^*, y^*) = \min_{x \in X} \max_{y \in Y} \{f_1(x, y)\} = f_1(-2{,}6,\ -1.4) = 6{,}08.$$



Obr. 5.2: *Graf testovací funkce (5.5)*



Obr 5.3: *Graf testovací funkce (5.6)*

Druhá testovací funkce byla podobná, ovšem s ohledem na průběh měla být úloha obtížněji řešitelná. Jednalo se o funkci

$$f_2(x, y) = x^2 - y^2 + 15 sin\left(\sqrt{x^2 + y^2}\right), \tag{5.6}$$

její graf je na obrázku 5.3. Bylo získáno řešení:

$$f_2(x^*, y^*) = \min_{x \in X} \max_{y \in Y} \{f_2(x, y)\} = f_2(4{,}13,\ 0.00) = 4{,}53.$$

V obou případech byla použita populace o velikosti 30 jedinců, počet iterací byl pro každý genetický algoritmus omezen na 30, hlavní cyklus byl omezen na 10 iterací. Podrobnější informace k tomuto problému lze nalézt v [53].

5.2 Generování a analýza scénářů pomocí heuristických algoritmů

Následující problém a způsob jeho řešení vychází z [40], postup je potom zpřesněn a detailně rozpracován v [50], [60] a [51]. Připomeňme, že problém matematického programování je obvykle symbolicky popisován jako

$$? \in \operatorname{argmin}\{f(\mathbf{x}) \mid \mathbf{x} \in C\}. \tag{5.7}$$

Pokud některé z původně konstantních parametrů funkce f budeme považovat za náhodné veličiny, pak lze (5.7) zapsat ve tvaru

$$? \in \operatorname{argmin}\{f(\mathbf{x},\boldsymbol{\xi}) \mid \mathbf{x} \in C(\boldsymbol{\xi})\}. \tag{5.8}$$

Zde $\boldsymbol{\xi}$ představuje náhodný vektor definovaný na pravděpodobnostním prostoru $(\Xi, \Sigma, \mathrm{P})$ a $f: \mathbb{R}^n \times \Xi \to \mathbb{R}$ je funkce $\mathbf{x}$ měřitelná pro každé rozhodnutí $\mathbf{x} \in \mathbb{R}^n$, které náleží do množiny přípustných řešení C. Aby bylo možné korektně vyřešit optimalizační problém, musí být dále provedena deterministická reformulace. Při rozhodování lze postupovat tzv. strategií „here-and-now“ (HN) nebo „wait-and-see“ (WS), viz 2.2. Pokud uvažujeme strategii HN, může reformulace vypadat např. takto:

$$? \in \arg\min_{\mathbf{x}} \{E_{\boldsymbol{\xi}}(f(\mathbf{x},\boldsymbol{\xi})) \mid \mathbf{x} \in C(\boldsymbol{\xi}) \text{ a.s.}\}, \tag{5.9}$$

kde E popisuje střední hodnotu účelové funkce a zkratka a.s. (almost surely) znamená, že uvedený předpoklad je splněn skoro jistě. Je třeba podotknout, že existují i jiné přístupy ke způsobu modelování náhodných parametrů (viz např. [22]).

V případě, že náhodný vektor má spojité rozdělení pravděpodobnosti, je řešení stochastické úlohy (5.9) obtížné. Zde by bylo možné použít aproximační techniky založené na diskretizaci [22]. Uvažujeme-li diskrétní rozdělení pravděpodobnosti, bude výpočetní náročnost ovlivněna zejména počtem možných scénářů (počtem možných realizací vektoru $\boldsymbol{\xi}$). Pokud označíme $p_s = P(\boldsymbol{\xi} = \boldsymbol{\xi}^s)$, pak střední hodnotu lze explicitně vypočítat a tato SB–reformulace (scenario-based, založená na scénářích, viz dále) vede na rozsáhlou nelineární úlohu:

$$? \in \arg\min_{\mathbf{x}} \left\{ \sum_{\xi^s \in \Xi} p_s f(\mathbf{x},\boldsymbol{\xi}^s) \mid \mathbf{x} \in \bigcap_{\xi^s \in \Xi} C(\boldsymbol{\xi}^s) \right\} \tag{5.10}$$

Hlavní problém zde je výběr vhodných realizací $\boldsymbol{\xi}^s$ zvaných scénáře, zejména pokud je k dispozici pouze neúplná informace rozdělení pravděpodobnosti. Techniky vztahující se ke scénářům lze nalézt např. v [7], zajímavý přístup je v [15].

Dále uvedený model (5.11) se vztahuje k řízení tavby (podrobnosti lze nalézt v [38, 39]), zde je použit jako obecný příklad k demonstrování aplikace DOP, a proto je popis zestručněn. V [50] byl tento příklad použit proto, že umožňoval pracovat s reálnými daty a tudíž i posoudit reálnost výsledků. Dvoustavový SB stochastický lineární program, představující zvláštní případ (5.10), lze popsat

$$
\begin{aligned}
&? \in \underset{\mathbf{x}}{\operatorname{argmin}} \left\{ \mathbf{c}^{\mathrm{T}}\mathbf{x} + \sum_{s=1}^{S} p_s Q\left(\mathbf{x}, \boldsymbol{\xi}^s\right) \middle| \mathbf{x} \geq 0 \right\}, \\
&Q\left(\mathbf{x}, \boldsymbol{\xi}^s\right) = \min_{\mathbf{y}^s} \left\{ \mathbf{q}^{\mathrm{T}}\mathbf{y}^s \;\middle|\; \mathbf{T}_1^s \mathbf{A}^1 \mathbf{x} + \mathbf{A}^2 \mathbf{y}^s \geq \mathbf{l}_2, \mathbf{T}_1^s \mathbf{A}^1 \mathbf{x} + \mathbf{A}^2 \mathbf{y}^s \leq \mathbf{u}_2, \right. \\
&\qquad\qquad \left. \mathbf{x} \geq 0, \mathbf{y}^s \geq 0, s = 1, \ldots, S \right\},
\end{aligned}
\tag{5.11}
$$

kde $\mathbf{x}$ udává množství vsázky n_1 surovin a $\mathbf{y}^s$ množství n_2 legovacích materiálů. Symboly $\mathbf{l}_2$ a $\mathbf{u}_2$ popisují minimální a maximální množství m sledovaných prvků po ukončení tavby. Matice $\mathbf{A}^1$ obsahuje prvky a_{ij}^1, které udávají obsah i-tého prvku v j-té surovině. Podobně prvky matice $\mathbf{A}^2$ obsahují informace o obsahu prvků v legovacích materiálech. Konečná množina $\Xi = \{\boldsymbol{\xi}^1, \ldots, \boldsymbol{\xi}^S\}$ obsahuje možné scénáře, pro rozdělení pravděpodobnosti platí $p_s = P(\boldsymbol{\xi} = \boldsymbol{\xi}^s) = 1/s\,, s = 1, \ldots, S$. Propal prvků vsázky je považován za náhodný a je popsán diagonální maticí $\mathbf{T}_1^s$, u legovacích materiálů se propal neuvažuje. Vektory $\mathbf{c}$ a $\mathbf{q}$ obsahují ceny za tunu příslušné suroviny resp. legovací přísady. Detailní popis tohoto modelu včetně vstupních hodnot je obsažen v [38, 39].

Cílem je získat optimální řešení $\mathbf{x}_{\min}^{\mathrm{HN}}$ za použití rozhodovací strategie HN a odpovídající hodnotu účelové funkce $z_{\min}^{\mathrm{HN}}$. Ke zmenšení úlohy lze použít techniku náhodného vzorkování. Označme náhodný výběr/vzorek ze scénářů $\boldsymbol{\xi}$ jako $\boldsymbol{\xi}_{[.]} = (\boldsymbol{\xi}_{[1]}, \ldots, \boldsymbol{\xi}_{[v]})^{\mathrm{T}}$. Máme nyní náhodné proměnné $\boldsymbol{\xi}_{[s]}$, které mají identickou distribuční funkci jako $\boldsymbol{\xi}$. Realizace tohoto náhodného výběru je označena jako $\boldsymbol{\xi}_{[.]}^s = \left(\boldsymbol{\xi}_{[1]}^s, \ldots, \boldsymbol{\xi}_{[v]}^s\right)^{\mathrm{T}}$, zjednodušeně lze použít zápis $\boldsymbol{\xi}_{[.]}^s = (\boldsymbol{\xi}^{s1}, \ldots, \boldsymbol{\xi}^{sv})^{\mathrm{T}}$. Pro výpočetní účely lze jednoduše nahradit výpočet střední hodnoty $E_\xi\{f(\mathbf{x}, \boldsymbol{\xi})\}$ (resp. $E_\xi\{Q(\mathbf{x}, \boldsymbol{\xi})\}$ pro dvoustupňové úlohy) výpočtem průměrné hodnoty pro náhodně vybrané scénáře $\frac{1}{v}\sum_{s=1}^{v} f(\mathbf{x}, \boldsymbol{\xi}^s)$ resp. $\frac{1}{v}\sum_{s=1}^{v} Q(\mathbf{x}, \boldsymbol{\xi}^s)$ a dostaneme:

$$
z_{min}^{v} = \frac{1}{v}\sum_{s=1}^{v} f\left(\mathbf{x}_{min}^{v}, \boldsymbol{\xi}^s\right) = \min_{\mathbf{x}} \left\{ \frac{1}{v}\sum_{s=1}^{v} f\left(\mathbf{x}, \boldsymbol{\xi}^s\right) \middle| \mathbf{x} \in C \right\} \tag{5.12}
$$

Náhodně generovaná pozorování $\boldsymbol{\xi}$ mohou sloužit k výpočtu odhadů hodnot účelové funkce. Scénáře jsou vybírány náhodně. Následně jsou scénáře $\boldsymbol{\xi}^s$ použity pro vytvoření stromu scénářů, a takto redukovaný program lze řešit namísto původní úlohy [22, 7]. Přecenění tohoto výběru „na slepo“ může samozřejmě vést k nesprávným zavádějícím

výsledkům. Je proto vhodné opakovat náhodný výběr scénářů metodou Monte Carlo. Lze tak získat náhled na stabilitu výsledků a citlivost na výběr scénářů.

V práci [25] je dokázáno, že platí nerovnosti:

$$E_\xi\left\{\varsigma_{min}^{\nu}\right\} \le E_\xi\left\{\varsigma_{min}^{\nu+1}\right\} \le z_{min}^{\mathrm{HN}} \le z = E_\xi\left\{f(\mathbf{x},\boldsymbol{\xi})\right\}. \tag{5.13}$$

Předpokládá se tedy, že náhodný vzorek $\boldsymbol{\xi}$ označený jako $\boldsymbol{\xi}_{[.]} = \left(\boldsymbol{\xi}_{[1]}, \ldots, \boldsymbol{\xi}_{[v_u]}\right)^T$ je dostupný, a ς_{min}^{ν} označuje náhodnou optimální hodnotu účelové funkce závislou na náhodném výběru množiny scénářů o velikosti ν. Existuje v_l náhodných výběrů, každý náhodný výběr má ν prvků, lze tedy zapsat $\forall i = 1, \ldots, v_l$: $\boldsymbol{\xi}_{[i.]} = \left(\boldsymbol{\xi}_{[i1]}, \ldots, \boldsymbol{\xi}_{[iv]}\right)^{\mathrm{T}}$. S použitím (5.13) a centrální limitní věty jsou v [25] odvozeny následující meze:

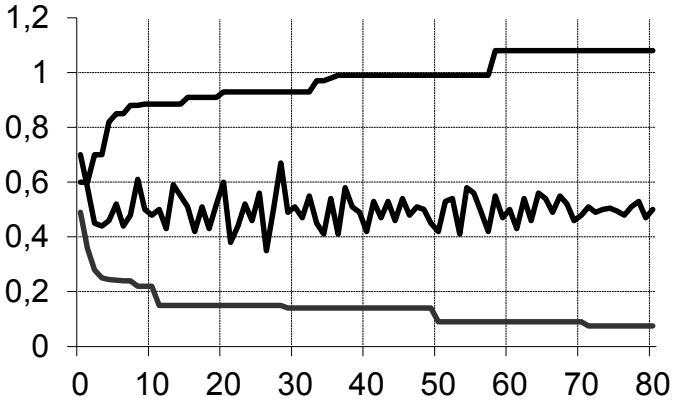
$$\begin{aligned} P\Bigg(\frac{1}{v_l}\sum_{i=1}^{v_l} \min_{\mathbf{x}(\boldsymbol{\xi}_{[i.]})} \left\{ \frac{1}{\nu}\sum_{s=1}^{\nu} f\left(\mathbf{x}\left(\boldsymbol{\xi}_{[i.]}\right), \boldsymbol{\xi}_{[is]}\right) \middle| \mathbf{x}\left(\boldsymbol{\xi}_{[i.]}\right) \in C \text{ a.s.} \right\} - \frac{t_{1-\alpha/2}\, s_l(v_l)}{\sqrt{v_l}} \le \\ \le E_\xi\left\{\varsigma_{min}^{\nu}\right\} \le z_{min}^{\mathrm{HN}} \le E_{\boldsymbol{\xi}}\left\{f(\mathbf{x},\boldsymbol{\xi})\right\} \le \\ \le \frac{1}{v_u}\sum_{s=1}^{v_u} f\left(\mathbf{x},\boldsymbol{\xi}_{[s]}\right) + \frac{t_{1-\alpha/2}\, s_u(v_u)}{\sqrt{v_u}} \Bigg) \approx 1-\alpha \end{aligned} \tag{5.14}$$

Symbol $t_{1-\alpha/2}$ značí $1-\alpha/2$ kvantil binomického (N(0;1)) rozdělení. Symboly $s_l(v_l)$ a $s_u(v_u)$ značí odhady směrodatných odchylek $\sqrt{\mathrm{var}\,\boldsymbol{\varsigma}_{\min}^{\nu}}$ a $\sqrt{\mathrm{var}\, f(\mathbf{x},\boldsymbol{\xi})}$.

Pro testovací aplikaci řízení tavby byly provedeny výpočty pro různé sady vybraných scénářů, získané optimální hodnoty účelové funkce jsou na obrázku 5.4 (střední křivka). Opticky se jeví, že výsledky jsou pro různé scénáře poměrně stabilní. Je ale otázka, zda se skutečně jedná o poměrně nízkou citlivost na náhodné vlivy, nebo zda scénáře vedoucí ke „špatným“ výsledkům nebyly ve výběru obsaženy. Proto byla snaha získat sadu extrémních scénářů a zjistit, jaké nejhorší (a nejlepší) výsledky se mohou objevit; jinak řečeno získat meze, ve kterých se hodnoty účelové funkce mohou v závislosti na náhodné veličině pohybovat. Získané meze jsou na obr. 5.4 (horní a dolní křivka). Hodnoty těchto mezí lze určit z následujících vzorců:

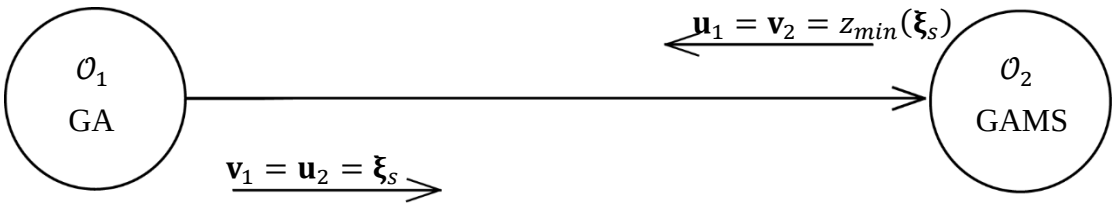
$$\min_{S\subset\Xi;|S|=n} \quad \min\left\{ \frac{1}{S}\sum_{\boldsymbol{\xi}^s\in S} f\left(\mathbf{x},\boldsymbol{\xi}^s\right) \middle| \mathbf{x} \in \bigcap_s C\left(\boldsymbol{\xi}^s\right) \right\} \tag{5.15}$$

$$\max_{S\subset\Xi;|S|=n} \quad \min\left\{ \frac{1}{S}\sum_{\boldsymbol{\xi}^s\in S} f\left(\mathbf{x},\boldsymbol{\xi}^s\right) \middle| \mathbf{x} \in \bigcap_s C\left(\boldsymbol{\xi}^s\right) \right\} \tag{5.16}$$



Obr. 5.4: *Získané optimální řešení a horní a dolní mez v závislosti na číslu iterace*

Pro řešení jsou použity dva spolupracující softwarové produkty – GAMS a genetický algoritmus. Množina vybraných scénářů je reprezentována chromozomem genetického algoritmu, hodnotou účelové funkce pro GA je $z_{\min}^{\mathrm{HN}}$, tedy pro daný příklad celkové náklady na tavbu daného složení, nastane-li příslušný scénář. K výpočtu ceny pro daný scénář je použit GAMS, který je z GA spouštěn jako externí úloha. K obousměrnému přenosu dat jsou použity diskové soubory. GA současně registruje mezní hodnoty. Zvolené řešení v žádném případě není optimální po stránce programové realizace, klady distribuovaného řešení umožňující řešení i velmi rozsáhlých úloh jsou do značné míry degradovány zpomalením výpočtu vlivem nutnosti spustit externí program vždy, když GA vyžaduje ocenění člena populace. Přesto toto řešení umožnilo ověření reálnosti myšlenky realizovat řešení podobných úloh pomocí jejich dekompozice dle principů DOP. Stejně jako v předchozím případě se jedná o dva spolupracující DOP elementy s obousměrným přenosem dat, nadřízeným uzlem $\mathcal{O}_1$ byl genetický algoritmus, podřízeným uzlem $\mathcal{O}_2$ byl GAMS. Transformaci dat prováděl přímo genetický algoritmus, je nutno podotknout, že se nejednalo o transformaci hodnot ale pouze formátů tak, aby byl vždy dodržen formát vyžadovaný programem GAMS. Vstupními daty pro GAMS byl vektor $\boldsymbol{\xi}_s$ vygenerovaný genetickým algoritmem, GAMS jako svůj výstup poskytoval optimální řešení pro tento scénář $z_{min}(\boldsymbol{\xi}_s)$. Graf popisující úlohu z hlediska DOP je na obr. 5.5.



Obr. 5.5: *DOP model úlohy*

Algoritmus použitý k řešení lze stručně popsat následovně:

1. Inicializace. Start hlavního programu obsahujícího GA. Kromě režijních dat typu jména souborů, cesta k externímu optimalizačnímu programu (GAMS) apod. je

nutné inicializovat i data nutná pro řešení daného problému, jako je počet scénářů, jejich dimenze a distribuční funkce pro náhodné veličiny – prvky matice $\mathbf{T}_1^s$. GA náhodně vytvoří iniciální populaci.

2. Proběhne jedna iterace GA, k ocenění jednotlivých členů populace je použit GAMS spouštěný jako externí úloha. Pomocí genetických operátorů je vytvořena nová populace – nová množina scénářů.
3. Není-li splněna podmínka ukončení, pokračuje se krokem 2, jinak konec.

5.3 Návrhy sítí, dopravní problém

Tato podkapitola včetně testovací úlohy a postupu řešení vychází z [54], jiné dopravní úlohy rovněž řešenou pomocí DOP přístupu lze nalézt např. v [13, 18, 19]. Předpokládejme problém návrhu dopravní sítě [10]. Tato síť propojuje producenty a spotřebitele. Následující model je autory označován jako model s přidáváním hran (*adding edges*, AE).

V reálném světě jsou často předem neznámé informace o aktuálních požadavcích spotřebitelů. Tato situace může být modelována metodami stochastického programování s rozhodovací strategií HN (viz 2.2 a 5.2), tvorba modelu vychází z [56], s ohledem na techniku řešení dále popsaný model ideově navazuje na [17].

Účelová funkce modelu dopravní úlohy [10, 5, 16], u které je cílem maximalizovat zisk, může mít tvar:

$$\max\left(\sum_{i_1}\left(\sum_{e} A_{i_1,e}\, x_e \right) g_{i_1} - \sum_{e} c_e x_e - \sum_{e_n} d_{e_n} \delta_{e_n} \right), \tag{5.17}$$

kde $x_e \geq 0$ je množství produktu přepraveného hranou e a $\delta_{e_n} \in \{0,1\}$ je návrhová proměnná, přičemž hodnota 1 značí, že bude přidána nová hrana e_n. $A_{i,e}$ jsou prvky incidenční matice, přičemž hodnota 1 znamená, že hrana e vede do uzlu i, hodnota –1 znamená, že hrana e vychází z uzlu i, nulová hodnota znamená, že hrana e nesouvisí s uzlem i. Symbol g_{i_1} značí jednotkovou cenu pro zákazníka i_1, c_e je jednotková přepravní cena hrany e, d_{e_n} je cena za přidání hrany e_n. Výraz $\sum_{i_1}\left(\sum_e A_{i_1,e} x_e\right) g_{i_1}$ představuje příjem od všech zákazníků. Zbývající část účelové funkce $\sum_e c_e x_e + \sum_{e_n} d_{e_n} \delta_{e_n}$ vyjadřuje cenu za přepravu a náklady na výstavbu nových hran.

Význam indexů je následující: E je množina hran, takže $e \in E$, $E_n \subset E$ je množina hran, které lze přidávat, takže $e_n \in E_n$, množina I_1 je množina zákazníků (odběratelů), takže $i_1 \in I_1$. V modelu jsou zavedeny další indexy, které budou použity v následujících vztazích: I_2 je množina producentů ($i_2 \in I_2$) a I_3 je množina tranzitních uzlů ($i_3 \in I_3$), tedy uzlů, které nejsou zdrojem ani cílem nějakého toku; tato volba symbolického značení indexů je motivována zvýšením čitelnosti zápisu modelu. Množina I je potom množinou všech uzlů v logistické síti, platí tedy $I = I_1 \cup I_2 \cup I_3$.

Označíme b_i jako poptávku v uzlu i. Model je koncipovaný jako vyvážený, tzn. součet poptávky, součet produkce a součet množství přepravovaného zboží jsou si rovny. To je vyjádřeno podmínkou $\sum_e A_{i,e}x_e = b_i, \forall i \in I$ v modelu (5.18). Model dopravní úlohy s přidáváním hran (Transportation model with AE) lze potom formulovat následovně:

$$\begin{aligned}
&\max\left(\sum_{i_1}\left(\sum_e A_{i_1,e}\, x_e\right)g_{i_1} - \sum_e c_e x_e - \sum_{e_n} d_{e_n}\delta_{e_n}\right) \\
&\sum_e A_{i,e}x_e = b_i, && \forall i \in I, \\
&x_{e_n} \le \delta_{e_n}\left(\sum_{i_2} - b_{i_2}\right), && \forall e_n \in E_n, \qquad (5.18)\\
&x_e \ge 0, && \forall e \in E, \\
&\delta_{e_n} \in \{0,1\}, && \forall e_n \in E_n.
\end{aligned}$$

Dále je zavedeno symbolické vyjádření pro neurčitost poptávky $b_{i_1,s}, \forall i_1 \in I_1, s \in S$. Tato neurčitost bude modelována náhodnou veličinou s diskrétním rozdělením pravděpodobnosti a s konečným počtem realizací zvanými scénáře, index $s \in S$ označuje scénář. Při zahrnutí neurčitosti do modelu (5.18) bude používán přístup HN (*here-and-now*).

Do účelové funkce bude zavedena penalizace za neuspokojenou poptávku, která bude specifikována pomocí nově zavedených rozhodovacích proměnných $y^+_{i_1,s} \ge 0$ (pro neuspokojenou poptávku, pokud poptávka v uzlu i_1 náhodně vzrostla) a $y^-_{i_1,s} \ge 0$ (množství zboží, které bylo „přivezeno navíc“, když se poptávka v uzlu i_1 náhodně snížila). Průměrné navýšení ceny oproti původní účelové funkci modelu (5.18) je popsáno výrazem $\sum_s p_s\left(\sum_{i_1}\left(q^-_{i_1}y^-_{i_1,s} + q^+_{i_1}y^+_{i_1,s}\right)\right)$. Symbol $q^+_{i_1}$ udává jednotkovou penalizaci za nedodané zboží do uzlu i_1, symbol $q^-_{i_1}$ představuje jednotkovou penalizaci za přepravu nadbytečného zboží do uzlu i_1 a p_s je pravděpodobnost, že nastane scénář s, $0 \le p_s \le 1, \forall s \in S$, platí $\sum_{s\in S} p_s = 1$. S použitím scénáři modelovaných neurčitostí se změní první omezující podmínka modelu (5.18) na tvar:

$$\sum_e A_{i_1,e}x_e + y^+_{i_1,s} - y^-_{i_1,s} = b_{i_1,s}, \quad \forall i_1 \in I_1, \forall s \in S. \qquad (5.19)$$

Tato modifikovaná podmínka (5.19) vyjadřuje, že součet veškerého přepraveného zboží plus neuspokojená poptávka mínus nadbytečně přepravené zboží musí být rovno celkové poptávce. Protože podmínka (5.19) nyní vyjadřuje situaci pouze pro cílové uzly (tzn. zákazníky, odběratele), je nutné definovat omezující podmínky i pro zdrojové a tranzitní uzly:

$$\sum_e A_{i_2,e}x_e = b_{i_2}, \forall i_2 \in I_2, \qquad (5.20)$$

$$\sum_e A_{i_3,e}x_e = b_{i_3}, \forall i_3 \in I_3. \qquad (5.21)$$

S použitím výše uvedeného pak lze stochastický model dopravní úlohy s přidáváním hran zapsat ve tvaru:

$$
\begin{aligned}
\max \sum_{i_1}\left(\sum_{e} A_{i_1,e}x_e\right)g_{i_1} - \sum_{e} c_e x_e - \sum_{e_n} d_{e_n}\delta_{e_n} - \sum_{s} p_s\left(\sum_{i_1}\left(q^-_{i_1}y^-_{i_1,s} + q^+_{i_1}y^+_{i_1,s}\right)\right) & \\
\sum_{e} A_{i_1,e}x_e = b_{i_1,s} + y^-_{i_1,s} - y^+_{i_1,s}, & \quad \forall i_1 \in I_1, \forall s \in S, \\
\sum_{e} A_{i_2,e}x_e = b_{i_2}, & \quad \forall i_2 \in I_2, \\
\sum_{e} A_{i_3,e}x_e = b_{i_3}, & \quad \forall i_3 \in I_3, \\
x_{e_n} \le \delta_{e_n}\sum_{i_2}\left(-b_{i_2}\right), & \quad \forall e_n \in E_n, \\
y^+_{i_1,s} \le b_{i_1,s}, & \quad \forall i_1 \in I_1, \forall s \in S, \\
x_e \ge 0, & \quad \forall e \in E, \\
\delta_{e_n} \in \{0,1\}, & \quad \forall e_n \in E_n, \\
y^+_{i_1,s},\, y^-_{i_1,s} \ge 0, & \quad \forall i_1 \in I_1, \forall s \in S.
\end{aligned}
\tag{5.22}
$$

Podmínka $x_{e_n} \le \delta_{e_n}\sum_{i_2}\left(-b_{i_2}\right)$ představuje horní mez pro tok nově přidanou hranou a je formulována tak, že tok žádnou z nově zapnutých hran nemůže být větší, než kolik činí produkce v celé síti (pro „pevné“ hrany je tento požadavek zahrnut v prvních třech omezujících podmínkách modelu (5.22).

Modely (5.18) a (5.22) byly implementovány v programu GAMS, k řešení byl použit solver CPLEX [65]. Tak byly otestovány „malé“ kontrolní příklady, získané výsledky byly akceptovatelné [54]. U řešení rozsáhlých reálných úloh lze očekávat významné prodloužení doby řešení, proto byla opět ověřena možnost dekompozice a distribuované řešení pomocí hybridního algoritmu, a to podobně jako v kapitole 5.2 s využitím kombinace GAMS–GA. GA je využit na vnější úrovni pro generování scénářů. Genetický algoritmus generuje a nastavuje hodnoty proměnných $\delta_{e_n}, \forall e_n \in E_n$, tedy pro všechny „zapínatelné“ hrany se v GA určuje, zda hrana bude zapnutá (hodnota 1) nebo vypnutá (hodnota 0). Protože se jedná v podstatě o logické proměnné, tedy proměnné dvouhodnotové, je reprezentace dat v GA jednoduchá – chromozom obsahuje prvky vektoru **δ**, každý prvek tohoto vektoru je uložen v jednom genu a má navenek velikost 1 bit (fyzická délka genu může být větší s ohledem na redundantní kódování s použitím stínů). K výpočtu hodnoty účelové funkce pro každý chromozom se spouští GAMS jako externí úloha. GA pro GAMS vždy připraví hodnoty δ_{e_n} do textového souboru, GAMS provede řešení a vypočtenou optimální hodnotu uloží do textového souboru, odkud si ji převezme GA.

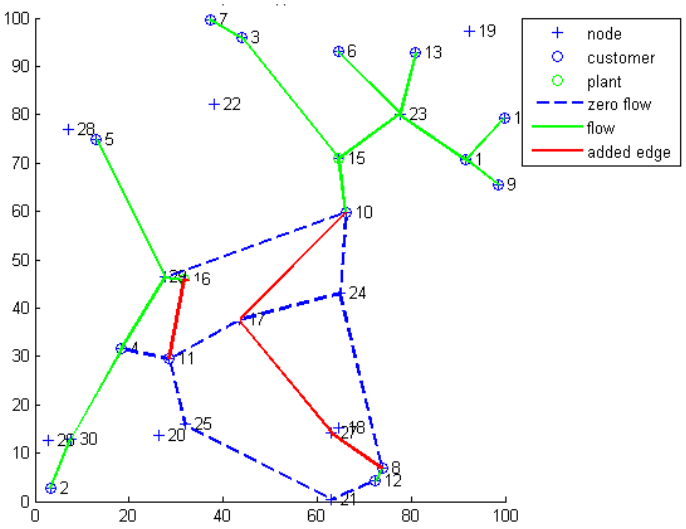
Při experimentech s testovacími úlohami malých rozměrů bylo zjištěno, že v některých případech je výsledná optimální hodnota účelové funkce dosažená hybridním přístupem horší, než optimální hodnota získaná kontrolním řešením pouze s využitím programu

GAMS. Analýzou tohoto jevu bylo zjištěno, že GA má tendenci připravovat scénáře, které obsahují velké množství zapnutých hran. V optimálním řešení určeném programem GAMS je ale v některých na povel GA zapnutých hranách nulový tok, neboť GA zapnuté hrany generuje genetickými operátory bez znalosti modelu. Tím se zhoršuje hodnota účelové funkce, neboť zbytečně obsahuje náklady na zapnutí hran, aniž by se jejich zapnutí příznivě projevilo. V textovém souboru předá proto GAMS nejen hodnotu účelové funkce, ale i hodnoty $\delta_{e_n}, \forall e_n \in E_n$, tedy toky všemi hranami. GA zjistí, ve kterých hranách, které požadoval zapnout, je nulový tok a odpovídající přebytečné jedničky v chromozomu odstraní. Upravené hodnoty δ_{e_n} opět uloží do textového souboru a znovu spustí GAMS pro přepočítání hodnoty účelové funkce. Po této korekci již výsledky získané klasickým i distribuovaným přístupem vycházely u testovaných úloh shodně.

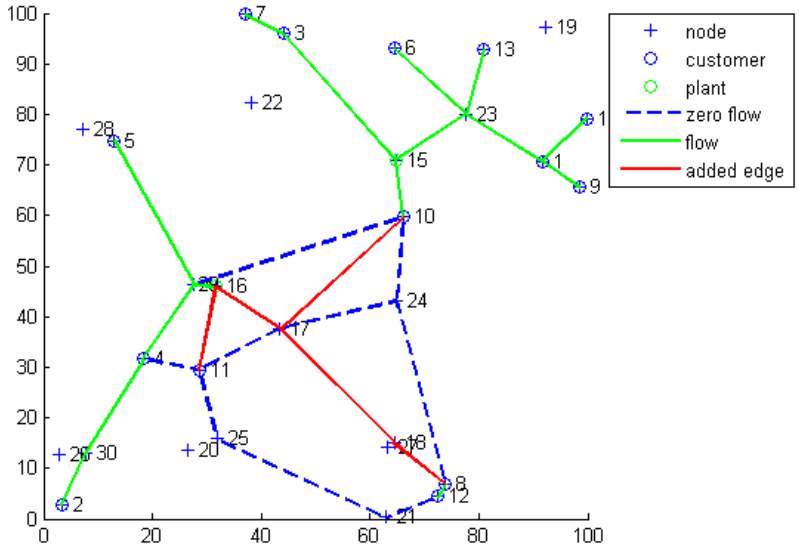
Motivací pro popsaný způsob realizace byla kromě ověření DOP i velikost řešené úlohy, kde i přes neefektivní spolupráci jednotlivých programů představovalo zvolené řešení poměrně vysoký přínos. GA v této úloze pracoval s poměrně malou populací o velikosti 40 jedinců. Typický počet iterací potřebných k získání výsledku byl 30, takže v nejhorším případě by GAMS byl teoreticky spuštěn 1200krát, s výše uvedenou korekcí 2400krát (ve skutečnosti byl počet spuštění podstatně nižší, neboť GA nepožaduje opakované vyhodnocení účelové funkce u jedinců, kteří se oproti předchozí iteraci nezměnili). Datová délka chromozomu byla 119 bitů, takže množina přípustných řešení obsahovala více než $6{,}6 \times 10^{35}$ kombinací. Tato hodnota by představovala počet nutných výpočtů pro případ použití plné enumerace (s použitím korekce dvojnásobek); což je samozřejmě realizačně naprosto nemožné.

DOP model je v základní verzi podobný jako v předchozím případě, tzn. je obdobou obr. 5.5. Zavedením korekce na nulové toky hranami se ale situace zkomplikovala. Ačkoliv po stránce programátorské se jedná o řešení jednoduché, logické a běžné, originální koncepce DOP modelu dle [38] je přece jen více matematickou než programátorskou abstrakcí a jednoduché grafické znázornění této situace neumožňuje.

Pro snazší možnost srovnání byly testovací příklady včetně vstupních dat převzaty z [17]. Model úlohy pro GAMS byl mírně upravenou verzí modelu dopravní úlohy z [16], ze stejného pramene byla rovněž převzata vizualizace výsledků na obr. 5.6 (deterministická poptávka) a 5.7 (stochastická poptávka). Grafy na obrázcích představují distribuční síť, ve které jsou vyznačeny hrany pro získané optimální řešení. Uzly představují zdroje (zelená barva) a spotřebitele – zákazníky (modrá). Zeleně jsou zakresleny pevné hrany s nenulovým tokem, modře čárkovaně pevné hrany s nulovým tokem a konečně červeně přidané hrany navržené genetickým algoritmem (v těch je samozřejmě ve všech nenulový tok).



Obr. 5.6: *Deterministický dopravní model s přidáváním hran (5.18) [54]*



Obr. 5.7: *Stochastický dopravní model s přidáváním hran (5.22) [54]*

Popsaný způsob spolupráce obou úloh je samozřejmě neefektivní, podobně jako v předchozí podkapitole i zde však jde především o ověření principu vzhledem k uvažované distribuovanosti modelu a řešících kódů. Navržený hybridní algoritmus se skládá z následujících kroků:

1. Inicializace, nastavení parametrů.
2. Vytvoření iniciální populace GA.
3. Ocenění členů populace GA. K výpočtu účelové funkce pro každého jedince je nutné vykonat následující kroky:

a. Uložení hodnot jednotlivých genů (prvků vektoru $\boldsymbol{\delta}$) do souboru. GAMS tato data zahrne do datového popisu modelu pomocí direktivy $INCLUDE.
b. Spuštění programu GAMS, ten jako výsledek uloží do souboru zjištěnou optimální hodnotu účelové funkce a hodnotu prvků vektoru $\mathbf{x}$ (hodnoty toků hranami).
c. GA si převezme výsledky. Zjistí, zda v některé hraně, kterou požadoval zapnout (odpovídající $\delta_{e_n} = 1$), je nulový tok. Pokud ne, je hodnota účelové funkce přijata jako ocenění daného jedince a pokračuje se krokem 4. Pokud ano, je dědičná informace příslušného jedince modifikována v tom smyslu, aby nebylo požadováno zapnout žádnou hranu, ve které v optimálním řešení dle GAMS byl zjištěn nulový tok (v dalších operacích jedinec vystupuje s touto upravenou a nikoliv původní genetickou informací). Znovu se spustí GAMS, jako ocenění jedince je nastavena hodnota účelové funkce tohoto druhého spuštění programu.

4. Uložení nejlepšího doposud známého řešení (hodnoty účelové funkce a odpovídajících prvků vektorů $\boldsymbol{\delta}$ a $\mathbf{x}$).
5. Pokud je splněna podmínka ukončení, zastavení algoritmu.
6. Vygenerování nové populace pomocí genetických operátorů. Pokračování krokem 3.

6 OPTIMIZATION TRANSFER PROTOCOL (OTP)

Při praktické realizaci DOP modelů a jejich řešení na počítači je značným problémem neexistence odpovídajícího modelovacího a softwarového nástroje. Použití speciálních optimalizačních nástrojů typu GAMS řeší problém jen částečně. Hlavní převratnou myšlenkou DOP je synchronizovaný nebo asynchronní běh několika optimalizačních elementů, které vhodně kooperují prostřednictvím výměny dat. Je vhodné, aby byla umožněna spolupráce různých optimalizačních prostředků běžících obecně na různých platformách a na různých počítačích a majících samozřejmě také rozdílný původ i softwarové provedení. Nabízí se myšlenka použití univerzální komunikační infrastruktury nezávislé na platformě a použité komunikační technologii a definice aplikačního protokolu nad touto infrastrukturou. Prakticky jedinou vhodnou komunikační infrastrukturou, která nabízí dostatečnou univerzalitu a prakticky univerzální podporu je komunikace využívající TCP/IP protokolu. V této části je naznačeno, jak při návrhu tohoto specializovaného aplikačního protokolu postupovat. Tento protokol byl pracovně nazván *Optimization Transfer Protocol*, ve zkratce OTP.

6.1 Rodina protokolů TCP/IP

Zahájení projektů, které ve svém důsledku vedly k vývoji protokolu TCP/IP (a posléze ke vzniku sítě Internet), sahá do šedesátých let 20. století. V období probíhající tzv. studené války mimo jiné vyvstal úkol zajištění komunikace mezi vládními a armádními úřady během a po případné atomové válce. Tomuto úkolu vyhoví síť, která bude schopná činnosti i při vyřazení některých jejích částí a která tedy nebude mít žádné centrum (tedy síť vybudovaná na prvcích, které dnes označujeme jakou router, to ale v počátcích vývoje nebylo tak zřejmé jako dnes). Je třeba podotknout, že v té době se pro všechny datové i hlasové přenosy používalo výhradně přepojování okruhů, paketizace byla dalším logickým krokem. Kromě této vojenské motivace se většina tehdy vedoucích výrobců počítačů začala problémy komunikace mezi počítači zabývat také z vlastního popudu, a to jak z důvodů budování terminálových sítí, tak i pro propojování počítačů navzájem, zpočátku zejména kvůli zálohování potřebnému v bankovních aplikacích a v postupně se rozvíjejících aplikacích pro řízení a sledování technologií v reálném čase. Tak ovšem vznikala izolovaná firemní řešení (např. DECNET firmy DEC[10] nebo SNA firmy IBM). Tato snaha

[10] Digital Equipment Corporation, jedna bývalých z vedoucích amerických společností v oblasti výpočetní techniky, založena 1957, v roce 1998 ji koupila firma Compaq, která byla později (v roce 2002) sama zakoupena firmou Hewlett-Packard. Počítače firmy DEC, zejména řada PDP, sehrály velmi významnou roli v historii výpočetní a komunikační techniky, např. operační systém UNIX byl vyvinut pro (mini)počítač PDP-7, jazyk C vznikl kvůli potřebě přenést UNIX na počítač PDP-11. Kopie počítačů DEC byly vyráběny i v bývalé ČSSR, DECNet byl v Československu provozován pod názvem SYRPOS (Systém riadenia počítačových sietí).

výrazně zesílila s nástupem mikropočítačů. Vývoj se zaměřoval jak na technické řešení propojení, tak na řešení softwarové.

Na počátku šedesátých let byl v USA a Velké Británii zahájen výzkum paketového přenosu. Významný impuls přinesl v roce 1962 výzkumný projekt financovaný agenturou DARPA[11]. Do výzkumu byla zapojena i akademická sféra. V roce 1969 byla uvedena do provozu experimentální síť *ARPANET*, tato síť obsahovala čtyři uzly – *UCLA* (University of California Los Angeles), *SRI* (Stanford Research Institute), *University of California Santa Barbara* a *University of Utah*. V roce 1972 byla u příležitosti konference ICCC (International Conference on Computers and Communications) představena síť *ARPANET demo*. Tato síť obsahovala cca 20 routerů a 50 počítačů a používala protokol **NCP** (*Network Control Protocol*). Protokol NCP byl určen pro experimentování a jeho rutinní provozní nasazení nebylo plánováno. V této síti byl zahájen provoz elektronické pošty. V roce 1973 byly k této síti připojeny první uzly ležící mimo USA (ve Velké Británii a v Norsku). (Pro představu o tehdejších technických prostředcích lze uvést, že ve stejném roce vyvíjí Robert Metcalf ve firmě Xerox síť Ethernet.) V letech 1977–1979 probíhal vývoj základní architektury TCP/IP ve spolupráci *Stanford University*, *BBN* (Bolt, Baranek and Newman) a *University College London*, v roce 1980 bylo zahájeno experimentální ověřování TCP/IP v síti ARPANET. Ve stejném roce byl protokol TCP/IP implementován pod operačním systémem BSD UNIX (spolupráce BBN a UCB – University of California Berkeley). V roce 1982 stanovilo ministerstvo obrany USA protokol TCP/IP standardem pro vojenské sítě, v roce 1983 se protokol TCP/IP stal standardním a jediným používaným protokolem pro síť ARPANET. Ve stejném roce dochází k oddělení sítě MILNET (*Military Network*) od sítě ARPANET. Za přispění firmy SUN se TCP/IP úspěšně přenáší i do komerční sféry. V té době model ISO/OSI[12] není ještě dopracován, jako alternativy k TCP/IP mohou ale sloužit firemní řešení – XNS (Xerox), DECNet, SNA (IBM).

Vývoj samozřejmě pokračoval dále a stále pokračuje, v roce 1990 rozrůstající se síť opouští datové linky původního ARPANETu a začíná se mluvit o síti Internet[13]. Mezi přelomovými daty je třeba uvést ještě alespoň rok 1992, kdy spatřila světlo světa služba *www*, a rok 1993, kdy byl vytvořen program *Mosaic*, který byl prvním grafickým www prohlížečem. Síť Internet dosáhla rozsahu, který v jejích počátcích nikdo neplánoval a ani si ho nedokázal představit. Adresace počítačů byla původně navržena s délkou 32 bitů. V době návrhu byly používány převážně sálové počítače a tak byl poskytovaný adresní prostor zdánlivě nevyčerpatelný. S masovým nasazením mikropočítačů a s obrovským nárůstem počtu připojených sítí i zařízení se však tento prostor ukázal být nedostatečným[14].

[11] Defense Advanced Research Project Agency, agentura ministerstva obrany USA

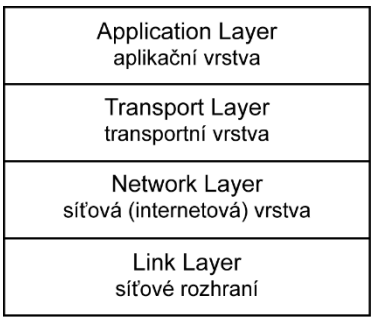
[12] Subkomise OSI (Open System Interconnection, původní název byl OSA – Open System Architecture)byla při ISO založena v roce 1977, tzv. referenční model ISO/OSI byl schválen v roce 1979. Tento model slouží jako myšlenkový model a neobsahuje konkrétní technické ani datové specifikace.

[13] Ačkoliv proti tomu jazykovědci občas protestují, je velké počáteční písmeno oprávněné, neboť internet (s malým písmenem na začátku) znamená jakékoliv propojení sítí, nikoliv jednu konkrétní síť.

[14] Problém činila nejen délka, ale i struktura adresy, protože kvůli požadavkům směrování nebylo možné ani zdaleka využít všechny existující kombinace. Detailní rozbor není pro účely tohoto textu nutný.

Proto bylo nutné vytvořit nový protokol s delšími adresami. Tímto protokolem je IPv6, který používá adresy o délce 128 bitů. Stávající protokol s adresami o délce 32 bitů se označuje jako IPv4 (předchozí tři verze byly fázemi vývoje protokolu, čtvrtá verze se používá dodnes).

Z uvedených skutečností je patrné, že vývoj TCP/IP a sítě Internet probíhal souběžně a že Internet je s touto rodinou protokolů silně (až neoddělitelně) spjat. To se projevuje nejen tím, že se v Internetu protokol TCP/IP používá pro datové přenosy, ale také v tom, že běžně používané aplikace a služby Internetu (e-mail, ftp, www apod.) jsou definovanými protokoly aplikační vrstvy (viz dále) TCP/IP.



Obr. 6.1: *Obvyklý vrstvový model TCP/IP*

Architektura protokolu (často se používá pojem *rodina protokolů*, ve skutečnosti se jedná o mnoho souvisejících a navzájem se využívajících protokolů) TCP/IP je na obr. 6.1. Je nutno podotknout, že existuje několik vrstvových modelů popisujících TCP/IP, jejichž počet vrstev se liší, takže obr. 6.1 představuje pouze jednu z možností, která je ale používána poměrně často (např. [57]). Členění do vrstev je pouze pomocné, důležité jsou jednotlivé protokoly, které jsou definované v závazných RFC[15] dokumentech. Další popis je zjednodušený a zestručněný, detailní popis protokolů a logiky fungování TCP/IP přesahuje rámec tohoto textu.

Hierarchicky nejnižší vrstva modelu je *vrstva síťového rozhraní* (*link layer*, někdy též *network interface, hardware* nebo *network access layer*). Z hlediska TCTP/IP je to vrstva poměrně nezajímavá, specifikuje síťový hardware a na něm bezprostředně závislé softwarové moduly (přenosové médium, přenos bitů, síťové rozhraní daného zařízení a jeho ovladače pro daný operační systém). TCP/IP neobsahuje specifikace této úrovně, předpokládá se fungující síť s vlastní specifikací (např. Ethernet, IEEE 802.3). Na úrovni této vrstvy se používá adresace zařízení pomocí tzv. hardwarových (fyzických) adres (MAC[16] adresa). Datovou strukturou typickou pro tuto vrstvu je rámec (frame), jehož struktura je

[15] RFC – Request for Comments, zkratka vznikla ze zavedeného předmětu e-mailů, které si mezi sebou zasílal vývojářský tým doktorandů na UCLA. Dokumenty jsou dostupné např. na adrese *http://www.rfc-editor.org/rfc.html* nebo *https://tools.ietf.org/html/*

[16] MAC = Medium Access Control, název je podle názvu podvrstvy druhé vrstvy modelu ISO/OSI. Adresa je obvykle přidělená výrobcem a má délku 48 bitů (nověji 64 bitů).

definována příslušnou síťovou technologií. V datové části rámce jsou přenášeny IP datagramy jakožto datové struktury následující vrstvy.

Síťová vrstva (network layer) někdy též označovaná jako *internetová vrstva* (internet layer) jako svůj nejdůležitější protokol obsahuje protokol **IP** (*Internet Protocol*), datová struktura tohoto protokolu se označuje IP datagram a pomocí těchto datagramů se přenášejí veškerá data v rámci TCP/IP komunikace[17]. IP protokol využívá pro adresaci zařízení již zmíněné IP adresy délky 32 (IPv4) nebo 128 (IPv6) bitů. Protokol IP je směrovatelný, nespojovaný a nespolehlivý[18] protokol. Definice struktury IP adresy (obou v současné době používaných verzí) umožňuje, aby data mohla být zaslána i aplikaci běžícímu ve stejném počítači, tzn. bez vysílání na síť. Kromě užitečných dat protokolů vyšších vrstev mohou být pomocí IP přenášena také služební data protokolů **ICMP** nebo **IGMP**. Protokol ICMP (*Internet Control Message Protocol*, RFC 792) je protokol řídících hlášení, poskytuje zprávy o chybách a zvláštních okolnostech při přenosu, obsahuje i diagnostické a informační nástroje. Mimo jiné obsahuje podporu pro často používané diagnostické nástroje *ping* a *traceroute*. Protokol IGMP (*Internet Group Management Protocol*), nejnověji specifikován v RFC 4604, slouží v IPv4 sítích[19] k řízení příslušnosti zařízení do skupin adresovaných skupinovou adresou. Na rozhraní první a druhé vrstvy jsou řazeny protokoly **ARP**[20] a **RARP**[21] (resp. jeho novější náhrady), které slouží k převodu mezi IP adresami a fyzickými adresami. Pro zařazení do první vrstvy hovoří skutečnost, že jsou tyto protokoly závislé na použité síťové technologii, pro zařazení do vrstvy druhé potom fakt, že k přenosům dat tyto protokoly používají IP datagramy.

Transportní vrstva zprostředkovává datový tok mezi účastníky komunikace, obsahuje dva protokoly, které jsou již přímo využitelné aplikacemi, a sice **UDP** a **TCP**[22]. *UDP* (*User Datagram Protocol*) je nespojovaný nespolehlivý protokol, který představuje obdobu IP protokolu nabízenou aplikacím, datová struktura se nazývá paket (packet). Protokol je jednoduchý, má minimální režii a je určen pro předávání malého objemu dat (ideálně by se data měla vejít do jednoho paketu) nenáročným aplikacím, obvykle charakteru dotaz/odpověď nebo jednorázového (občasného nebo periodického) zasílání malého objemu dat bez požadavku potvrzení doručení (typicky např. meteorologická stanice). *TCP* (*Transmission Control Protocol*) je spojovaný a spolehlivý protokol, který po navázání

[17] Ačkoliv je možné, aby aplikace použila pro komunikaci přímo IP protokol, nelze tuto praxi příliš doporučit. Pro použití aplikacemi jsou určeny protokoly TCP a UDP transportní vrstvy.

[18] Přívlastkem nespolehlivý (unreliable) se rozumí skutečnost, že protokol nepracuje s potvrzováním a tudíž se nelze spolehnout na to, že odeslaná data budou úspěšně doručena a přijata, odesílatel užívající IP protokol o tom nedostane žádnou informaci.

[19] V IPv6 je použit odlišný protokol, a sice *Multicast Listener Discovery* (MLD), který je součástí ICMPv6.

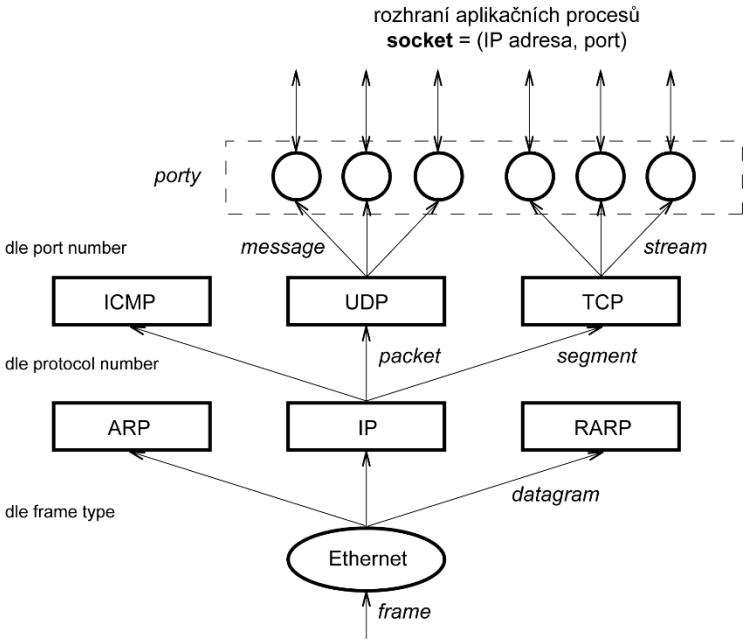
[20] ARP = *Address Resolution Protocol*, zjištění fyzické adresy pro známou IP adresu.

[21] RARP = *Reverse Address Resolution Protocol*, zjištění IP adresy při známé fyzické adrese. Tato potřeba nastává nejčastěji při konfiguraci zařízení po startu. K úspěšné komunikaci v TCP/IP sítích ovšem dnes samotná znalost IP adresy nestačí, proto jsou namísto RARP používány novější protokoly, obvykle DHCP (*Dynamic Host Configuration Protocol*).

[22] Označení rodiny protokolů TCP/IP znamená „TCP over IP“, tedy přenos dat protokolu TCP pomocí IP datagramů.

spojení umožňuje obousměrný datový tok mezi komunikujícími zařízeními. Aplikace se nemusí starat o detekci a ošetřování chyb při přenosu. Detailní popis protokolu lze najít přímo v příslušném RFC dokumentu [66].

Protože hardwarová i IP adresa identifikují síťové rozhraní počítače. Na počítači může běžet současně více úloh, proto je nutné řešit také adresaci jednotlivých úloh. Každá úloha pro svoji identifikaci při komunikaci používá tzv. *číslo portu*. Jedná se o šestnáctibitovou hodnotu přidělenou každému účastníku komunikace. Kombinace IP adresy a čísla portu se nazývá soket (*socket*) a představuje jednoznačnou adresu úlohy v celém Internetu[23]. Číslo portu může být aplikaci přiděleno dynamicky (na požádání při startu komunikace) nebo pevně (tzv. *well-known port*). Aplikace, která komunikaci zahajuje, musí znát adresu svého protějšku včetně čísla portu, aby mu mohla zaslat data. Sama ovšem může použít dynamicky přidělené číslo portu, které zašle svému partnerovi při zahájení komunikace. U služeb s architekturou klient/server je proto nutné, aby číslo portu, které používá server, bylo klientům známé. Podobně jako IP adresy nelze ani čísla portů přidělovat a používat zcela libovolně. Původně čísla portů přidělovala IANA[24], od 21. března má toto na starosti ICANN[25]. Pro známé porty přiřazené běžným síťovým službám je vyhrazen rozsah 0–1023, rozsah 1024–49151 je určen pro tzv. registrované porty (jejich použití by se mělo registrovat u příslušné autority), zbývající rozsah 49152–65535 je určen pro volné použití a dynamické přidělování [57].



Obr. 6.2: *Tok dat v modelu TCP/IP (při příjmu)*

[23] Text neuvažuje např. problematiku privátních adres a NAT (Network Address Translation), která toto tvrzení do značné míry relativizuje. Tyto detaily nejsou pro tento text důležité.
[24] *Internet Assigned Numbers Authority*
[25] *Internet Corporation for Assigned Names and Numbers*, nezisková organizace, která dohlíží na technické a organizační záležitosti související s Internetem.

Aplikační vrstva obsahuje definice aplikačních protokolů, které tvoří příslušné síťové služby určené přímo uživatelům nebo zajišťující provoz sítě s využitím protokolů nižších vrstev (FTP, HTTP, SMTP, DNS , směrovací protokoly apod.). Popis síťových služeb není cílem tohoto textu.

Na obr. 6.2 je znázorněna situace při příjmu dat. Jako příklad síťové technologie je na obrázku předpokládán Ethernet, vybrány byly pochopitelně pouze některé protokoly. *Frame type* je osmibitová hodnota, která udává, jaký druh dat je obsahem rámce, čímž je určeno, jakému protokolu druhé vrstvy budou data předána. IP protokol předává data vyšším protokolům podle tzv. *čísla protokolu* (*protocol number*, délka 16 bitů). Transportní protokol potom předává data aplikaci podle čísla portu, jak již bylo uvedeno.

Z uvedené stručné charakteristiky rodiny protokolů TCP/IP plyne, že tato představuje solidní univerzálně použitelný základ pro komunikaci mezi úlohami, kde nezáleží na technickém zařízení, operačním systému, umístění, způsobu připojení ani způsobu naprogramování příslušné aplikace. TCP/IP bylo od počátku koncipované pro heterogenní sítě a dnes je tzv. otevřeným standardem, popisy protokolů jsou veřejně k dispozici.

6.2 Návrh OTP

V kapitole 5 je uvedeno několik praktických úloh, na jejichž řešení bylo možné nahlížet jako na DOP úlohy. U všech bylo konstatováno, že jejich programová realizace je neefektivní. Tato neefektivnost měla několik příčin. Předně byly úlohy v podkapitolách 5.2 a 5.3 realizovány jako heterogenní, kdy jedna část byla speciálně naprogramovaná pro daný konkrétní případ (genetický algoritmus), a druhá byla řešena komerčním optimalizačním softwarem (GAMS). Ačkoliv tento způsob realizace byl funkční a umožnil nalézt uspokojivý způsob řešení rozsáhlých stochastických úloh jistého typu, který by byl zobecnitelný i pro jiné úlohy, přinášel zároveň dva podstatné zdroje zpomalení výpočtu[26]. Jednak se jednalo o předávání dat mezi oběma programy prostřednictvím diskového souboru, jednak o nutnost opakovaně spouštět GAMS z genetického algoritmu vždy, když bylo vyžadováno vyhodnocení účelové funkce. I když např. u problému popsaného v podkapitole 5.3 použití genetického algoritmu přineslo oproti metodě úplné enumerace snížení počtu vyhodnocení účelové funkce o více než 30 řádů (!), přesto šlo o řešení nouzové a z programátorského hlediska „neestetické“. U úlohy dle podkapitoly 5.1, která byla řešena pomocí spolupráce dvou genetických algoritmů, zase byla možná paralelizace výpočtu, která ale z realizačních důvodů zůstala nevyužita. Pro úplnost je třeba připo-

[26] Je otázkou, zda hovořit o zpomalení v situaci, když při dalším zvětšování rozsahu úlohy by se úloha stala klasickým způsobem neřešitelnou. Zde je myšlena spíše skutečnost, že výpočet by mohl proběhnout i rychleji, pokud by byla realizace ještě sofistikovanější, i když by to vyžadovalo vynaložení podstatně většího objemu programátorské práce.

menout skutečnost uvedenou v odstavci 3.3.9, že genetické algoritmy, stejně jako jiné heuristické algoritmy pracující s populací řešení, přímo obsahují vnitřní paralelismus v ohodnocování jednotlivých členů populace – výpočet hodnoty účelové funkce probíhá pro každého člena populace nezávisle, a pokud tomu nebrání technická omezení, může probíhat výpočet pro všechny jedince současně, tedy paralelně. Této možnosti, která by mohla přinést zásadní snížení výpočetního času, se však používá jen zřídka. Realizačně jednodušší varianta spočívající ve spolupráci několika GA s menšími populacemi je uvedena v [29], kde se však jednalo o sice zajímavé, leč také spíše nouzové řešení.

DOP obsahuje významné dekompoziční a modelovací možnosti. Tento teoretický koncept by bylo možné vhodnou programovou realizací využít na kvalitativně vyšší úrovni než v jednoúčelových realizacích uvedených v kapitole 5. V dalším textu této podkapitoly je naznačen ideový návrh podobného řešení včetně některých konkretizací pro jednodušší případy.

Distribuovaná podstata DOP vede jednoznačně na spolupráci více úloh. Jeví se jako rozumné, aby byl každý DOP prvek realizován jednou úlohou (programem). Pro volbu způsobu komunikace mezi úlohami je vhodným kandidátem protokol TCP. Výhodou je jednak skutečnost, že tento protokol je široce podporovaný, jednak to, že při jeho použití nezáleží na tom, zda komunikující úlohy běží na stejném nebo na různých, vhodnou sítí propojených počítačích. Současně poskytuje vcelku luxusní prostředek pro přenášení bloků dat libovolné délky oběma směry, aniž by bylo nutné v aplikaci řešit potvrzování, detekci a ošetřování chyb při přenosu (samozřejmě kromě fatální chyby spojení).

Pro návrh řešení byly stanoveny následující zásady:

- každý DOP prvek bude realizován jedním samostatným programem,
- programy mohou běžet na stejném nebo na různých počítačích, v případě různých počítačů nemusejí nutně používat shodnou platformu,
- jednotlivé programy budou komunikovat s využitím protokolu TCP,
- pro snadnější sestavení a konfiguraci celé DOP úlohy nebude konfiguraci úplné úlohy znát každý prvek, ale pouze jeden hlavní prvek (*master element*), ostatní prvky budou označovány jako podřízené (*slave elements*),
- s ohledem na předchozí zásadu nebudou ve většině případů jednotlivé podřízené prvky komunikovat přímo mezi sebou a většina komunikace se bude odehrávat zprostředkovaně pomocí master uzlu, výjimkou může být požadavek na provedení dílčího výpočtu nebo na jednorázové zaslání dat např. do vizualizačního prvku,
- popis úlohy i konfigurace jednotlivých prvků bude soustředěna na jednom místě (v centrálním uzlu), konfigurační data pro podřízené uzly budou zcela minimální a pokud možno na úloze nezávislá (např. jejich číslo portu),
- pokud podřízené uzly vyžadují ke své činnosti inicializaci (např. GA potřebuje znát parametry, jako je velikost populace, počet genů v chromozomu apod.),

budou jim v souladu s předchozí zásadou zaslány centrálním uzlem (důvodem je centralizace popisu úlohy a konfigurace na jednom místě),

- všechny elementy (resp. programy, které je realizují) poběží po celou dobu výpočtu; pokud to bude možné, zajistí jejich spuštění centrální uzel před zahájením výpočtu,
- podřízené prvky budou z hlediska TCP protokolu realizovány jako servery (po spuštění musí být schopné kdykoliv přijmout data od centrálního prvku),
- budou existovat různé typy podřízených prvků, hlavními typy budou *výpočetní prvky* (akceptují vstupní data a poskytují výstupní data) a *datové spotřebiče* (typicky např. vizualizační prvky sloužící k zobrazení průběhu řešení),
- pokud budou nutné transformační prvky (prvky realizující datové transformační funkce Φ a ψ, viz 4.2), budou realizovány jako zvláštní případ výpočetních uzlů.

Předpokládá se návrh a realizace komunikačních a jiných režijních funkcí pro platformu Windows uložených do dll knihovny, a tím vytvoření jakési „softwarové stavebnice“. Je zřejmé, že kromě naprogramování centrálního prvku a knihovny komunikačních funkcí je nutné řešit jednotlivé podřízené výpočetní uzly (pro realizaci jednotlivých optimalizačních algoritmů a některých pomocných výpočtů – např. uzly pro zdroje neurčitostí). Portfolio těchto uzlů by mělo být dostatečně široké, aby byla pokryta co nejširší oblast řešených úloh. Vzhledem k programátorské i algoritmické náročnosti realizace těchto výpočetních uzlů samozřejmě nelze předpokládat, že by vznikající moduly v krátké době pokryly možnosti komerčních optimalizačních softwarových produktů, jako je např. GAMS. Proto budou existovat jednoduché programy, které budou sloužit jako obálky pro zapouzdření optimalizačních programů, které nejsou schopny spolupracovat s DOP úlohami (např. GAMS). Tato obálka zajistí komunikaci s centrálním uzlem (a podle potřeby i s ostatními uzly), zajistí spuštění příslušného programu a předá jeho výsledky. Tímto způsobem byla mimochodem vyzkoušena reálnost využití výše uvedených principů, a to na modifikaci úlohy dle 5.2. Ačkoliv toto řešení vypadá jako na první pohled polovičaté a provizorní, lze existenci těchto typů uzlů považovat i za výhodu, neboť tímto způsobem lze jednoduše zajistit spolupráci systému DOP s jiným libovolným softwarem.

Samozřejmý je požadavek na řízení celého systému daty. To znamená, že pro realizaci jakékoliv DOP úlohy nebude v ideálním případě nutné nijak zasahovat do programového kódu žádného uzlu, pouze se vytvoří datový popis úlohy. Pokud budou jednotlivé prvky naprogramované s využitím překládaného jazyka, může nastat drobný problém se zápisem výrazů představujících účelovou funkci a omezení. Nabízejí se dvě řešení – buď zahrnutí syntaktického analyzátoru a obecného výpočetního modulu přímo do výpočetních uzlů, nebo, pro složitější případy, možnost externího výpočtu účelové funkce. Ve druhém případě výpočet provede jednoúčelový (realizačně velmi jednoduchý) výpočetní prvek vytvořený speciálně pro potřeby řešení dané úlohy.

Název této kapitoly hovoří o návrhu protokolu. Definice protokolu představuje návrh datových struktur a pravidel chování účastníků komunikace. Zatímco pravidla chování lze vyčerpávajícím způsobem specifikovat, u datových struktur spočívá problém v obrovské rozmanitosti předávaných dat, kde ve fázi předběžného návrhu zdaleka nelze postihnout všechny možnosti, které bude při realizaci nutné řešit. Přesto je užitečné popsat hlavní rysy, které se vztahují k principu činnosti DOP úloh.

Vypočet je řízen *centrálním prvkem*. K řízení postupu výpočtu patří mimo jiné inicializace podřízených prvků, zadávání úkolů těmto uzlům a přebírání výsledků od těchto uzlů. Centrální uzel řídí také synchronizaci výpočtu. Podporuje paralelizaci v tom smyslu, že umožňuje spustit více výpočtů současně. Vyžaduje-li to charakter výpočtu, lze zařadit čekání na skončení výpočtu specifikovaného výpočetního uzlu. Protože se většina výpočtů odehrává v podřízených prvcích, nejsou na výpočetní rychlost centrálního prvky kladeny vysoké nároky a může být realizován v jakémkoliv, tedy i interpretovaném programovacím jazyce. Při ověřování experimentálních verzí centrálního prvku byl používán jazyk C++.

Datový popis DOP úlohy je využíván centrálním uzlem, podřízené uzly jej nemají k dispozici. Popis je tvořen textovým souborem, který musí být uložen tak, aby byl dostupný centrálnímu uzlu. Pro zvýšení čitelnosti a vícenásobné použitelnosti dat je implementováno vnořování (vkládání) souborů (include). Při návrhu byla volba textového popisu jednoznačná, zvažován byl ale formát. Rozšířený a progresivní formát XML sice přináší některé výhody (např. možnost vnořování prvků), ovšem s ohledem na čitelnost a snadné vytvoření a modifikace souborů ve fázi vývoje byl nakonec zvolen prostý textový formát v některých částech inspirovaný strukturou ini souborů MS Windows. Ukázka popisu jednoduché úlohy je odstavci 6.2.1.

Popis má dvě části – popis prvků a popis činnosti. *Popis prvků* obsahuje popisy jednotlivých prvků použitých v úloze, pořadí, v jakém jsou prvky popisovány, i pořadí jednotlivých datových položek popisu je libovolné. U každého prvku je uvedena jeho *identifikace* (textový název, kterým je identifikován v popisu prvků i úlohy), *síťová lokalizace* (IP adresa nebo jméno počítače a port), tyto položky jsou povinné. Volitelně dále může být uvedena *lokalizace spustitelného souboru* (pro případ, že spuštění programu reprezentujícího prvek má provádět centrální prvek). Následuje *popis formátu vstupních dat*, a pokud prvek produkuje výstup, tak také *popis formátu výstupních dat*. Pro optimalizační výpočetní prvky je dále specifikována účelová funkce, a to buď výrazem, nebo specifikací uzlu, který výpočet hodnoty účelové funkce zajistí. Následovat mohou *specifická data* typická pro jednotlivé druhy výpočetních prvků.

Identifikace jednotlivých elementů pro účely textového popisu úlohy je tvořena *unikátním textovým jménem*, pro které platí stejné podmínky, jaké jsou stanoveny pro identifikátory v jazyce C, maximální délka jména je 32 znaků. Při komunikaci nejsou tato textová jména zasílána, odesílatel i příjemce jsou identifikováni svojí síťovou adresou a číslem portu.

Struktura zasílaných zpráv je velmi jednoduchá. Na začátku je vždy šestnáctibitová hodnota popisující typ zprávy, následuje datová část, která nemá pevný formát ani délku a liší se nejen pro jednotlivé typy zpráv, ale také pro různé prvky a různé řešené úlohy. Popis typů zasílaných zpráv je v tabulce 6.1. Zprávy jsou zasílány v tzv. *OTP transakcích*. OTP transakce představuje navázání spojení, zaslání zprávy, přijetí reakce protistrany (s výjimkou situace, kdy byla zaslána zpráva M_DATA, kdy se žádná odpověď nezasílá) a ukončení spojení.

Tab. 6.1: *Typy zpráv OTP*

Kód	Typ zprávy	Poznámka
0	M_INI	Žádost o inicializaci, zaslání inicializačních dat
1	M_RESET	Uvedení do výchozího stavu
2	M_COMPUTE	Žádost o výpočet, zaslání vstupních dat
3	M_DATA	Zaslání obecných dat
4	M_RESULT	Zaslání výsledku po dokončení výpočtu
5	M_STATUS	Signalizace úspěšnosti nebo chyby operace

Pro popis činnosti DOP úlohy byl vyvinut jednoduchý jazyk, jehož jednotlivé příkazy popisují jednotlivé akce. V podstatě se jedná o povely k odeslání zpráv od centrálního prvku prvkům podřízeným a synchronizaci. Za klíčovým slovem reprezentujícím příkaz se nachází jako první povinný parametr textové jméno uzlu, kterého se akce týká, potom mohou následovat parametry specifické pro daný příkaz. Pro jednoduchost zpracování je možné na jednom řádku uvést pouze jeden příkaz. V současné experimentální verzi centrálního prvku jsou implementovány následující příkazy:

RUN *prvek*
Spuštění programu odpovídajícího prvku (implementováno pouze pro programy běžící na stejném počítači jako centrální prvek).

INIT *prvek, <data>*
Požadavek na inicializaci, součástí zprávy jsou inicializační data. Inicializovaný prvek odpovídá zprávou M_STATUS, nulová hodnota kódu výsledku znamená úspěšné dokončení, centrální prvek čeká na odpověď.

RESET *prvek*
Specifikovanému prvku je poslána zpráva M_RESET, oslovený prvek odpovídá zprávou M_STATUS, nulová hodnota kódu výsledku znamená úspěšné dokončení, centrální prvek čeká na odpověď.

SEND *prvek, <data>*
Specifikovanému prvku je poslána zpráva M_DATA s příslušnými daty. Prvek nijak neodpovídá.

COMPUTE *prvek, <data>*
Specifikovanému prvku je poslána zpráva M_COMPUTE a příslušná vstupní data. Prvek odpovídá zprávou M_RESULT obsahující výsledek nebo zprávou M_STATUS se signalizací chyby (nenulová hodnota kódu výsledku). Centrální prvek nečeká na dokončení.

COMPUTE_W *prvek, <data>*
Modifikovaná varianta příkazu **COMPUTE**. Specifikovanému prvku je poslána zpráva M_COMPUTE a příslušná vstupní data. Prvek odpovídá zprávou M_RESULT obsahující výsledek nebo zprávou M_STATUS se signalizací chyby (nenulová hodnota kódu výsledku). Centrální prvek čeká na dokončení.

WAIT *prvek*
Pokud byla specifikovanému prvku poslána zpráva M_COMPUTE, tak centrální prvek počká na dokončení operace (výsledek nebo chybová zpráva).

LOOP *n*
Uvození začátku iterace, *n* je počet iterací. Úsek programu mezi příkazem **LOOP** a klíčovým slovem **END** bude opakována *n*-krát.

Zpráva M_COMPUTE spouští výpočet ve výpočetním prvku, výsledek výpočtu je vždy zaslán žadateli (tak je mimo jiné ošetřen také výpočet hodnoty účelové funkce externím uzlem). Pro snazší zápis je možné v popisu běhu úlohy používat proměnné, takže např. provedení příkazu COMPUTE může být v konfiguračním souboru zapsáno např. ve tvaru (X, Z) = COMPUTE OPT1, A. Zde se předpokládá, že prvek OPT1 je prvkem provádějícím výpočet optima. V příkladu symboly X, Z a A reprezentují proměnné. A je vstup do výpočtu, X je hodnota rozhodovacího vektoru odpovídající optimálnímu řešení, Z je nalezená optimální hodnota účelové funkce.

Pro pohodlnou práci je vhodné, aby centrální prvek umožňoval používání matematických a případně i jiných funkcí. V současné době je realizována jediná funkce, a sice funkce RND pro generování pseudonáhodných čísel.

Je zvažována možnost, kdy celý popis úlohy bude možné vygenerovat (a modifikovat) interaktivně centrálním uzlem, tento cílový stav je ovšem ještě dosti vzdálený.

6.2.1 Výpočetní uzel obsahující genetický algoritmus

Zvláštní pozornost byla věnována začlenění genetického algoritmu do DOP. Důvodem byla jednak specifika GA, jednak dlouholetý zájem autora o problematiku GA. Již v rámci práce [49] byl implementován GA jako objekt v jazyce C++. Tato implementace byla postupně modifikována a posloužila za základ programu pro realizaci DOP výpočetního prvku GA. Výhodou objektové realizace je možnost v programu za chodu vytvořit instanci objektu GA se zvolenými parametry, tyto parametry jsou GA zasílány jako inicializační data. Byl vytvořen program realizující OTP verzi GA jako konzolová aplikace systému Windows. GA provádí minimalizaci.

Úloha se chová jako server, tzn. že po spuštění zahájí naslouchání na portu daném konfigurací. Úloha očekává OTP transakci se zprávou M_INIT. Po obdržení inicializačních dat si tato data aplikace uloží, odpovídá zprávou M_STATUS s kódem 0 (nebo 101 v případě obdržení neplatných dat) a je připravena akceptovat požadavky na výpočet. Jako reakce na zprávu M_COMPUTE je vytvořena instance GA a spuštěn výpočet. Výsledek výpočtu je zaslán žadateli zprávou M_RESULT, konkrétně je zaslán chromozom odpovídající nejlepšímu získanému řešení v binární podobě a nejlepší hodnota účelové funkce. Po odeslání výsledku je instance GA zrušena, inicializační parametry jsou zapamatovány pro případný další výpočet. Pokud byl požadován výpočet aniž předcházela inicializace, odpovídá uzel GA zprávou M_STATUS s kódem chyby 100.

Pro výpočet účelové funkce se při experimentech předpokládalo výhradně vyhodnocení funkce pomocí externího výpočtu (v jiném prvku), zaslání funkčního předpisu není zatím možné. Pro tento externí výpočet zasílá GA zprávu M_COMPUTE uzlu určeném v konfiguraci pomocí běžné OTP transakce a čeká na výsledek.

Pro otestování použitelnosti myšlenek OTP byla řešena zjednodušená verze úlohy popsané v podkapitole 5.1. Zjednodušení spočívalo v tom, že řešení bylo prováděno pouze pro jednu hodnotu **x** a **y**, nikoliv pro 40 hodnot jako v 5.1. Tím se dosáhlo snížení nároků na experimentální verzi centrálního prvku. Podle očekávání toto zjednodušení vedlo k nižší úspěšnosti řešení, kdy řešení srovnatelné s řešením dle 5.1 se nepodařilo nalézt v každém běhu úlohy, ale pouze přibližně v jedné třetině běhů. Přesto je test nutno považovat za úspěšný, neboť prokázal funkčnost a použitelnost zvolené koncepce OTP.

Pro popis úlohy byl použit níže uvedený text, znak středník se používá pro uvození komentáře, význam jednotlivých řádků je uveden přímo v textu.

```
[NODE]
;
name = GA1
location = otp://localhost:27015/
cfce = otp://localhost:27017/    ; účelová funkce
;
[NODE]
;
name = GA2
location = otp://localhost:27016/
cfce = otp://localhost:27018/    ; účelová funkce
;
[NODE]
name = VIZUALIZACE
location = otp://localhost:27019/
;
[GA1.INI]
;inicializační parametry (společné pro oba GA)
;
; parametry pro gen
; -----------------
;
gen_psex = 0            ; bez sex. reprodukce
gen_nbit = 3            ; redundance 3 bity
gen_shade = 1           ; budou stíny
```

```
gen_min1 = 5            ; vysledek 1: pro hodnoty genu 101, 110, 111
gen_max0 = 2            ; vysledek 0: pro hodnoty genu 000, 001, 010
gen_tmut = 1            ; typ mutace:
                        ; 0: mění se všechny bity genu náhodně
                        ; 1: negace jednoho náhodně vybraného bitu
gen_pmut = 5            ; pravděpodobnost mutace genu
gen_ncross = 0          ; typ křížení na úrovni genu
                        ; 0 - uniformní,
                        ; 1 - celý gen od náhodně vybraného rodiče
;
; parametry pro chromozom
; -----------------------
;
chrom max age = 5       ; bude stárnuti, max. 10 generací
chrom_n_genes = 16      ; délka chromozomu v genech
chrom_n_cross = 0       ; křížení na úrovni chromozomu,
                        ; 0 - uniformní,
                        ; > 0 - počet hranic pro křížení
                        ; (tj. počet úseků - 1)
chrom_smart_new = 2     ; postup inicializace - náhodně
;
; parametry pro algoritmus
; ------------------------
;
ga_popsize = 10         ; pocet jedincu populace
ga_iterations = 40      ; pocet iteraci GA
;
[COMPUTATION]
;
VAR X : 16b             ; proměnná, délka 16 bitů
VAR Y : 16b
INIT GA1, #GA1.INI      ; zaslání inicializace GA1
INIT GA2, #GA1.INI      ; inicializace GA2 (totožné paranmatry jako GA1)
X = RND(65535)          ; náhodné počáteční řešení X
(Y,Z) = COMPUTE_W GA2   ; výpočet Y_opt pro dané X
LOOP 50                 ; opakovat 50x
(X,Z) = COMPUTE_W GA1
(Y,Z) = COMPUTE_W GA2
SEND VIZUALIZACE,X,Y,Z  ; zobrazení (mezi)výsledků na konzole
END
```

Jak je zřejmé z popisu úlohy, byly použity celkem čtyři prvky – centrální prvek, dva identické výpočetní prvky obsahující genetický algoritmus a jeden vizualizační prvek. Všechny odpovídající programy byly naprogramovány v C++ jako konzolové aplikace operačního systému MS Windows. Dále je z konfigurace zřejmé, že při testu byly všechny programy na jednom počítači (specifikace localhost v lokalizaci všech uzlů). Výpočtu účelové funkce obou genetických algoritmů zajistil centrální prvek, jinak samozřejmě bylo možné realizovat dva další výpočetní prvky. Centrální prvek současně prováděl dekódování dat chromozomu. Řešení, kdy účelovou funkci (účelové funkce) vyhodnocuje centrální prvek, se jeví být do budoucna jako zajímavé. Nabízí se možnost popsat výpočet těchto funkcí přímo v konfiguraci úlohy, čímž se dosáhne zvýšení kompaktnosti popisu úlohy a eliminuje se nutnost zásahů do zdrojových kódu jednotlivých programů při změně řešené úlohy.

7 ZÁVĚR

Tato práce se zabývá problematikou distribuovaných optimalizačních programů s cílem rozpracování teoretického konceptu DOP k možnostem praktického využití pro reálné výpočty. Po stručném úvodu do problematiky optimalizace, vybraných optimalizačních problémů a metod a pojednání heuristických metod se zvláštním důrazem na genetické algoritmy, je popsán koncept DOP, který je původně zaměřen na modelování. Dále jsou uvedeny příklady použití DOP při řešení vybraných komplikovaných optimalizačních úloh. Ve všech těchto úlohách jsou používány genetické algoritmy, ve dvou případech potom jsou genetické algoritmy použity v kombinaci s optimalizačním softwarem GAMS. Dále je v kapitole 6 představena nová původní koncepce softwarové realizace DOP úloh využívající důsledně distribuovaný přístup nejen pro modelování, ale i pro výpočty. Tato koncepce je založena na komunikaci jednotlivých úloh, které reprezentují DOP elementy, pomocí původního protokolu OTP, jehož ideový návrh je součástí této práce. Byla realizována experimentální verze centrální úlohy a experimentální verze vybraných výpočetních uzlů, pomocí kterých byla úspěšně ověřena funkčnost a použitelnost návrhu OTP uvedeného v kapitole 6.

Návrh OTP se důsledně drží ideje distribuovaného optimalizačního systému. Kromě zvýšení výpočetní rychlosti využitím paralelního zpracování nebo možnosti řešit rozsáhlejší úlohy metodou dekompozice lze v tomto přístupu spatřovat i nový pohled na přístup k modelování a řešení rozsáhlých optimalizačních úloh. Distribuovaný přístup je současným trendem v mnoha oblastech aplikací výpočetní techniky, a to v komunikacích, databázích, výpočtech, v řídicích aplikacích a v mnoha dalších. Prakticky u všech těchto oblastí je optimalizace důležitou součástí jejich realizace, někdy dokonce nezbytnou. Distribuovaný přístup k optimalizacím umožňuje přizpůsobení struktury řešených optimalizačních úloh reálným podmínkám, kdy lze provádět na nižších úrovních dílčí optimalizace, které budou interagovat s ostatními dílčími optimalizacemi, a které se budou podílet na celkovém výsledku.

Představený koncept je funkční. Další práce budou směřovány do realizace komunikační knihovny OTP, do realizace již nikoliv experimentální, ale provozní verze centrálního prvku, a do průběžné tvorby programů pro výpočetní prvky tak, aby postupně pokryly dostatečně širokou škálu typů optimalizačních úloh (a tedy různých odvětví matematického programování). Vytčené cíle jsou sice náročné, leč nikoliv nereálné.

Vzhledem k tomu, že až dosud byl teoreticky pojatý DOP převážně modelovacím, nikoliv výpočetním nástrojem, ukázalo se, že použití pro výpočty přináší na DOP další kategorii požadavků (např. v podkapitole 5.3 nebylo možné vcelku bezproblémovou programovou realizaci řešení úlohy pomocí DOP modelu uspokojivě popsat). Proto lze čekat, že realizace OTP přispěje i k dalšímu rozvoji a prohloubení koncepce DOP.

LITERATURA

[1] Alander J. T.: *An Indexed Bibliography of Genetic Algorithms in Operations Research*. Report 94-1-OR, Department of Information Technology and Production Economics, University of Vaasa, January 1997.

[2] Austand S. N.: *Proč stárneme*. Mladá fronta, Praha, 1999. ISBN 80-204-0804-5.

[3] Barbosa H. J. C.: *A Coevolutionary Genetic Algorithm for Constrained Optimization*. In Proceedings of CEC'99, Washington DC, 1999, pp. 1605–1611. ISBN 0-7803-5536-9.

[4] Bazaraa M. S., Jarvis J. J., Sherali H. D.: *Linear Programming and Network Flows*. Wiley and Sons, 2010. ISBN 978-0-470-46272-0.

[5] Bradley S. P., Hax A. C., Magnanti T. L.: *Applied Mathematical Programming*. Addison-Wesley, 1977. ISBN 978-0201004649.

[6] Dawkins R.: *Sobecký gen*. Mladá fronta, Praha, 1998. ISBN 80-204-0730-8.

[7] Dupačová J., Consigli G., Wallace S. W.: *Scenarios for Multistage Stochastic Programs*. Anals of Operations Research, 100:25–53, 2000. ISSN 0254-5330.

[8] Goldberg D. E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. ISBN 978-0201157673.

[9] El-Ghazali T.: *Metaheuristics: From Design to Implementation*. Wiley, 2009. ISBN 978-0470278581

[10] Ghiani G., Laporte G., Musmanno R.: *Introduction to Logistic Systems Planning and Control*. John Wiley & Sons, 2004. ISBN 9780470849163.

[11] Griva I., Nash S. G., Sofer A.: *Linear and Nonlinear Optimization*. George Mason University, Fairfax, Virginia, 2009, ISBN 978-0-898716-61-0.

[12] Ho M. W.: *Genetické inženýrství – naděje nebo hrozba?* Alternativa, Praha, 1998. ISBN 80-85993-52-X.

[13] Holešovský J., Popela P., Roupec J.*: On a Disruption in Congested Networks*. Proceedings of 19th International Conference of Soft Computing, MENDEL 2013, pp. 191–196, ISBN 978-80-214-4755-4, ISSN 1803-3814.

[14] Holland J. H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, Cambridge, Massachussetts, 1992. ISBN 0-262-58111-6.

[15] Hoyland K., Wallace S. W.: *Generating Scenario Trees for Mulstistage Problems*. Technical Report 4/97, Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, 1996.

[16] Hrabec D.: *Stochastic Programming for Engineering Design*. Master thesis, FME, Brno University of Technology, Brno, Czech Republic, 2011.

[17] Hrabec D., Popela P., Novotný J., Haugen K. K., Olstad A.: *The stochastic network design problem with pricing*. In Proceedings of the 18th International Conference of Soft Computing MENDEL 2012, pp. 416–421, 2012. ISBN 978-80-214-4540-6, ISSN 1803-3814.

[18] Hrabec D., Popela P., Roupec J., Jindra P., Haugen K. K., Novotný J., Olstad A.: *Hybrid Algorithm for Wait-And-See Network Design Problem*. In Proceedings of 20th International Conference of Soft Computing, MENDEL 2014, pp. 97–104, ISBN 978-80-214-4984-8, ISSN 1803-3814.

[19] Hrabec D., Popela P., Roupec J., Mazal J., Stodola P.: *Two-Stage Stochastic Programming for Transportation Network Design Problem*. In Mendel 2015: Recent Advances in Soft Computing. Advances in Intelligent Systems and Computing. 2015. pp. 17-25. ISBN: 978-3-319-19824- 8. ISSN: 2194- 5357.

[20] Cháb J.: *Hodnocení výkonnosti hierarchických genetických algoritmů*. Diplomová práce, VUT FSI Brno, 2003.

[21] Janíček P.*: Systémové pojetí vybraných oborů pro techniky. Hledání souvislostí*. CERM, Brno, 2007, ISBN 978-80-7204-554-9.

[22] Kall P., Wallace S. W.: *Stochastic Programming*. John Wiley and Sons, Chichester, 1994. ISSN 0160-5682.

[23] Kvasnička V.: *Umelá evolúcia*. Kognitívne vedy III. Bratislava, 2000. pp. 90–108.

[24] Kvasnička V., Pospíchal J., Tiňo P.: *Evolučné algoritmy*. STU Bratislava, 2000.

[25] Mak W. K., Morton D. P., Wood R. K.: *Monte Carlo Bounding Techniques for Deterministic Solution Quality in Stochastic Programs*. Operations Research Letters, 24:47, pp. 47–55, 1999.

[26] Ošmera P., Král J., Roupec J.: *Use of genetic algorithms in neural networks*. In journal Inženýrská mechanika (5): 45–51, 1993.

[27] Ošmera P., Roupec J.: *Influence of initialization of the first population on the quality of genetic algorithms*. In Proceedings of 11th International Conference on Process Control and Simulation, Košice (Slovakia), 1994.

[28] Ošmera P.: *Hybrid and Distributed Genetic Algorithms for Travelling Salesman Problem in LAN*. In Proceedings of Mendel '95, Brno, 1995. pp. 105–108.

[29] Ošmera P., Šimoník I., Roupec J.: *Multilevel distributed genetic algorithm*s. In Proceedings of the International Conference IEE/IEEE on Genetic Algorithms. Sheffield, 1995. pp. 505–510.

[30] Ošmera P., Kvasnička V., Pospíchal J.: *Genetic Algorithms with Diploid Chromosomes*. In Proceedings of Mendel'97, Brno, 1997, pp. 111–116.

[31] Ošmera P.: *Integrated Evolutionary Algorithms*. In Proceedings of Mendel 2000, Brno, Czech Republic, pp. 113–117, 2000.

[32] Ošmera P., Roupec J.: *Limited Lifetime Genetic Algorithms in Comparison with Sexual Reproduction Based GAs*. In Proceedings of MENDEL'2000, Brno, Czech Republic, pp. 118–126, 2000.

[33] Ošmera P., Roupec J., Matoušek R.: *Genetic Algorithms with Diploid Chromosomes and Sexual Reproduction*. In book: Sinčák, P. – Vaščák, J.: *Quo* Vadis *Computational Intelligence?* Physica – Verlag, A Springer Verlag Company, pp. 317–323, 2000. ISBN 978-3-7908-1322-7, ISSN 1867-5662.

[34] Ošmera P., Roupec J., Matoušek R.: *Genetic Algorithms with Sexual Reproduction*. In Proceedings of World Multiconference on Systemics, Cybernetics and Informatics SCI 2000, Orlando (Florida, USA), 2000, pp. 306–311. ISBN 9800766871.

[35] Palisa V.: *Velká neznámá aneb jde o život*. Eminent, Praha, 1996. ISBN: 80-85876-18-3.

[36] Plesník J., Dupačová J., Vlach M.: *Lineárne programovanie*. Alfa, Bratislava. 1990.

[37] Popela P., Dvořák J.: *Global Optimization and Genetic Algorithms*. In Proceedings of Mendel '96, Brno, 1996. pp. 194–197

[38] Popela P.*: An Object-Oriented Approach to Multistage Stochastic Programming: Models and Algorithms*. Phd Thesis, MFF UK Praha, 1998.

[39] Popela P.: *Application os Stochastic Programming in Foundry*. Folia Fac. Sci. Nat. Univ. Masarykianae Brunensis, Mathematica, 7:117–139, 1998.

[40] Popela P., Roupec J.: *GA-based Scenario Set Modification in Two-Stage Melt Control*. In Proceedings of Mendel '99, Brno, 1999. pp. 112–117

[41] Popela P.: *Stochastic Programming*. University of Malta, 2004.

[42] Popela P., Matoušek R., Sklenář J.: *The Formal Framework for DOP*. In Proceedings of the 14th International Conference on Soft Computing MENDEL '2008. Brno, pp. 235–240, 2008. ISBN 978-80-214-3675-6.

[43] Popela P., Sklenář J., Matoušek R., Žampachová E., Roupec J.: *Advances in the Formal Framework for DOP*. Proceedings of 17th International Conference of Soft Computing, MENDEL 2011, pp.320-325, ISBN 978-80-214-4302-0.

[44] Popela P., Sklenář J., Matoušek R., Roupec J., Mrázková E.: *Advanced Decomposition Techniques Applied to DOP*. Mendel Journal series, Vol.2012, (2012), No.1, pp.582-587, ISSN 1803-3814.

[45] Purves W. K., Sadava D. E., Orians G. H., Heller H. C.: *Life, The Science of Biology*. Sinauer Assoc. Inc., 2003. ISBN 978-0716798569.

[46] Roupec J., Krejsa J.: *Dominance and Recessivity in Genetic Algorithms*. In Proceedings of Mendel'96, Brno, 1996. pp. 197–199.

[47] Roupec J., Popela P., Ošmera P.: *The Additional Stopping Rule for Heuristic Algorithms*. In Proceedings of Mendel'97, Brno, 1997. pp. 135–139.

[48] Roupec J., Popela P., Ošmera P.: *Optimizing GA Lifetime Parameter Analyzing Error Probability Estimates*. In Proceedings of Mendel '2000, Brno, 2000. pp. 139–144.

[49] Roupec J.: *Vývoj genetického algoritmu pro optimalizaci parametrů fuzzy regulátorů*, PhD Thesis, Brno, 2002.

[50] Roupec J., Popela P.: *Scenario Generation and Analysis by Heuristic Algorithms*. In Proceedings of World Congress on Engineering and Computer Science WCECS 2007, San Francisco, USA, ISBN 978-988-98371-6-4, pp. 931–935.

[51] Roupec J., Popela P.: *Genetic Algorithms for Scenario Generation in Stochastic Programming: Motivation and General Framework*. In book S. L. Ao, B. B. Rieger, S. S. Chen (Eds.): Lecture Notes in Electrical Engineering, book series: Advances in Computational Algorithms and Data Analysis, Vol. 14, Springer Verlag, 2009, ISBN 978-1-4020-8918-3, pp.527–536

[52] Roupec J.: *Advanced Genetic Algorithms for Engineering Design Problems*. In journal Engineering Mechanics vol. 17 (2011), No. 5/6, ISSN 1802-1484, pp. 407– 417.

[53] Roupec J., Popela P.: *The Nested Genetic Algorithms for Distributed Optimization Problems.* Proceedings of The World Congress on Engineering and Computer Science 2011, San Francisco, USA, pp.480-484, ISBN 978-988-18210-9-6.

[54] Roupec J., Popela P., Hrabec D., Novotný J., Olstad A., Haugen K. K.: *Hybrid Algorithm for Network Design Problem with Uncertain Demands*. Proceedings of the World Congress on Engineering and Computer Science, 2013, Vol I, WCECS 2013, 23-25 October, 2013, San Francisco, USA, ISBN: 978-988-19252-3-7, ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[55] Ryan C.: *Shades. Polygenic Inheritance Scheme*. In Proceedings of Mendel'97, Brno, 1997, pp. 140–147

[56] Shapiro A., Dentcheva D., Ruszczynski A.: *Lectures on Stochastic Programming: Modeling and Theory*. SIAM, 2009. ISBN: 978-0-89871-687-0.

[57] Stevens W. R.: *TCP/IP Illustrated, Volume I: The Protocols*. Addison-Wesley, 2003. ISBN 0201633469.

[58] Studnička V.: *Objektové implementace genetických algoritmů*. Diplomová práce, VUT FSI, Brno, 2009.

[59] Šandera Č.: *Heuristic Algorithms in Optimization.* Diplomová práce, VUT FSI Brno, Università degli Studi dell'Aquila, 2008.

[60] Šandera Č., Popela P., Roupec J.: *The Worst Case Analysis by Heuristic Algorithms*. In Proceedings of the 15th International Conference on Soft Computing MENDEL '2009. Brno, 2009, ISBN 978-80-214-3884-2, pp.109-114.

[61] Šeda M.: *Využití moderních heuristických metod v rozvrhování.* Disertační práce, VUT FS Brno, 1998.

[62] Škrabal O.: *Genetické algoritmy a rozvrhování*. Diplomová práce, VUT FSI Brno, 2010.

[63] Trča S.: *Cesty k dlouhověkosti*. Avicenum, Praha, 1987.

[64] Vrtík P.: *Vliv počtu pohlaví na chování genetického algoritmu se sexuální reprodukcí.* Diplomová práce, VUT FSI Brno, 2001.

[65] *GAMS – A User's Guide* [online]. GAMS Development Corporation, Washington DC, USA, 2007. [cit. 15.1.2016]. Dostupné z: http://www.un.org/en/development/desa/policy/mdg_workshops/training_material/gams_users_guide.pdf

[66] *RFC 793, Transmission Control Protocol* [online]. Defense Advanced Research Projects Agency, USA, 1981 [cit. 30. 1. 2016]. Dostupné z https://tools.ietf.org/html/rfc793