VYSOKÉ UČENÍ TECHNICKÉ
Fakulta strojního inženýrství
Ústav matematiky

Ing. Čeněk Šandera

Hybridní model metaheuristických algoritmů

Hybrid model of metaheuristic algorithms

Zkrácená verze PhD Thesis

Obor:	Aplikovaná matematika
Školitel:	prof. RNDr. Ing. Miloš Šeda, Ph.D.
Oponentni:	
Datum obhajoby:	

Klíčová slova

metaheuristika, optimalizace, obecný model, hybridní algoritmy

Keywords

metaheuristic, optimization, general model, hybrid algorithms

Obsah

1 Úvod

V teoretických i aplikačních oborech se nezřídka vyskytují úlohy, jejichž řešení je s dnešními výpočetními možnostmi nedosažitelné. Takovéto úlohy bývají motivovány problémy vyskytujícími se napříč mnoha vědními oblastmi, a to od biologie přes ekonomii až po abstraktní matematiku. Nemožnost nalézat správná řešení není dána pouze nějakými chybějícími metodami, ale spíše samotnou podstatou složitosti úloh. Analýza jejich výpočetní náročnosti naznačuje, že existuje mnoho případů, kdy žádnou dostatečně efektivní metodu řešení vytvořit prostě nelze. V současné době se sice jedná pouze o domněnku, ale s přibývajícím časem stále ubývají důvody o ní příliš pochybovat.

Z akutní potřeby řešit tyto *neřešitelné* úlohy vznikají různé postupy k jejich zjednodušení. Jedním z nich je snížení nároků na přesnost výsledků a spolehnutí se převážně na experimentální přístup. Mimo jiné se mezi takové metody řadí i *metaheuristiky*, které tvoří nosné téma celé této disertace. Hlavní myšlenkou metaheuristik je systematicky prohledávat stavový prostor úlohy a ze získaných informací určovat svůj následující postup. Metaheuristiky bývají popsány velmi obecným schématem a jsou tak aplikovatelné na mnoho různých druhů problémů. Stejným způsobem se tak mohou řešit například spojité, diskrétní nebo i kombinatorické úlohy.

Kvůli své obecnosti, nemají metaheuristiky žádnou jednotnou definici a každý autor má velkou volnost v jejich návrhu. Pravidelně tak vznikají nové metody s nekompatibilní terminologií, kde nemusí být často ani jasné, v čem se vzájemně odlišují. Ze stejného důvodu se stává velmi komplikovaným i jejich vzájemné propojování a kombinování dílčích vlastností. Tato disertační práce popisuje obecný model, který takový jednotný popis umožňuje, a přesto autorům nových algoritmů zachovává velkou tvůrčí svobodu.

Disertační práce představuje metaheuristiky v širším kontextu, a proto jsou úvodní kapitoly věnovány popisu různých tříd úloh a hodnocení jejich složitosti. Uveden je zde i stručný přehled jejich možných způsobů řešení, a to včetně méně známých a alternativních cest. Po následném úvodu do problematiky metaheuristik je vypracováno obsáhlé shrnutí charakteristik jednotlivých algoritmů a na jeho základě je navržen obecný model. Celá práce je zakončena ukázkami témat autorem publikovaných metaheuristik, společně s diskuzí vztaženou k vytvořenému obecnému modelu.

2 Metaheuristiky

Metaheuristiky jsou třídou výpočetních metod používaných na optimalizační problémy, jejichž řešitelé nevyžadují nutně nalezení optimálního řešení. Pomocí metaheuristických algoritmů lze prohledávat stavový prostor úlohy, avšak bez záruky nalezení dostatečně kvalitního řešení v rozumném čase.

Slovo *metaheuristika* vzniklo spojením dvou původem řeckých slov *meta* a *heuristika*. Předpona *meta* se používá k označení přesahu kořenového pojmu a znamená *něco za* nebo přeneseně i *nad*. Termín *heuristika* (z řeckého *heuriskein* znamenající *objevovat* nebo *vynalézat*) označuje metody, které vznikly pozorováním a jsou ověřeny pouze zkušeností bez exaktní prokazatelnosti.

První použití výrazu *metaheuristika* se přisuzuje Fredovi Gloverovi v článku [1] z roku 1986, kde jím označuje nadvrstvu nad běžnou heuristikou[1]. Jako alternativní název pro stejné metody se občas používal i pojem *moderní heuristiky* (viz [2]), od konce 90.let se ale již téměř výhradně používá široce rozšířeného pojmu *metaheuristika*.

2.1 Definice

Odborná komunita se zatím jednoznačně neshodla na definitorickém vymezení pojmu *metaheuristika*, a to převážně z důvodu, že k vývoji nových metod nebyla zatím v podstatě potřeba. V literatuře lze nalézt několik existujících definic (nebo alespoň zobecněných popisů), které si v mnoha ohledech odpovídají a pokrývají podobné třídy algoritmů, v dílčích detailech se ale liší. Příkladem mohou být následující definice.

- [3]: *Metaheuristika je množina algoritmických konceptů, které mohou být využity k definování heuristických metod aplikovatelných na široké spektrum různých problémů. Jinak řečeno, metaheuristiky mohou být chápány jako zobecněné heuristické metody navržené k určování postupu problémově závislých heuristik (lokální prohledávání, konstrukční heuristiky) směrem k oblastem ve stavovém prostoru obsahující vysoce kvalitní řešení. Metaheuristika je tedy obecný algoritmický rámec aplikovatelný na mnoho různých problémů s relativně malým počtem úprav nutných pro přizpůsobení se konkrétnímu problému.*
- [4]: *Metaheuristika je proces iterativního generování, který řídí podřízenou heuristiku kombinováním rozdílných konceptů pro prohledávání a využívání stavového prostoru pomocí učících se strategií k strukturování informací pro efektivní nalezení téměř optimálního řešení.*
- [5]: *Metaheuristika je iterativní proces, který řídí a mění operace podřízených heuristik, aby efektivně vytvářely kvalitní řešení. Může manipulovat s úplným (nebo neúplným) jednotlivým řešením nebo s celou množinou řešení během každé iterace. Podřízené heuristiky*

[1] "Tabu search may be viewed as a 'meta-heuristic' superimposed on another heuristic."

mohou být vysoko (nebo nízko) úrovňové procedury, jednoduchá lokální vyhledávání a nebo jen konstrukční metody.

- [6]: *Metaheuristika je inteligentní iterativní proces, který vykonává prohledávání a může být aplikovaný na optimalizační problémy, jako např. problém obchodního cestujícího.*
- [7]: *Metaheuristický algoritmus může být definován jako vysokoúrovňová obecná metodologie (šablona), která je používána jako návod k odvození heuristik pro řešení specifických optimalizačních problémů.*
- Yang ve své knize [8] podotýká, že žádná obecně uznávaná definice zatím neexistuje a trendem poslední doby je pojmenovávat metaheuristikou jakýkoliv stochastický algoritmus s randomizací a lokálním prohledáváním, čehož se drží ve svých publikacích i on.

Tyto popisy ukazují, že autoři intuitivně chápou pojem *metaheuristika* obdobným způsobem a vyhýbají se příliš svazujícím definicím. Striktní definice je vhodná především pro matematickou práci, ale nemusí být vhodná pro experimentální tvorbu nových algoritmů.

Hlavními problémy uvedených popisů je buď přílišná obecnost, nebo naopak vyžadování příliš konkrétní vlastnosti. Metaheuristika ve skutečnosti nemusí být inteligentní[2], samoučící se[3] a nemusí být ani nutně stochastická[4]. Tyto silné předpoklady mohou být z definice vypuštěny, protože zbytečně zužují třídu využitelných metod.

Vyjasnění pojmu

Napříč publikovanou literaturou je pozorovatelná nekonzistence nejen v přesné definici metaheuristiky, ale i v jejím obecnějším pojetí. Tímto pojmem se označují jak vysokoúrovňové koncepty, které dávají jenom velmi zevrubný návod na řešení konkrétních úloh, tak i kompletní algoritmy čekající jen na povel ke spuštění. Mnoho metaheuristik je inspirováno nějakým specifickým přírodním (biologickým) jevem a jsou popsány pouze pomocí terminologie a jevů pozorovaných ve studovaném systému. Jako metaheuristika jsou označovány i metody přesně definované matematickým popisem a fungující na úlohách s definovanými vlastnostmi. Stejně tak lze často číst věty typu "*Vytvořená metaheuristika nalezla řešení našeho problému za 28 sekund ...*", kde je zjevně za metaheuristiku označován spuštěný algoritmus na konkrétní instanci úlohy.

Pokud vyjdeme z významu pojmu *heuristika*, tak její spojení s předponou *meta* by mělo znamenat, že *metaheuristika* je heuristika vytvářející heuristiky. Ani pojem *heuristika* nemá ale svou přesnou definici, a proto nemůže mít přesnou definici ani jeho odvozený pojem. Obecně

[2] *Inteligence* je stejně problematický nedefinovatelný pojem.

[3] *Samoučící* je pojem vzbuzující přílišná očekávání a kladoucí vysoké nároky na práci s informacemi.

[4] Z principu není nutné, aby se algoritmus choval při každém běhu různě.

lze ale chápat heuristiku jako smysluplný návod na řešení konkrétní úlohy. Metaheuristika je tedy smysluplný návod na vytváření smysluplných návodů pro řešení konkrétních typů úloh. Nevyjasněnou otázkou ovšem zůstává význam pojmu *konkrétní typ úlohy*. Míra konkretizace není jednoznačná, protože heuristika může dávat slušné výsledky pouze pro jednu konkrétní instanci problému, nebo zvládne řešit instance úlohy s podobnými vlastnostmi (podobné hodnoty parametrů), popřípadě může být i schopna řešit všechny možné instance konkrétní úlohy. Vztáhneme-li tuto úvahu na *metaheuristiky*, není jasné, jestli má metaheuristika poskytovat návody na vytváření heuristik pro úlohy typu *kvadratická optimalizace*, *konvexní optimalizace*, *spojitá optimalizace* nebo *optimalizace obecného zobrazení* $f : X \to Y$.[5] Uvedené příklady jsou postupně svými podtřídami a pro definici metaheuristiky by musela být dána jasná hranice, která by určila, co je příliš obecná úloha pro heuristiku a co naopak příliš konkrétní pro metaheuristiku. Je jasné, že na takové hranici by se odborná komunita těžko jednoznačně shodovala a navíc by ve své podstatě ani nebyla nijak významně užitečná (vzhledem k základní myšlence heuristik, tedy že se exaktně nemusí dokazovat).

Z této rozvahy ale může plynout i rozumnější následující závěr. **Metody, které dokáží přímo řešit libovolnou třídu optimalizačních úloh, jsou označovány jako heuristiky, kdežto pokud je potřeba metody nějak funkčně upravovat, jedná se o metaheuristiky**. *Přímým řešením* je myšlen jasně definovaný funkční koncept, který se dá měnit pouze numerickými (nebo jinak přesně definovanými) parametry. Metaheuristiky ale dávají obecnější popis, takže k přímému řešení musí být ještě dodefinovány jednotlivé funkční bloky (specifické operátory pro daný typ úlohy - např. spojitá vs. kombinatorická optimalizace). Tím je ve výsledku zajištěno, že metaheuristika generuje příslušné heuristiky.

Metaheuristiky tedy mohou být vytvářeny pro široké spektrum úloh, stejně jako pro úzce specializované problémy. Z praktického pohledu je ale vhodnější vytvářet metaheuristiky pro co nejobecnější třídy a klást na ně co nejméně dodatečných podmínek. Metaheuristiky mají často stochastické rysy, ale nejsou tím nijak omezeny. Kvalitní metaheuristiky mohou být jak stochastické tak i deterministické povahy. (Meta)heuristiky obecně nezaručují žádnou kvalitu řešení, ale nic nebrání jejich exaktnímu prozkoumání a stanovení potřebných vlastností. Metaheuristika zůstane metaheuristikou, i když bude dokázána její případná konvergence. Speciálně u NP-těžkých a NP-úplných problémů lze konstruovat metaheuristiky, které při dostatečně dlouhém čase projdou celý stavový prostor, a řešení tedy zaručeně naleznou. Heuristický přístup je ale v tomto případě zakotven v pořadí prohledávaných bodů, protože není v silách žádného dnešního výpočetního zařízení projít celý stavový prostor v rozumném čase. Po uplynutí stanovené doby jsou i konvergentní heuristiky přerušeny a jako výsledek reportují pouze nejlepší nalezené řešení.

Uvedený výklad je založen na porozumění pojmu *metaheuristika* v obecné a jazykovědné rovině. Zvolené pojmenování ale v reálném světě nemusí nutně přesně odpovídat typu metod,

[5] Podrobnější popis různých typů optimaliazce lze nalézt v nezkrácené verzi disertační práce.

které jsou pod něj zařazovány. Jak již bylo řečeno, rozebírané metody se nějakou dobu nazývaly i *moderní heuristiky* a pojem *metaheuristika* byl vytvořen v brzkých počátcích výzkumu, kdy ještě nebyl příliš jasně vytyčen směr rozvoje celého oboru. Mohlo by se tak zdát vhodnější změnit název na nějaký více jednoznačný, ale v dnešní době je již originální termín tak zaužívaný, že by jakákoliv jeho změna neměla žádnou šanci na prosazení.

V literatuře jsou ale všechny zmíněné koncepty různě zaměňovány a explicitně mezi nimi není rozlišováno. Většinou to nesnižuje srozumitelnost textu, protože je z kontextu zjevné, jestli se jedná o metaheuristiku (podle výše uvedeného popisu) a nebo heuristiku (algoritmus vygenerovaný na základě nějaké metaheuristiky). Tohoto přístupu bude použito i v této disertaci, pokud nebude vyloženě potřeba rozlišovat mezi těmito pojmy, tak budou oba souhrnně a zjednodušeně označovány za metaheuristiky.

2.2 Specializace metaheuristik pro optimalizační úlohy

V kontextu softcomputingu a optimalizačních metod je *metaheuristika* často chápána jako návod jak zkonstruovat algoritmus generující body stavového prostoru pouze na základě předchozích získaných znalostí. Při obecném pohledu na optimalizovanou funkci $f : X \to Y$ to znamená, že pomocí metaheuristiky lze vytvářet algoritmy (heuristiky), které pro danou úlohu dokáží generovat posloupnost bodů $(x, f(x))$ tak, aby se hodnoty $f(x)$ s časem zlepšovaly (ve smyslu optimalizačního kritéria).

Jako základní předpoklad pro úspěšný běh metaheuristiky tedy musí být zajištěno, že nalezené dvojice hodnot $(x, f(x))$ spolu navzájem *nějak korelují*. Taková korelace nemusí existovat globálně na celém stavovém prostoru, ale stačí, když určitý vztah mají mezi sebou body pouze na nějakém lokálním okolí. V případě, že by neexistovala žádná závislost mezi polohou bodu x ve stavovém prostoru a hodnotou jeho účelové funkce $f(x)$, neměla by se metaheuristika podle čeho rozhodovat a generování nových bodů by tedy probíhalo zcela náhodně.

Metaheuristiky tedy obecně nepotřebují žádné apriorní předpoklady o vlastnostech účelové funkce, protože jejich základní schopností je pohyb v neznámém prostoru a vyhledávání tamních nejlepších hodnot. Zjednodušeně lze chování metaheuristik přirovnat k hledání nejvyšší hory v neznámé krajině za naprosté tmy. Algoritmus musí o dalších krocích rozhodovat pouze ze znalosti nadmořských výšek v předchozích navštívených místech. Je zřejmé, že pokud je ráz krajiny předem znám, je větší šance na nalezení efektivnějšího způsobu prohledávání. Některé metaheuristiky tedy mohou pracovat pro určitý typ úloh výrazně lépe než pro jiné, ale dopředu nemusí být nutně úplně jasné, které typy jsou pro danou metaheuristiku vhodné a které nikoliv. A jelikož je vysoce nad schopnosti každého *průzkumníka* pamatovat si všechna navštívená místa, musí být získané informace nějakým způsobem navíc postupně zpracovávány a nepotřebné odstraňovány.

2.3 Druhy metaheuristických algoritmů

Jelikož metaheuristiky ani heuristiky nejsou exaktně odvozovány ze základních principů, bývá jejich činnost založena na inspiraci známými procesy. Zdaleka nejčastějším zdrojem inspirace při vývoji nových metaheuristik je živá příroda. Autoři těchto algoritmů se odvolávají na evoluční principy, na genetické zákonitosti, na chování mravenců, ptáků, včel i netopýrů. Metaheuristiky lze tedy rozlišovat na základě druhu inspirace, ale toto rozdělení nepřináší kvalitativní rozdělení z pohledu chování algoritmů. Podstatnějším znakem z hlediska výkonnosti a chování algoritmu je, jestli se jedná o *populační* nebo *jednoduché* prohledávání. Při jednoduchém (*single based*) je v paměti udrženo pouze jedno hlavní řešení, kdežto u populačních algoritmů je jich udržována celá skupina a nová řešení jsou generována na základě její kombinace. Velmi významnými jsou i velikosti oblastí, které jsou prohledávány. Při *lokálních* typech algoritmů jsou nová řešení generována v blízkosti původních, naopak při *globálních* metodách je možné nové řešení generovat od původní skupiny velmi daleko. Další důležité rozdíly mezi metaheuristikami jsou shrnuty v následující kapitole.

Pro zlepšení výkonu metaheuristiky na konkrétních typech úloh se zkouší mnoho různých postupů. Kromě základního nastavování řídicích parametrů a jednoduchých úprav dílčích částí metaheuristiky lze použít i metody pro kombinování různých metaheuristických konceptů dohromady. Příkladem jsou tzv. *paralelní metaheuristiky*, kde se spustí více metaheuristik pro stejný problém současně a v určitých okamžicích si vyměňují informace. Tyto spolupracující metaheuristiky mohou být stejných typů, ale lze spolu kombinovat i jejich různé varianty, nebo dokonce zcela odlišné typy. Pokud jsou spuštěny dvě instance, existuje pouze jeden způsob jejich vzájemného propojení (z topologického pohledu). Pokud jich je ale propojeno více dohromady, potom mohou vznikat různé topologie s odlišnými vlivy na celkové chování algoritmu. Šíře variability topologií počíná výměnou informací každého dílčího celku s každým přes hierarchické pospojování až po zcela obecné nebo náhodné spojení. Jednotlivé samostatné metaheuristiky se často nazývají *ostrovy*, čímž se zdůrazňuje nezávislost jejich prohledávání na ostatních dílčích celcích. Ostrovy si mezi sebou mohou vyměňovat jak nejlepší nalezená řešení, tak i např. náhodné skupinky řešení. Vhodným propojením několika metaheuristik se tady nedocílí pouze urychlení výpočtu a větší šíře prohledávané oblasti, ale vznikají tak zcela nové vzorce chování, které mohou napomoci k efektivnějšímu řešení zadaných úloh (více viz [9]).

Druhým významným způsobem kombinování různých metaheuristik je tzv. *hybridizace*. Jedná se o spojení několika různých konceptů (nebo jen jejich dílčích částí) do jednoho funkčního celku.[6] Hybridizace metaheuristik nemusí probíhat pouze spojením s odlišnou metaheuristikou, ale může být založena i na spojení s naprosto odlišným konceptem. Často jsou metaheuristiky spojovány s exaktními algoritmy, jako je například lineární nebo dynamické programování, ale takový způsob hybridizace vyžaduje apriorní znalost typu řešené úlohy a není tedy problémově nezávislý. Algoritmy tohoto typu jsou často velice efektivní, díky čemuž získávají značnou

[6] Výše zmíněnou paralelizaci lze podle uvedeného popisu považovat za speciální případ hybridizace.

popularitu. Jejich oblibu dokazuje nejen fakt, že jim jsou věnovány samostatné konference, ale získaly dokonce i vlastní pojmenování *matheuristiky* [10]. Hybridizace je často prováděna i s dalšími nástroji soft computing, jako jsou neuronové sítě nebo fuzzy matematika. Tyto a další nástroje z nejrůznějších oblastí výzkumu (umělá inteligence, strojové učení, operační výzkum, statistika, ...) nahrazují nebo rozšiřují některé ze základních částí běžných metaheuristik, popř. mohou sloužit i pro řízení algoritmu nebo komunikaci dílčích podčástí.

Hybridizace se často rozlišuje podle úrovně spolupráce na tzv. *slabou* a *silnou* v závislosti na zachované integritě. Pokud kombinací několika konceptů vznikne úplně nový algoritmus, ve kterém lze spatřit už jen náznaky původních konceptů, je hybridizace považována za silnou. Naopak slabou hybridizaci lze dosáhnout spíše vnějším propojením základních konceptů, kde celistvost původních algoritmů zůstane zachována v nezměněné podobě. Protože hybridizace je z principu velmi variabilní koncept, bylo vyvinuto několik způsobů klasifikace takto vzniklých algoritmů. Jednotný pohled na hybridizaci a její rozdělení do tříd lze nalézt v přehledném shrnutí v [11]. Hybridizace je jedno z nosných témat této disertační práce a v následujících kapitolách jí je věnován ještě další prostor.

Kromě značné variability v koncepčních přístupech jednotlivých metaheuristik, lze mnoho odlišností spatřit i v implementačních detailech. Nejvýznamnějším důvodem rozdílností při implementaci je odlišná architektura výpočetních zařízení poskytující metaheuristikám běhové prostředí. Zdaleka nejběžnější je spouštění metaheursitky na klasickém *procesoru stolního počítače*. Procesor zde sériově vykonává definované instrukce a podle potřeby zapisuje nebo čte z přístupné paměti. Výrobci pravidelně zvyšovali výkon procesorů a nebyl tedy důvod k významným obměnám základních metaheuristických principů. Posledních pár let ale zvyšování výkonu procesorů významně zpomalilo a místo toho se začaly prosazovat vícejádrové procesory. Každý takový procesor zvládá několik paralelních výpočtů zároveň a znatelně tak může urychlit průběh výpočtu. Metaheuristiky jsou ze své povahy velice dobře hardwarově paralelizovatelné na úrovni vyhodnocování účelové funkce. Jádra vícejádrových procesorů ale většinou nemají možnost nezávisle přistupovat k paměti, a proto musí být metaheuristiky pečlivě navrhovány tak, aby se jádra vzájemně neomezovala a aby se předešlo možným bezpečnostním problémům [12]. Jiným směrem se rozhodli jít výrobci grafických karet, kde je mutliprocesorový přístup základním rysem jejich činnosti. Grafické karty vykonávají desítky až stovky identických úkonů současně a jejich celkový výkon se rychle začal vyrovnávat běžným procesorům. Největší výrobci začali ke svým kartám poskytovat vývojové nástroje[7], které dovolovaly jejich potenciál naplno využít. Architektura grafických karet má ale mnoho omezení a vyvíjené metaheuristiky se jim musely silně přizpůsobovat[8]. Při úspěšném návrhu byly ale výsledky aplikovaných metaheuristik na grafických kartách velice slibné [13].

[7] Nvidia vytvořila nový nástroj CUDA a firma AMD nástroj OpenCL.

[8] Grafické karty používaly pouze jednoduchou přesnost čísel a všechny procesory musely ve stejný okamžik vykonávat naprosto stejnou instrukci.

Metaheuristiky nemusí být spouštěny pouze na jediném zařízení, ale jsou velmi vhodné i pro současný běh na více počítačích zároveň. Velmi efektivní je *distribuované řešení*, kde je propojeno libovolné množství nezávislých počítačů komunikujících podle stanoveného protokolu. Na každém počítači tak může být spuštěna samostatná metaheuristika, nebo může být využit pouze pro dílčí části výpočtu. Spojené počítače nemusí mít stejný výkon ani stejný operační systém, pouze musí vědět, kdy, kam a co zaslat. Příkladem je opensource systém Boinc, který umožňuje propojení počítačů přes internet a díky kterému lze spouštět metaheuristiky na stovkách ([14]) či tisících počítačích zároveň.

Velmi moderním trendem v informačních systémech je tzv. *cloud computing*. Podobně jako u distribuovaných výpočtů je i zde propojeno velké množství nezávislých počítačů, avšak s tím rozdílem, že každý pouze shromažďuje a zpřístupňuje data. Řídicí logika je postavena na základním přístupu odvozeného z funkcionálního programování a pojmenovaného *MapReduce*[9]. Pouze pomocí dvou základních funkcí `map` a `reduce`, lze velice efektivně implementovat řadu náročných algoritmů. Jedny z hlavních výhod jsou vysoká škálovatelnost algoritmu a rychlé zpracování enormních objemů dat [15]. Úspěšně implementovány byly tímto přístupem např. genetické algoritmy [16], diferenciální evoluce [17] a další.

Jak je z uvedeného souhrnu zřejmé, ke zvýšení výkonu metaheuristik se lze dostat několika odlišnými cestami. Od vývoje nových základních konceptů přes jejich různé kombinace a hybridizace až po využívání nejmodernějších hardwarových nástrojů. V některých případech se lze obejít i bez klasického počítače a vytvořit si vlastní hardware přímo na míru[10] nebo lze pro vyhodnocování účelové funkce využít přímo i lidský mozek (avšak ne pro zvýšení rychlosti výpočtu, ale spíše pro jeho zatím nezastupitelné schopnosti v rozpoznávání a klasifikaci obrazových dat)[11]. Hlavním pilířem všech uvedených tříd jsou ale základní metaheuristické koncepty a v následujících kapitolách bude několik takových typů představeno a následně i zobecněno.

[9] Přístup byl vyvinut firmou Google ke zpracování obrovských objemů dat.

[10] Genetický algoritmus byl realizován na FPGA, tedy na programovatelných klopných obvodech [18]

[11] Např. projekt *picbreeder.org* provozuje metaheuristiku vyvíjející obrazy, kde je jejich průběžné ohodnocování přenecháno na lidské úvaze.

3 Metaheuristické algoritmy

Metaheuristiky čerpají inspiraci z mnoha odlišných vědeckých oblastí. Běžná nejsou jen propojení s relativně blízkými odbory, jako je matematika, statistika nebo umělá inteligence, ale velmi často jsou základní koncepty odvozeny z oblastí, které u aplikované informatiky působí velmi exoticky. Typičtí zástupci dnešních metaheuristik jsou založeni buď na jednoduchých intuitivních pravidlech pohybu v neznámém prostoru (Hill climbing, Tabu search, ...), nebo na inspiraci přírodními fenomény, jako je evoluce, genetika (genetické algoritmy) nebo pohyb zvířat (Ant colony optimization, ...). Poslední jmenovaný přístup má mezi novými metaheuristikami výsadní postavení získané především atraktivností vlastního pojmenování než reálným inovativním přínosem. Následující odstavce názorně demonstrují šíři využité inspirace v publikovaných metaheuristikách.

Bacterial Foraging Optimization Algorithm [19] je algoritmus založený na chování bakterie Escherichia-coli. Hlavní stavební kameny jejího chování jsou odvozeny z chemotaxe[12], rozmnožování a obecného hejnového chování bakterií. *Bat-Inspired Algorithm* [20] simuluje schopnosti netopýrů, kteří loví hmyz pomocí echolokace. Pohyb netopýrů je ovlivněn vzdáleností od potravy, která zde zastupuje kvalitu řešení účelové funkce. Na principu echolokací je postaven i algoritmus napodobující delfíny při lovu [21]. Obecnějším přístupem zvířat (lvů, vlků, ...) k lovu je inspirován algoritmus *Hunting search* publikovaný v [22]. Metaheuristika *Cuckoo search* [23] je inspirována kukačkami a jejich zvykem klást svá vejce do hnízd jiných ptáků. Mezi další živočišné druhy, které posloužily k inspiraci metaheuristik, patří například kočky [24], lososi [25], opice [26], orli [27], ...

Chování včelích rojů je významnou inspirací pro více různých metaheuristik zároveň. *Honey bee mating optimization* [28] napodobuje chování včelí královny při páření a následném kladení vajíček. Včelí královna si zde vybírá z velkého množství trubců, jejichž genofond kombinuje se svým. *Artificial bee colony algorithm* [29] naopak napodobuje chování včel při hledání nektaru na loukách a každá včela reprezentuje jedno řešení úlohy. Běžné včely spolu komunikují pomocí signál, předávaných v úlu, což je u metaheuristiky nahrazeno následováním včely s nejlepší hodnotou účelové funkce. Metaheuristiky založené na chování včel jsou jedny z velmi populárních, a proto vzniklo mnoho jejich dalších druhů a aplikací (viz [30]).

Bumblebees algorithm [31] je založený na podobném konceptu jako Angels & Mortals, kde se jednotlivé postavy pohybují po toroidu a vzájemně se ovlivňují. Tento algoritmus obdobně využívá čmeláky, kteří na toroidu hledají květiny. Nejedná se o klasický hejnový algoritmus, ale spíše o sofistikovanější přístup k náhodnému prohledávání a udržování různých řešení v paměti. Čmeláci byli inspirací i pro algoritmus *Bumble Bees Mating Optimization* [32], který ke konstrukci nových řešení využívá speciálně upravený hladový algoritmus. Hmyz obecně patří mezi

[12] Pohyb organismu či buňky ve směru chemického gradientu.

velmi časté objekty využívané v metaheuristické inspiraci, protože se v přírodě musel naučit kooperovat ve velkých počtech a vykazuje tak vysoký stupeň samoorganizace. Další příklady mohou být termiti (*Termite colony optimization* [33]), červi (*Glowworm swarm optimization* [34]) nebo i octomilky (*Fly optimization Algorithm* [35]).

Zajímavým zdrojem inspirace je i astrofyzika a mechanika. Na pohybu vesmírných těles způsobených gravitací jsou založeny algoritmy *Gravitational algorithm* [36] a *Central force optimization* [37], kde hlavní úlohu hrají přitažlivé síly mezi jednotlivými vesmírnými objekty reprezentující řešení ve stavovém prostoru. Podobně i algoritmy *Big Bang-Big Crunch*[38] a *Big Crunch* [39] využívají principů gravitace a iterativně přitahují a odpuzují jednotlivá tělesa. Stejně byl postaven i algoritmus *Galaxy-based Search Algorithm* [40], kde jsou hlavními objekty celé rotující galaxie a algoritmus *Black hole* využívající fyziku černých děr [41].

Z dalších fyzikálních jevů byly k vybudování nových metaheuristik využity principy elektromagnetismu (*Electro-magnetism optimization* [42]), elektrostatického náboje (*Charged system search* [43]), deterministického chaosu [44] nebo i kvantové fyziky používané k obohacení genetických algoritmů [45] či simulovaného žíhání [46]. Pro přehledný souhrn dalších fyzikálně inspirovaných metaheuristik lze doporučit stručnou publikaci [47].

Využity byly i procesy formování přírody a procesy zajišťující životní koloběh. Způsob, jakým voda stéká po hornatém terénu a tvoří řeky a potoky, se používá v metaheuristice *River Formation Algorithm* [48], tvorba mraků je základem *Atmosphere Clouds Model Optimization Algorithm* [49] a celý vodní koloběh je zahrnut v *Water cycle algorithm* [50]. *Biogeography-based optimization* [51] čerpá inspiraci z ostrovní biogeografie a za základní princip používá myšlenku, že rychlost změny počtu živočišných druhů na ostrově je výrazně závislá na rovnováze mezi počtem imigrujících a emigrujících druhů. Stejně jak živočichové, tak i rostliny daly svým chování vzniknout několika metaheuristikám, jmenovitě to byl princip opylování v *Flower Pollination Algorithm* [52] nebo růst trávy v *Weed colonization Algorithm* [53].

Mimo přírodních procesů se využívá i konceptů pozorovaných v lidském chování a sociální interakci. *Cultural algorithms* [54] rozšiřují koncept genetických algoritmů o tzv. *belief space*, který symbolizuje předávání znalostí mezi generacemi. Vývoj lidské kultury je pozvolnější a není přímo závislý na změnách v genetickém vývoji populace, vzájemně se ale oba vývoje ovlivňují. Belief space je rozdělen na pět samostatných prostorů, které reprezentují určité oblasti znalostí a jejich řízené propojení je následně využíváno k tvorbě nových řešení. Podobně i *Imperialist competitive algorithm* [55] je považován za obdobu genetických algoritmů v sociální oblasti, tedy že se jedná o sociální evoluci populace. Sociálním postavením jedince je inspirován algoritmus *Social emotional optimization* [56], kde každé řešení symbolizuje konkrétního člověka, jehož cílem je dosáhnout co nejvyššího postavení ve společnosti. Do stejné oblasti se řadí i algoritmus *League championship algorithm* [57] simulující taktiku sportovních klubů ve snaze dosáhnout co nejlepšího rozložení sil při soutěžním šampionátu.

Vývoj nových algoritmů pokračuje ale i bez přímé inspirace reálnými procesy, a to za pomoci

konceptů založených spíše na informatické nebo matematické představě o fungování metaheuristik. *Nested partition algorithm* [58] průběžně klasifikuje části stavového prostoru podle pravděpodobnosti výskytu kvalitních řešení a následně podle toho generuje nová řešení z nejslibnějších oblastí. *Scatter Search* [59] vybírá z aktuální populace reálných řešení vhodnou podskupinu a různé dvojice jejich prvků následně lineárně interpoluje. Prvky takové vhodné podskupiny nemusí být jen nejlepšími prvky z pohledu hodnot účelové funkce, ale i například prvky zajišťující udržení dostatečné variability. Lineární kombinace dvou vybraných prvků mohou být chápany jako body tvořící jejich spojující cestu. Touto myšlenkou se řídí i metaheuristika *Path relinking* [59], která zmíněný Scatter search zobecňuje i pro nespojité proměnné. Pro každou vybranou dvojici řešení se nalezne cesta postupně měnící jedno řešení na druhé a jako nově vyprodukované řešení se použije nějaký její bod. Jelikož takových spojovacích cest může existovat více, uvažuje se pouze cesta s nejlepšími hodnotami účelové funkce.

Metaheuristika *Variable Neighborhood Search* [60] je příkladem rozšíření klasického lokálního prohledávání rozšiřující základní koncept o vlastní přístup k útěku z lokálního optima. V případě, že v okolí není žádný bod s lepší hodnotou účelové funkce než má aktuální pozice, je způsob generování tohoto okolí změněn. Pokud se ani v nově vytvořeném okolí (např. okolí s větším průměrem) nenachází žádný lepší bod, tak se buď zvolí další způsob generování okolí, nebo se algoritmus prostě ukončí. *Iterated local search* [61] volí odlišnou strategii pro vyváznutí z lokálního optima. Jakmile v okolí už nelze nalézt žádné lepší řešení, je aktuální bod náhodně pozměněn, a to natolik významně, že by se při následném pokračování v lokálním prohledávání nemělo vrátit zpět do stejného optima. Jedná se v podstatě o obdobu úplného restartu algoritmu, ovšem s tou výjimkou, že jsou zachovány všechny získané znalosti i celá paměť (např. tabu list). Lokální prohledávání je využíváno i metaheuristikou *Memetic Algorithms* [62], kde je standardní koncept evolučních algoritmů prokládán zlepšováním vybraných řešení pomocí lokálních (meta)heuristik. Tento přístup může být také interpretován jako postupné zlepšování jedinců v průběhu jejich života (učení se lepšímu způsobu přežití) a nespoléhání se pouze na sílu evolučních principů. Mezi další samostatné rozšíření lokálního prohledávání lze zařadit i *Reactive search optimization* [63], které zajišťuje adaptivní přizpůsobování parametrů metaheuristiky za jejího běhu. Typickým příkladem je přizpůsobování délky tabu listu v závislosti na předpokládané fázi algoritmu[13].

3.1 Co z toho plyne?

Přístupy jednotlivých metaheuristik se od sebe mohou velmi lišit. Neexistuje jediné obecné schéma, které obsáhne všechny popsané možnosti a přitom popisuje pouze a jen právě metaheuristiky. Rozdíly jsou patrné již v samotných základech algoritmů, tedy ve způsobu reprezentace

[13] Na začátku algoritmu stačí krátká paměť zajišťující rychlý výpočet, kdežto v blízkosti lokálních optim je potřeba tabu list s větším počtem řešení, aby nedošlo k zacyklení

řešení nebo v určování jejich kvality. Následně se liší i způsoby odvozování nových řešení a předávání informací získaných předchozím prohledáváním. Ne všechny metaheuristiky jsou určeny ke stejnému účelu (lokální vs. globální apod.), což obecné uchopení metaheuristik samozřejmě ztěžuje. Nejzákladnější princip metaheuristik je ale stále stejný, a to vyzkoušet velké množství různých řešení a vybrat z nich to nejlepší. Hlavním rozdílem oproti zcela náhodnému prohledávání je, že metaheuristiky implicitně předpokládají jistou souvislost mezi reprezentací řešení a hodnotou jeho účelové funkce. Většina odlišností tedy pramení z rozdílných předpokladů o struktuře účelové funkce a o významu jednotlivých získaných informací v průběhu prohledávání.

Z implementačního pohledu jsou většinou metaheuristiky chápány jako modifikující se množina řešení. Mezi klíčové charakteristiky jednotlivých přístupů, které definují rozdíly mezi jednotlivými koncepty, patří

- velikosti množiny řešení,
- doba, po kterou může jedno řešení přímo odvozovat nová řešení,
- počet prvků (rodičů) přímo podílejících se na produkci nových řešení,
- způsob uchovávání získaných informací,
- způsob předávání uchovaných informací,
- druh stochasticity,
- počet generovaných řešení

a několik dalších v závislosti na obecnosti pojmu metaheuristika.

Na základě svého konceptu trpí každá metaheuristika i svými nedostatky. Některé algoritmy vznikají pouze jako vylepšení stávajících metod k překonání jejich problémů, avšak z *No Free Lunch teorému* plyne, že nikdy nemůže překonat všechny své nedostatky[14]. Uživatelé metaheuristik si proto musí být vědomi jejich omezení a neočekávat nesplnitelné. Jako nejčastější problémy jsou uváděny předčasná konvergence[15], nedostatečná šíře prohledávané oblasti, možnost generování stále stejných řešení nebo přílišná variabilita a neschopnost se ustálit. Specifické problémy mohou mít i metaheuristiky pouze s určitými typy účelových funkcí. Například algoritmus *Differential evolution* má takové problémy se zašumělými funkcemi, kdežto konvenčním evolučním algoritmům tolik obtíží nezpůsobují [64]. Vhodnou modifikací ale lze diferenciální evoluci obměnit a schéma prohledávání přizpůsobit i přidanému šumu (viz [65]). Metaheuristika musí být vždy poplatná svému účelu, který může být pro různé uživatele různý, a proto bude v další kapitole navržen obecný model dovolující kombinovat různé metaheuristické přístupy a ověřovat jejich podstatné vlastnosti.

---

[14] Podrobnější vysvětlení lze nalézt v nezkrácené verzi disertační práce

[15] Příliš rychlá konvergence do lokálního extrému bez možnosti dále prohledávat stavový prostor.

4 Zobecněný model metaheuristik

Hlavním účelem metaheuristik je zprostředkovávat prohledávání stavového prostoru, a to pouze na základě hodnot účelových funkcí v prozkoumaných bodech. V předchozí kapitole bylo popsáno několik speciálních případů metaheuristik, jejichž principy mohou být vzájemně velmi významně odlišné. I přes velkou různorodost ale lze vypozorovat několik společných vzorců chování a využít je tak pro formulování obecného modelu.

Základním stavebním kamenem jakéhokoliv metaheuristického modelu musí být práce s dvojicemi typu (x, y), kde x je navštívený bod a $y = f(x)$ je zjištěná hodnota optimalizačního kritéria (účelové funkce). Z takto uspořádaných informací musí být metaheuristika schopna generovat nové body x, které mají *jistou šanci*[16] na lepší hodnotu y. Reálná výpočetní zařízení jsou navíc omezena limitovanou pamětí, a proto musí metaheuristika neustále rozhodovat, které páry (x, y) z paměti odstraní a které si ponechá pro další použití. Základními schopnostmi metaheuristických algoritmů tedy musí být

- schopnost vyhodnotit kvalitu konkrétního bodu x (tzv. vyhodnotit hodnotu účelové funkce),
- schopnost uchovat nebo zpracovat získané informace (dvojice (x, y)),
- schopnost rozhodnout o odstranění získaných informací z paměti,
- schopnost vygenerovat nová řešení na základě uchovaných informací.

Každá metaheuristika definuje vlastní způsoby zajišťování uvedených schopností. Při jejich jednotném popisu pomocí obecného modelu je lze snadno vzájemně kombinovat a využívat pro vytváření nových odvozených metaheuristik. Takové spojení několika principů různých metaheuristik dohromady bude dále nazýváno **hybridizací**.

Obecné modely

Myšlenka zobecnění existujících metaheuristik pod jeden koncept není nová a v odborných publikacích se tak již takové modely nepravidelně objevují řadu let. Například v [66] je vyvinut obecný popis určený primárně k lokálnímu prohledávání. Mimo základních verzí *Hill climbing* algoritmu, *Tabu search*, *Variable neighbourhood* algoritmu apod. jím lze popsat i genetické algoritmy a některé další populační metaheuristiky. Naopak model publikovaný v [67] se zaměřuje především na evoluční metaheuristiky, pod něž lze řadit jak genetické algoritmy, tak i *Ant colony optimization* nebo *Scatter search*. Mezi obecné modely, které si ale získaly větší pozornost patří výsledky publikované v [68] a popisující metaheuristiky následujícím pseudokódem

[16] Právě tento pojem je hlavním rysem metaheuristik - zlepšení hodnoty účelové funkce je negarantované a stojí pouze na heuristickém ověření.

```
inicializuj P pomocí externí procedury
while termination = FALSE do
    S <- OF(P)
    if |S|>1 then
        S' <- SCM(S)
    else
        S' <- S
    S'' <- IM(S')
    P <- IF(S'')
použij post-optimalizační proceduru na P.
```

Množina `P` je zde označována jako *Pool*, tedy množina jedinců reprezentující samostatná řešení, `IF/OF` jsou *input/output function* a zajišťují vstupy/výstupy z/do poolu. `SCM` je *Solution Combination Method*, která je použita pouze v případě, že je pro produkci nového řešení vybráno z poolu více než jedno řešení. `IM` je *Improvement method* vylepšující jedno předané řešení a již zmíněná funkce `IF` rozhodne o jeho následném začlenění do aktuální populace (poolu). Tímto popisným schématem lze modelovat velkou část metaheuristik, příkladem může být *Tabu search* nebo simulované žíhání, které mají $|P| = 2$, $|S| = 1$ a rozhodovací logika (adaptivní paměť a akceptační kritérium) je implementována v `IM`. Genetické algoritmy jsou definovány pro $|P| > 1$ a $|S| > 1$ a operátory křížení a mutace jsou zahrnuty v SCM, krok IM vůbec není využit. Další příklady i s obsáhlou diskuzí lze nalézt v [68].

Jiná základní myšlenka popisu metaheuristik byla publikována v [69] a nazvána *Adaptive memory programming* (AMP). Hlavní roli zde sehrává *paměť*, která slouží ke konstrukci nových řešení. Například u metody *Tabu search* je pamětí *tabu list*, u genetických algoritmů je to celá populace a u optimalizace mravenčí kolonií je pamětí matice feromonových stop. Pseudokódem je Adaptive Memory Programming v publikaci [69] popsaný jako

```
inicializuj paměť M
while termination = FALSE do
    pomocí paměti generuj nové řešení S
    vylepši S pomocí metody lokálního prohledávání
    vylepšené řešení S' použij k aktualizaci paměti M.
```

V uvedeném popisu lze vypozorovat mnoho shodných myšlenek s předchozím obecným modelem. Nejvýznamnější rozdíl mezi nimi je v tom, že *paměť* nemusí být tvořena jen populací řešení, ale může být v podstatě libovolnou datovou strukturou. Vyjadřovací silou se oba modely vzájemně vyrovnají, protože přidaná obecnost, která je u AMP docílena zavedením pojmu *paměti*, může být u předchozího modelu realizována přímo ve funkcích `IF, IM,` ...

Ze zcela jiného pohledu jsou metaheuristiky popisovány pomocí přístupu nazvaného *Agent Metaheuristics Framework* (AMF) [70]. Základem jsou zde *multiagentní systémy*, které udržují

skupinu vzájemně spolupracujících a interagujících objektů (*agentů*). Využit je zde tzv. RIO přístup, který každému agentovi definuje jeho *Roli*, mezi rolemi definuje vzájemné *Interakce* a vše dohromady sjednocuje *Organizace*. V popisu metaheuristiky jsou použity agenti čtyř rolí - *intensifier, diversifier, guide a strategist*[17]. Role intensifier a diversifier zajišťují prohledávání prostoru, a na rozdíl od AMP modelu jsou chápány separovaně. Intensifier se soustředí na slibné oblasti, kde je potřeba prohledávat prostor více důkladně, kdežto diversifier prohledává globálně celý stavový prostor. Role guide zprostředkovává interakci mezi intensifierem a diversifierem a určuje jim, které oblasti budou prohledávat. V AMP modelu této roli částečně odpovídá funkce *paměti*. Poslední rolí je strategist, která interaguje s agenty typu guide a zastupuje adaptivitu algoritmů. Vyhodnocuje tedy aktuální stav prohledávání a podle toho modifikuje parametry celé metaheuristiky. Tímto způsobem lze popisovat většinu běžných metaheuristik, například v *Tabu search* jsou role intensifier a diversifier spojené do jedné, která prohledává aktuální okolí a roli guide hrají funkce pracující s tabu listem a výběrem nového aktuálního řešení. V běžném Tabu search není role strategist zastoupena, jejím přidáním může vzniknout algoritmus *Reacitve search*, který adaptivně mění délku tabu listu. Genetické algoritmy mají roli intensifier reprezentovánu křížením jedinců, roli diversifier mutací a roli guide výběrem rodičů pro křížení.

Definování stávajících metaheuristik pomocí popsaných rolí lze jednotlivé agenty kombinovat a vytvářet nové metaheuristiky. Například *Coalition-based metaheuristika* [71] vznikla použitím AMF modelu a spojením s konceptem hyperheuristik.

V následujících kapitolách bude představen nový obecný model, který se od předchozích odlišuje nejen mírou obecnosti, ale i celkovým přístupem k pojmu metaheuristika. Metaheuristika zde není chápána jako rigidní osnova, která v každém kroku definuje smysl prováděné operace, ale naopak je v návrhu nechávána maximální volnost. Za existenci metaheuristik vděčíme především výpočetní síle dnešních počítačů, a tak i obecný model je vybudován s ohledem na rozumný způsob jejich počítačové implementace.

4.1 Koncepční model

Významným společným znakem většiny kanonických tvarů metaheuristik je, že jedno konkrétní řešení reprezentují jako jednu samostatnou entitu umístěnou ve vhodném prostoru. Většina výzkumníků si tak může představit metaheuristiku jako body rozmístěné ve stavovém prostoru, které se různě pohybují, vytváří a zanikají a vzájemně se ovlivňují. Tato představa se ukázala jako vhodná a intuitivní, a tedy i budovaný obecný model musí umět tuto představu jednoduše zprostředkovat. Za základní stavební kámen modelu musí být zvolena dostatečně obecná struktura, která je schopná přímočaře popisovat běžné metaheuristiky, ale zároveň nesmí nijak omezovat vývoj metaheuristik s odlišným přístupem.

[17] Pojmenování je ponecháno v původním anglickém jazyce

Klíč

Základní a nejobecnější datová struktura je entita nazvaná `klíč`. `Klíč` je abstraktní pojem, pouhé jméno, pod který se může schovat libovolná povolená hodnota. Termín `klíč` je zvolen kvůli významové podobnosti s databázovým klíčem. `Klíč` si lze představit jako jméno proměnné, do které se ukládají vhodné hodnoty. Ve vytvářené metaheuristice může `klíč` zastupovat jedno řešení, nebo i celou jejich skupinu, stejně tak zastupuje i jednotlivé proměnné a nebo pomocné hodnoty. `Klíč` jenom pojmenovává hodnoty, se kterými metaheuristika pracuje a na které se lze průběhu dotazovat.

Každý `klíč` je unikátní a mohou mezi nimi existovat různé vztahy. `Klíče` jsou primárně hierarchicky uspořádány, ale nejsou tím nijak omezeny. Jejich topologie (vzájemné vztahy) mohou být v podstatě libovolné.

Ke každému `klíči` může být připojena určitá hodnota, ale není to nezbytně nutné. `Klíč` může sloužit i jenom jako abstraktní pojmenování pro určitou skupinu hodnot. Pokud má `klíč` nějakou hodnotu přiřazenou, budeme jej psát jako uspořádanou dvojici

$$q_i = (k_i, v_i)\,, \text{ pro } i \in \mathbb{N},$$

kde k_i je jedinečný klíč a v_i je jeho přiřazená hodnota. Vyhledáním daného klíče k_i tedy může být vždy získána i jeho přiřazená hodnota v_i. Soubor všech vytvořených párů q_i reprezentuje *databázi* všech hodnot používaných a sdílených za běhu metaheuristiky.

Základní typy hodnot v, které lze ke `klíči` k přiřazovat jsou

- `klíč` - mohou tak být vytvářeny páry (k_1, k_2),
- `operátor` - základní funkční mechanismus metaheuristického modelu (viz dále),
- datová hodnota - jde o jakýkoliv běžný datový typ, který metaheuristika vyžaduje. Může se jednat o přiřazení čísel, znaků, textových řetězců apod.
- speciální hodnoty - jsou hodnoty využívané k řízení běhu metaheuristiky. Jedná se pouze o hodnoty `LOOP_STOP` a `NONE` (viz dále).

Struktura všech `klíčů` je tvořena stejnými pravidly. `Klíče` jsou hierarchicky uspořádány postupným navazováním základních datových typů. K již vytvořeným `klíčům` lze snadno přidávat nové podklíče, nebo je spolu kombinovat. Pro existující `klíč` k_1 lze vytvořit například následující klíče

$$k_2 = [k_1, x]$$
$$k_3 = [k_1, y]$$
$$k_4 = [k_2, k_3] = [k_1, x, k_2, y],$$

kde `klíč` k_2 je hierarchicky pod klíčem k_1 a jeho jméno je rozšířeno o hodnotu x. Obdobně `klíč` k_3 je hierarchicky pod klíčem k_1 a je rozšířen o hodnotu y. Pokud by `klíč` k_1 reprezentoval jedno řešení (bod ve stavovém prostoru), tak `klíče` k_2 a k_3 mohou reprezentovat jeho souřadnice na dvou osách x a y.

`Klíč` k_4 je vytvořený spojením `klíčů` k_2 a k_3 a reprezentuje jejich vzájemný vztah. Přiřazením hodnoty k takovému `klíči` lze vytvářet v podstatě libovolné relace, a to dokonce i fuzzy relace nebo grafové struktury s definovanými váhami. Uspořádané dvojice

$$q_1 = ([k_1, k_2], 0.4)$$
$$q_2 = ([k_1, k_3], 0.8)$$

tak mohou reprezentovat například fuzzy relaci prvků k_1 a k_2 s hodnotou 0.4 a nebo hranu v grafu mezi vrcholy k_1 a k_3 s váhou 0.8. Důležité je ale si uvědomit, že prvek $[k_1, k_2]$ je pouze novým `klíčem`, ke kterému lze přistupovat jako ke kterémukoliv jinému `klíči` (tzn. vytvářet mu nové podklíče, řetězit apod.)

K intuitivnímu uchopení navrhovaných metaheuristik je velice důležité snadno pracovat s celými skupinami `klíčů` zároveň. Uspořádaná posloupnost `klíčů` by tak mohla být jedním ze základních datových typů modelu, ale jelikož ji lze snadno implementovat za pomoci ostatních typů, tak je definována pouze jako doplňková struktura. Klíč k_s reprezentující skupinu je tedy definován jako

$$q_0 = ([k_s, n], 4)$$
$$q_1 = ([k_s, 1], v_1)$$
$$q_2 = ([k_s, 2], v_2)$$
$$q_3 = ([k_s, 3], v_3)$$
$$q_4 = ([k_s, 4], v_4),$$

kde dvojice q_0 udává celkovou délku skupiny a každá další dvojice přiřazuje danému indexu i hodnotu v_i. V tomto případě není nutné, aby klíč k_s měl přiřazenou nějakou konkrétní hodnotu a *fyzicky* existovat v databázi mohou pouze jeho podklíče.

V dalším textu bude tato struktura používána ve zkrácené formě

$$q_s = (k_s, \langle v_1, v_2, v_3, v_4 \rangle).$$

Operátor

Základním funkčním prvkem v hybridním metaheuristickém modelu je entita nazvaná `operátor`. Myšlenkově jde o koncept blízký funkcím ve funkcionálním programování[18] popř. i v

procedurálním programování[19]. `Operátorem` je tedy rozuměn ucelený blok instrukcí, které jsou vykonávány nad předanými `klíči`. Každý `operátor` má přiřazený `klíč`, který obsahuje vstupní hodnoty a `klíč`, ke kterému se přiřadí zpracované výstupní hodnoty. Výstupem je vždy pouze jeden `klíč`, kdežto vstupních `klíčů` může být více. Ve shodě s běžnou syntaxí budeme pro zjednodušení psát

$$k_{\text{out}} = \text{op}(k_{\text{in}}^1, k_{\text{in}}^2, \ldots, k_{\text{in}}^m),$$

což znamená, že vstupem do operátoru je skupina `klíčů` $k_{\text{in}}^1, k_{\text{in}}^2, \ldots, k_{\text{in}}^m$ a výstup je přiřazen ke `klíči` k_{out}.

Jednotlivé instrukce v `operátoru` mohou být buď další `operátory` a nebo přímé volání nízkoúrovňových funkcí poskytované programovým prostředím implementovaného modelu (matematické funkce, uživatelsky definované funkce, ...). Každý `operátor` má neomezený přístup k databázi uložených hodnot, tedy ze znalosti klíče k může kdykoliv získat jeho přiřazenou hodnotu v. Stejně tak mohou `operátory` data i ukládat, tedy přiřazovat `klíčům` k jejich hodnoty v.

Běh metaheuristiky je popsán pouze jako posloupnost po sobě jdoucích operátorů s definovanými vstupními a výstupními `klíči`.

$$\begin{aligned} k_1 &= \text{op}_1(k_{\text{in}}) \\ k_2 &= \text{op}_2(k_1) \\ k_3 &= \text{op}_3(k_1, k_2) \\ k_4 &= \text{op}_4(k_3) \\ &\ldots \end{aligned}$$

Z Churchovy-Turingovy teze plyne, že pro každý algoritmus lze vytvořit ekvivalentní Turingův stroj (tzn. algoritmus je *naprogramovatelný*) pokud je použit programovací jazyk s alespoň jednou z následujících vlastností

- *neomezená rekurze* - možnost (teoreticky) neomezeného volání funkce sebe sama,
- *cyklus while* - opakování bloku kódu tak dlouho, dokud nepřestane platit definovaná podmínka,
- *podmíněný skok* - výsledek vyhodnocení definované podmínky rozhodne o místě v programu, kterým se bude pokračovat.

---

[18] přeneseně také v lambda kalkulu - výpočetním modelu funkcionálního programování.

[19] V procedurálním programování jsou funkce chápány jako prostředek pro zjednodušení psaní programu a teoreticky se bez nich lze obejít. Ve funkcionálním programování tvoří funkce nedílnou součást jeho paradigmatu, a proto je toto pojetí bližší popisovanému pojmu `operátor`.

Aby byl i metaheuristický model schopný popsat jakýkoliv algoritmus, musí mít také alespoň jednu z uvedených vlastností. Pouhým definováním sledu jednotlivých operátorů by nebylo možné realizovat komplexní řídicí logiku.

K získání potřebných vyjadřovacích schopností je koncept `operátorů` rozšířen o možnost opakování. Každý `operátor` tedy definuje posloupnost instrukcí a při jejich dokončení jsou všechny znovu opakovány. Opakování probíhá tak dlouho, dokud nějaká z instrukcí nepřiřadí ke svému výstupnímu klíči k_{out} speciální hodnotu `LOOP_STOP`, která zapříčiní okamžité ukončení operátoru a pokračování jeho následníkem. Pokud je tedy `operátor` op_A definován např. jako posloupnost následujících `operátorů`

$$\begin{aligned}\text{op}_A{:=}&\\ &k_1 = \text{op}_1(k_{\text{in}})\\ &k_2 = \text{op}_2(k_1)\\ &k_3 = \text{op}_3(k_1, k_2)\\ &k_4 = \text{op}_4(k_3),\end{aligned}$$

přiřazují se výstupním `klíčům` při postupném vyhodnocování určené hodnoty a po dokončení `operátoru` op_4 se pokračuje znovu `operátorem` op_1. Pokud v dalším běhu přiřadí `operátor` op_2 ke svému výstupnímu `klíči` k_2 hodnotu `LOOP_STOP`, operátory op_3 a op_4 se přeskočí a pokračuje se následujícími operátory (pokud žádné v posloupnosti již nejsou, běh metaheuristiky se ukončí). Jako výstup `operátoru` op_A je použita poslední platná přiřazená hodnota před přiřazením hodnoty `LOOP_STOP`. V uvedeném příkladě by k výstupnímu `klíči` k_A `operátoru` op_A byla přiřazena hodnota z `klíče` k_1.[20] Operátor tedy může vracet buď `klíč`, nebo hodnotu `LOOP_STOP` a nebo speciální hodnotu `NONE`, která značí prázdný výstup. Tímto jednoduchým principem lze snadno realizovat *cyklus while*, a tím splnit jednu z výše uvedených podmínek pro úplný programovací jazyk. V každém `operátoru` lze například definovat `operátor` op_T, který bude sloužit pouze k vyhodnocování ukončovací podmínky a v případě jejího splnění vrátí hodnotu `LOOP_STOP` a v případě nesplnění naopak vrátí hodnotu `NONE`.

$$\begin{aligned}\text{op}_B{:=}&\\ &k_1 = \text{op}_1(k_1)\\ &k_2 = \text{op}_\text{T}(k_1)\end{aligned}$$

Zde se bude `operátor` op_1 vykonávat tak dlouho, dokud jej `operátor` op_T neukončí. Za povšimnutí stojí, že výstupní `klíč` `operátoru` op_1 je stejný jako jeho vstupní `klíč` a může tedy opakovaně zpracovávat vstup, dokud není podle předdefinovaných kritérií v op_T.

[20] Tento přístup je sice dostatečně obecný, ale z uživatelského hlediska není nejpřívětivější. V programové implementaci je přístup již vhodně přizpůsoben.

Cyklus *while* je přirozeným vyjadřovacím prvkem v metaheuristických algoritmech a je i hlavním koncepčním rozdílem oproti funkcionálnímu programování, kde tuto úlohu sehrává *rekurze*. Protože cyklus *while* je dostačující logickou strukturou pro naprogramování libovolného algoritmu

Interpretace modelu

Důvodem, proč je metaheuristický model vybudován popsaným způsobem, je jeho hlavní smysl, tedy snadná hybridizace. Koncept `operátorů` vytváří jednotné abstraktní pojetí řídicí logiky a jejích jednotlivých procedur. V kontextu typických metaheuristik, např. genetických algoritmů, zastupuje `operátor` např. způsoby křížení chromozomů, výběr populace nebo ukončovací podmínku. Standardizací těchto operátorů lze docílit přímočaré hybridizace, tedy kombinování operátorů napříč různými metaheuristickými koncepty.

Pojem `klíč` naopak vytváří abstrakci pro datové hodnoty a zároveň i pro myšlenkové celky použití v metaheuristikách. V případě genetických algoritmů mohou `klíče` reprezentovat jednotlivé chromozomy, jejich geny i celé populace. Pro úspěšnou hybridizaci tedy stačí, aby byla tato data reprezentována stejnými strukturami, a tím i využívána různými `operátory`.

4.2 Formalizovaný model hybridního metaheuristického modelu

Pro snadnější implementaci a srozumitelnost konceptu je popsaný hybridní metaheuristický model převeden do rigorózní terminologie. Tento druh popisu usnadňuje a zpřesňuje použité myšlenky. Koncepční ani následující matematický model nejsou přímo vhodny k implementaci pomocí programovacích jazyků. Jejich smysl je podávat základní, ale zároveň úplný, popis hybridního metaheuristického modelu. Pro implementaci se pak může zvolit nadstavba vybudovaná na popsaném základním modelu, která usnadní a zpříjemní vytváření nových algoritmů.

Klíč

`Klíč` tvoří základní stavební kámen modelu a slouží k pojmenování zpracovávaných hodnot i jejich celých skupin. Klíče mohou být složeny z libovolných prvků, ale nejsnáze se představují ve formě textových řetězců. Není však nutno se omezovat pouze na množinu písmen, ale lze využít libovolnou obecnou množinu, na které je možné definovat relaci uspořádání.

Pro definitorické účely tedy lze uvažovat množinu A jako základní *abecedu*, ze které jsou složena *slova* využitelná pro vytváření `klíčů`. Jelikož *slovo* je posloupnost prvků *abecedy*, lze relací uspořádání abecedy A uspořádat i množinu všech *slov*, tzv. *lexikografickým uspořádáním*. Možnost řadit vytvořená slova není pro matematický model podstatná, ale je velmi užitečná z následných implementačních důvodů. Množinu všech neprázdných *slov* označme jako A^+ a její vhodnou podmnožinu jako $\Sigma \subset A^+$. Množina Σ je tedy *jazyk* nad *abecedou* A.

Jazyk Σ lze považovat za *abecedu*, ze které jsou tvořeny `klíče`. Σ tedy obsahuje vhodná pojmenování proměnných použitých v modelované metaheuristice. Množinu všech neprázdných klíčů označme jako Σ^+, tedy jako *jazyk nad abecedou* Σ.

K vytváření `klíčů` ze slov jazyka Σ^+ je dobře využitelná standardní operace *zřetězení* označovaná "." (tečka). `Klíč` $\sigma \in \Sigma^+$ tak může být získán jako $\sigma = \sigma_1.\sigma_2$. (Platným `klíčem` jsou jak prvky Σ, tak i prvky Σ^+, protože $\Sigma \subset \Sigma^+$). Operace zřetězení je asociativní a platí tedy $(\sigma_1.\sigma_2).\sigma_3 = \sigma_1.(\sigma_2.\sigma_3)$.

Pro implementační důvody je také důležitý pojem *prefix* (předpona) `klíče` $\sigma \in \Sigma^+$, což je takové slovo σ_p, že platí $\sigma = \sigma_p.\sigma_x$ pro nějaké $\sigma_x \in \Sigma^+$.

Pro popis běhu metaheuristického modelu jsou důležité dva speciální symboly

- τ odpovídající hodnotě `LOOP_STOP` a
- ε odpovídající hodnotě `NONE` a reprezentující prázdné slovo.

Ani jeden z těchto symbolů nesmí být využit pro vytváření klíčů. Jako množinu Σ^* označme rozšíření množiny všech `klíčů` o prázdné slovo ε, tedy $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

Paměť metaheuristiky je definována jako množina uspořádaných dvojic $M = \{(\sigma_i, v_i), \ldots\}$, kde $\sigma_i \in \Sigma^+$ jsou vytvořené klíče a $v_i \in \mathcal{V}$ jsou hodnoty, které jsou ke klíčům přiřazeny. Hodnoty v_i mohou být buď `klíče`, `operátory`, prázdný klíč ε a nebo běžné datové hodnoty (čísla, řetězce atd.). Množina těchto přípustných hodnot je označena jako $\mathcal{V}$.

Následující definice popisují druhy *skupin klíčů*, které jsou v modelu využity. Jako *skupina klíčů* délky k bude označována uspořádaná posloupnost $p^k = \{\sigma_i\}_{i=1}^k$, kde $\sigma_i \in \Sigma^+$ a $i = 1, \ldots, k$. Jako p^0 je označována prázdná posloupnost nulové délky. V některých případech ale není přesný počet klíčů v posloupnosti důležitý, a proto je definována pouze nějaká jeho hranice. Definujme následující symboly pro libovolné $k \in \mathbb{N}$

- p^{k+} je posloupnost klíčů délky alespoň k.
- $p^+ \equiv p^{1+}$ je posloupnost klíčů libovolné nenulové délky.
- $p^* \equiv p^{0+}$ je posloupnost klíčů libovolné délky, včetně prázdné posloupnosti (tedy posloupnosti délky 0).
- $\mathcal{P}^k$ je množina všech posloupností p^k, tedy posloupností délky k.
- $\mathcal{P}^{k+}$ je množina všech posloupností p^{k+}, tedy posloupností délky alespoň k.
- $\mathcal{P}^+$ je množina všech nenulových posloupností p^+.
- $\mathcal{P}^*$ je množina všech posloupností p^*.

Všechny uvedené posloupnosti jsou složeny z prvků množiny Σ^+, tedy neobsahují prázdný klíč ε. Pokud jej budou moci obsahovat, tak bude k označení posloupnosti připsán dolní index ε (například p_ε^{k+} nebo $\mathcal{P}_\varepsilon^*$).

`Operátory` zpracovávají hodnoty získané z předaných klíčů a poskytují jednu výstupní hodnotu. V matematickém modelu budou operátory z důvodu kategorizace rozděleny na dva základní typy. Obecnější typ operátoru je definován za použití `klíčů`, tedy jako zobrazení n-tice klíčů do množiny přípustných hodnot:

$$X : \Sigma_1 \times \ldots \times \Sigma_n \to \mathcal{V} \cup \{\tau\}.$$

Tyto operátory zobecňují všechny procedury potřebné pro běh metaheuristiky. Hodnota $n \in \mathbb{Z}_0^+$ a může být tedy i nulová, což značí žádný vstupní klíč. Jako množinu všech operátorů obecného typu označme množinu $\mathcal{X}$.

Druhý typ operátorů se váže k základním principům metaheuristiky a je vytvořen kvůli klasifikaci jejích jednotlivých operátorů. Vstupem do tohoto operátoru je uspořádaná n-tice posloupností, kde $n = 0, 1, \ldots$ a výstupem `operátoru` může být buď posloupnost `klíčů` a nebo nějaký ze speciálních symbolů. Obecně lze tedy tento `operátor` definovat jako zobrazení

$$X_{\mathcal{P}} : \mathcal{P}_1 \times \ldots \times \mathcal{P}_n \to \mathcal{P}_{n+1} \cup \{\tau, \varepsilon\},$$

kde $\mathcal{P}_i$ jsou vhodné podmnožiny $\mathcal{P}^*$, přičemž pro $i \in \{1, \ldots, n\}$ se jedná o množiny vstupních posloupností a pro $i = n + 1$ je množina výstupní.

Operátory typu $X_{\mathcal{P}}$ lze převést na operátory obecného typu pomocí speciálního operátoru $X_S \in \mathcal{X}$, který je obaluje. Operátor X_S pouze převezme hodnoty klíčů k_i, najde pro ně v paměti odpovídající hodnoty v_i a použije jako vstup pro operátor $X_{\mathcal{P}}$. Výstupní posloupnost je uložena do paměti a odpovídající klíč je použit jako výstup X_S.

Metaheuristika $\mathcal{M}$ je posloupnost instrukcí $(X, \sigma_{\text{in}}, \sigma_{\text{out}})$, kde X je `operátor` se vstupy z `klíče` σ_{in} a výstupem přiřazeným do σ_{out}. Ke vstupnímu `klíči` může být přiřazena celá n-tice klíčů vstupující do operátoru.

Stroj, který definovanou metaheuristiku $\mathcal{M}$ umí provést, je stroj s jednoduchou řídicí logikou a schopností číst a zapisovat data do paměti M. Ke čtení paměti slouží funkce $R_M(k)$ vracející pro `klíč` k hodnotu v takovou, že $(k, v) \in M$. Zápis naopak provádí funkce $W_M(k, v)$ tak, že odstraní předchozí záznam (k, v_0) z M, tedy $M_{\text{temp}} = \{(k_1, v_1) : (k_1, v_1) \in M \wedge k_1 \neq k\}$ a následně množinu M_1 rozšíří o nový klíč $M = M_{\text{temp}} \cup \{(k, v)\}$.

Logika provádění metaheuristiky je rekurzivním voláním jednoduchého vykonání operátoru

$i = 1$

dokud $i \leq |\mathcal{M}|$

  přečti instrukci $q_i = (X_i, \sigma^i_{\text{in}}, \sigma^i_{\text{out}})$

  přečti vstupní parametry $p = R_M(\sigma^i_{\text{in}})$

  vykonej operátor $v = X_i(p)$

když $v = \tau$, tak ukonči provádění operátoru a vrať výsledek

ulož výstup $W_M(\sigma^i_{\text{in}}, v)$

zvyš hodnotu i na hodnotu i+1 a opakuj vykonání instrukce

opakuj vykonání operátoru

Dokud nějaký z operátorů nevrátí symbol τ, stále se opakuje vykonávání celého operátoru. Jakmile se symbol τ objeví, je běh operátoru ukončen a reportuje se poslední platná hodnota ($v \neq \tau$). Operátory, které jsou obsaženy v přečtených instrukcích, se vykonávají stejným postupem. Rekurzivně tak lze tedy vykonat celou metaheuristiku.

Možnosti aplikace

Popsaný algoritmický modelu je natolik univerzální, že v něm lze popsat jakýkoliv algoritmus[21]. Pro mnoho běžných úloh by byl ale naprosto nevhodný a byl vytvořen pouze se záměrem usnadnit práci s metaheuristikami. Pro metaheuristiky je přirozené, že jsou definovány jako operátory předávající si posloupnosti obsahující různá řešení úlohy. Hlavní síla modelu je v tom, že reflektuje tento přístup a dovoluje tak vytvářet množství nových konceptů. Jednotný způsob ukládání dat a operátorů, spolu s přímočarým seskupování proměnných do logických struktur, vede ke snadnějšímu zobecňování a znovupoužitelnosti metaheuristických konceptů.

Mnoho metaheuristik se zdá být na první pohled významně rozdílných, ale popsaným modelem je lze sjednotit. Následující příklady demonstrují šíři konceptů, které lze modelem jednoduše realizovat

- Základním přístupem je udržování *populace* jedinců, kde každý jedinec reprezentuje samostatné řešení. Jedinec je tedy určen klíčem k_i, který je zřetězený s klíči odpovídajícími souřadnicím řešení (tedy $k_i.x$, $k_i.y$, ...). Populace je posloupnost klíčů k_i.
- Posloupnosti jsou vhodně využitelné i v lokálním prohledávání, kde se přímo nepracuje s populacemi, ale i přesto vznikají větší soubory řešení. *Hill climbing* algoritmus nebo *Tabu search* generují pro aktuální řešení jeho *okolí*, které je snadno pomocí posloupnosti reprezentovatelné. V *Tabu search* se posloupnost klíčů přímo hodí i *Tabu list*, tedy seznam posledních navštívených bodů.
- Z pohledu vývoje nových prohledávacích strategií je velice výhodná *bezschémovost* databáze. Například rozšířit standardní genetický algoritmus o pohlavní křížení lze docílit pouze jednoduchým přiřazením pohlaví každému vygenerovanému řešení k_i, tedy např. $(k_i.\text{sex}, \text{female}$ a přeformulovat operátory, aby jej braly v potaz.

[21] Pro deterministický Turingův stroj.

- Mravenci v *Ant colony optmization* zanechávají feromonové stopy mezi jednotlivými uzlovými body prohledávaného prostoru a podle toho se řídí jejich následující cesty. Každý uzlový bod lze reprezentovat vlastním klíčem b_i a aktuální feromonovou stopu mezi uzly b_u a b_v přiřadit ke klíči vzniklému jejich zřetězením, tedy např. $(b_u.b_v, 0.2456)$.
- Pokud je potřeba pracovat s *více instancemi jedné metaheuristiky* zároveň (více nezávislých feromonových stop zároveň), docházelo by při předchozím způsobu ke kolizi dat v databázi. Stačí ale přiřadit pro každou instanci unikátní klíč u_i a problém je ihned vyřešen: $(u_i.b_u.b_v, 0.2456)$.
- Stejně, jako může fungovat více instancí metaheuristik dohromady, tak může být i *jedna instance použita na různé populace*. Jedná se o přístup typický pro paralelní metaheuristiky, kde existuje několik nezávislých populací, které jsou zpracovávány stejným přístupem jen s odlišnými parametry. Populace jsou uspořádány do nejrůznějších topologií a podle nich si mezi sebou vyměňují určité informace (jedince). Každá populace má svůj klíč a topologie je popsána jejich zřetězením. Například $(p_1.p_2, 1)$ znamená, že mezi populací p_1 a p_2 je nějaký vztah, který lze specifikovat např. počtem vyměňovaných jedinců označených jako n: $(p_1.p_2.n, 8)$.
- Paralelizace je často řešena tak, že jeden prvek patří do právě jedné populace. Není ale problém prvek zařadit do více populací zároveň. Zařazení prvku do libovolného počtu populací je pouze otázkou jeho vložení do příslušné posloupnosti. Správné zacházení s takovými prvky musí být ale realizované pečlivě navrženými operátory.
- Zařazení prvku do populace nemusí být pouze jednoznačné (tedy *patří* nebo *nepatří*), ale může tam patřit s různými stupni příslušnosti. Lze tak formulovat fuzzy metaheuristiku, kde pro populaci p_i a prvek k_j se snadno zapíše stupeň příslušnosti jako např. $(p_i.k_j, 0.8)$. Stejný zápis může být použit i pro pravděpodobnostní nebo jakýkoliv jiný vztah mezi populací a nějakým prvkem.
- I samotné prvky mezi sebou mohou mít různé relace, které lze definovat libovolně podle potřeb. Například v názvosloví evolučních algoritmů má každý prvek k_i svého rodiče r_j. Takový vztah lze zapsat buď jako $(k_i.r_j, \text{parent})$, nebo i jako $(k_i.\text{parent}, r_j)$. Rozdíl mezi těmito záznamy je ve způsobu vyhledávání této informace, tedy v otázce, kterou chceme danou informací zodpovědět. V prvním případě by otázka zněla *"Jaký je vztah mezi k_i a r_j?"* a odpověď by byla *parent*. Druhým záznamem odpovídáme na otázku *"Kdo je rodičem klíče k_i?"*. Zvolení správného zápisu informace do paměti tedy musí být vždy přizpůsobeno povaze operátorů, které s nimi pracují.
- Relace mezi klíči (prvky, populacemi, ...) nemusí být pouze binární[22]. Stejným způsobem lze postupovat i při definování obecných n-arních relací. Zařazení tří klíčů k_1,k_2 a k_3 do relace

[22] Binární relace určuje vztah mezi dvěma prvky. Obecné relace mohou určovat vztah mezi libovolným počtem prvků.

lze docílit jejich zřetězením $k_1.k_2.k_3$ a přiřazením odpovídající hodnoty, např. $(k_1.k_2.k_3, 1)$. Pokud je relace R symetrická vůči všem kombinacím klíčů, tzn. když $k_1.k_2.k_3 \in R \Rightarrow k_3.k_1.k_2 \in R$ apod., nemusí se do databáze zapisovat všechny tyto kombinace, ale stačí je uložit pouze v lexikografickém uspořádání a považovat ji za kanonickou formu.

- Různé metaheuristiky vyžadují různé druhy informací přiřazených ke konkrétním řešením. Model dovoluje snadnou hybridizaci v tom smyslu, že jedno řešení může mít zároveň přiřazenou normalizovanou hodnotu účelové funkce (potřebnou pro operátory umělých imunitních systémů), tak i např. rychlost (potřebnou pro operátory hejnových algoritmů). Vše je realizováno pouze přidáním odpovídajícího klíče do databáze. Pokud by hrozila kolize používání stejných jmen pro různé parametry, lze tomu předejít zřetězením s novým klíčem definujícím konkrétní operátor.
- Díky definici databáze takovým způsobem, že klíčům mohou být přiřazována jak data tak i operátory, lze mnoho operátorů navržených pro práci s běžnými řešeními úlohy použít i pro práci s operátory. Koncept hyperheuristik je založený na zkoušení a ohodnocování různých nízkoúrovňových heuristik (operátorů) a podle jejich výsledků je používat na další hledání. Místo populace řešení tak lze používat populaci operátorů, což pro velkou část tradičních operátorů není žádný rozdíl, protože obojí je reprezentováno pomocí klíče. Např. operátory výběru tak lze používat naprosto beze změny, stačí jen přiřadit operátorům v populaci odpovídající hodnoty účelových funkcí.

Možnosti hybridizace

Smyslem budování obecného modelu je usnadnit hybridizaci různých metaheuristik dohromady a vtisknout jim jednotnou podobu. Rozložením chování metaheuristik na samostatné operátory a sjednocením komunikačního rozhraní lze dosáhnout snadné záměny a spolupráce jednotlivých konceptů. Každý operátor je z vnějšku popsán počtem a délkami svých vstupních posloupností a stejně tak i délkou své výstupní posloupnosti. Pokud tedy existují dva různé operátory, které mají stejné velikosti vstupů a výstupu, jsou vzájemně zaměnitelné.

V případě metody *Tabu search* se z jednoho bodu generuje jeho okolí a z tohoto okolí se zase vybere jediné řešení. Jsou to tedy dva operátory

$$X_{\text{tabu}_1} : \mathcal{P}^1 \to \mathcal{P}^n$$

a

$$X_{\text{tabu}_2} : \mathcal{P}^n \to \mathcal{P}^1,$$

kde n je nějaké přirozené číslo závislé na dimenzi úlohy. Operátor X_{tabu_2} má stejný typ jako například celý genetický algoritmus

$$X_{\mathrm{GA}} : \mathcal{P}^n \to \mathcal{P}^1,$$

který z počáteční populace o velikosti n vrátí po několika iteracích nejlepší nalezené řešení. Jedním ze způsobu hybridizace těchto dvou zmíněných algoritmů tedy může být nahrazení jednoduchého výběru nejlepšího prvku z vygenerovaného okolí za celý genetický algoritmus. Doba výpočtu se tím určitě prodlouží, ale získá se tím kvalitativně nové chování algoritmu a tím i možnost lepšího výkonu pro nějaké typy účelových funkcí.

Hlavní rys algoritmu *Tabu search*, jeho hlavní odlišnost od ostatních metaheuristik, je využívání seznamu několika posledních navštívených bodů společně s metodou lokálního prohledávání. Při globálním prohledávání nemá *Tabu list* velký smysl, protože je velice malá pravděpodobnost navštívení stejných bodů víckrát po sobě během krátké doby. Zakomponovaný genetický algoritmus do nastíněné hybridní metaheuristiky by neměl vykonávat stovky iterací, jako je tomu běžné, ale může skončit pouze po několika málo iteracích, které tak přinesou lokálnější prohledávání stavového prostoru.

Ne každé nahrazení operátoru tedy nemusí být rozumné a vnější kompatibilita operátorů by neměla být jediným indikátorem jejich možné záměny. Mnoho funkčních konceptů ale může vzniknout i z na první pohled nerozumných záměn, a proto je obecný model nijak neomezuje.

Ani samotná vnější definice celé metaheuristiky nemusí být jednoznačná. Zmíněný genetický algoritmus typicky vrací pouze nejlepší prvek, který nalezl během svého běhu. Algoritmus ale lze nadefinovat i např. jako

$$X_{\mathrm{GA}_n} : \mathcal{P}^n \to \mathcal{P}^m,$$

tedy že vrací celou závěrečnou populaci (tedy $m = n$) nebo například pouze m nejlepších prvků. Tímto přístupem lze stejný operátor využít pro hybridizaci dalších typů metaheuristik.

4.3 Shrnutí obecného modelu

Vytvořený model byl popsán různými navzájem doplňujícími se způsoby. Koncepční model v kapitole 4.1 popisuje základní struktury určující celou povahu modelu. Jedná se o nejobecnější řídicí principy a způsob ukládání informací potřebných pro řádný běh metaheurisitky. Jako hlavní typ pro uchování dat je popsán `klíč` zastupující libovolný objekt metaheuristiky. Klíče lze spolu řetězit, čímž se mohou vytvářet vazby mezi jednotlivými entitami modelu. Hlavními funkčními objekty vykonávající práci s daty (klíči) jsou `operátory`. Operátory jsou v metaheuristikách popsány pouze pomocí vstupních a výstupních parametrů. Každá metaheuristika tedy může být definována naprosto abstraktně, tzn. bez znalosti konkrétních operátorů.

Kapitola 4.2 ukazuje formalizovaný způsob popisu vhodný pro klasifikaci operátorů podle jejich vnějšího rozhraní. Vytvořené popisy jsou vhodné pro základní popis chování, ale nejsou příliš vhodné pro reálnou práci. Počítačová implementace modelu popsaná v nezkrácené verzi

disertační práci významně zjednodušuje práci s modelem a přináší několik užitečných rozšíření (metody `map`, `apply`, anonymní bloky, automatické ukončování operátorů, ...). Všechna rozšíření jsou plně kompatibilní se základním modelem a nijak ho nemění.

Metaheuristický model je natolik silný a obecný, že dokáže přirozeně popsat velké množství metaheuristik. Lze vytvářet metaheuristiky základních typů nebo i velmi specifické, metaheuristiky pro lokální i globální prohledávání, nebo dokonce metaheuristiky paralelní s libovolnou topologií.

Hlavním smyslem modelu je ale umožnit snadnou hybridizaci různých metaheuristických konceptů. Toho lze dosáhnout rozložením známých metaheuristik na samostatné operátory (nebo jejich logické celky) a následně vytvořit abstraktní popis jejich spolupráce. Metaheuristika, která je takto popsána, může být aplikována na řešení nějaké úlohy až po úplném dodefinování všech abstraktních parametrů. A právě toto je prostor pro hybridizaci, protože není důležité, jak přesně daný operátor uvnitř pracuje, ale je důležité, jaké dokáže zpracovat vstupy a kolik výstupů z nich vrátí. Prohazováním operátorů z jednoho konceptu do druhého lze dosáhnout odlišného chování, a tím i možnosti zvýšit šanci na nalezení optima.

Metaheuristika je tedy abstraktně popsána jako aplikování *nějakých* operátorů na *nějaké* klíče. Konkrétní operátor se přiřazuje abstraktnímu klíči až v době kompilace celého algoritmu. Některé operátory ale mají v metaheuristice své pevné místo a nemají možnost být zaměněny za jiné. Jsou to buď pomocné technické operátory a nebo takové, které tvoří samotný základ metaheuristiky a bez kterých by daná metaheuristika nebyla tím, čím je. Například pro metodu *Tabu search* to jsou operátory pracující s tabu listem a všechny ostatní jsou zaměnitelné (generování okolí, výběr nejlepšího prvku, apod.). Rozhodnutí, které operátory budou fixní a které zaměnitelné, je plně na úvaze tvůrce metaheuristiky. Obecně lze říci, že pokud operátor není závislý na typu stavového prostoru úlohy, je velmi pravděpodobné, že tvoří neoddělitelnou část metaheuristiky.

5 Závěr

Představená práce je věnována optimalizaci pomocí metaheuristických metod. Existuje mnoho tříd optimalizačních problémů stejně jako možných přístupů k jejich řešení. Analýza jejich složitosti ukazuje, proč jsou některé z těchto problémů komplikovanější než jiné a hlavně, že existuje i velké množství problémů, které v dnešní době nejsme schopni efektivně řešit vůbec. Právě tyto *nejsložitější* problémy ospravedlňují existenci metaheuristického přístupu, protože jeho prostřednictvím jsou nacházeny alespoň nějaká, i když ne nutně optimální, řešení.

Metaheuristiky lze velmi zjednodušeně popsat jako *chytřejší náhodné prohledávání*. Jejich princip je založen na iterativním procházení bodů stavového prostoru úlohy a systematickém zpracovávání takto získaných informací. Metaheuristiky nejsou striktně definované algoritmy, ale spíše jen obecná schémata dovolující budovat efektivní a problémově specifické způsoby řešení. Snahou metaheuristik je tak poskytovat návod ke konstruování algoritmů bez ohledu na druh stavového prostoru úlohy. Stejným metaheuristickým konceptem proto mohou být vytvářeny algoritmy pro kombinatorické, spojité, grafové nebo jakékoliv jiné optimalizační úlohy.

Žádná metaheuristika však nemůže být univerzálně nejlepší pro všechny typy úloh. Místo hledání nejefektivnější metaheuristiky je tak rozumnější disponovat vzájemně rozdílnými koncepty, ze kterých je možné vybírat nejvhodnější typ pro danou úlohu. V současné době živelně vzniká nespočet nových metaheuristik inspirovaných nejrůznějšími přírodními fenomény. Pomocí připodobnění vytvořené metaheuristiky k nějakému objektivně fungujícímu jevu (evoluce, gravitace, biologické chování zvěře, ...) se jim často dodává potřebné zdání korektnosti a robustnosti. Všechny využité předlohy jsou ale značně zjednodušovány a často jsou u nich zanedbávány i velmi významné okolní vlivy. *Bat-inspired algorithm* tak ve skutečnosti nemá nic společného s netopýry a ani genetické algoritmy nesimulují reálné procesy biologického rozmnožování. Každá vytvořená metaheuristika však obsahuje víceméně podrobný popis jednotlivých kroků abstraktního procházení stavového prostoru úlohy. Terminologie i způsob podání se liší v závislosti na druhu deklarované inspirace, což může velmi znesnadňovat jejich používání. Při aplikaci na konkrétní problém není nijak zvlášť užitečné dogmaticky udržovat původní smysl metaheuristiky, ale může být naopak velmi přínosné inovativně obměňovat základní schéma tak dlouho, dokud nenaplňuje požadované kvality. Koncept jedné metaheuristiky tak může být obohacován prvky z naprosto odlišných schémat, čímž mohou vznikat algoritmy se zcela novými vlastnostmi.

Kombinování různých schémat dohromady je obvykle nazýváno *hybridizací* a v kontextu metaheuristik se většinou jedná o intuitivní *ad-hoc* spojení několika nezávislých principů. Metaheuristiky nemají žádnou pevně danou strukturu ani definici, a proto není cesta k univerzálnímu hybridizačnímu postupu úplně přímočará. V této disertaci byl představen nový obecný způsob popisu metaheuristik vyvinutý za účelem zjednodušení návrhu i následné hybridizace. Tento metaheuristický model je vytvořený natolik obecně, že dokáže zastoupit jakýkoliv algo-

ritmus. Většina běžných algoritmů by ale pomocí něj byla zapsána velice těžkopádně, protože jeho hlavní rysy jsou voleny speciálně s ohledem na typické potřeby metaheuristik.

Základní datový stavební kámen modelu je *klíč* a základním funkčním prvkem je *operátor*. Klíče zobecňují přístup metaheuristiky k paměti a slouží pro předávání dat. Klíč nemá žádný datový typ a lze k němu přiřazovat libovolnou hodnotu (čísla, operátory, klíče, posloupnosti, ...). Operátory zapouzdřují určitou funkční logiku, a to buď nízkoúrovňové úkony programovacího jazyka, nebo posloupnost jiných operátorů. Každý operátor může získávat vstupy z předaných klíčů a své výstupy naopak může přiřazovat pod vlastní definovaný klíč. Každá metaheuristika je modelována jako posloupnost operátorů, které si navzájem předávají své výstupní klíče.

Všechna data jsou udržována ve sdílené databázi, do které má každý operátor neomezený přístup. Pro získání dat je ale potřeba znát přesný klíč, pod kterým jsou uložena. Klíče nezastupují pouze jednotlivá data, ale popisují i vztahy mezi nimi. Klíče tak slouží pro uspořádání dat do skupin (např. populací), pro přiřazování hodnot abstraktním objektům (např. souřadnice řešení, hodnoty účelové funkce), pro popsání relací mezi klíči (např. rodič-potomek) a mnoho dalších. Operátory převezmou předané klíče, zpracují je a vrátí výsledek. Jediná řídicí logika, která je v modelu obsažena, je cyklus `while` ukončovaný speciální hodnotou `LOOP_STOP`. Konkrétně operátory uvnitř vybrané skupiny jsou volány stále dokola, dokud libovolný z nich nevrátí `LOOP_STOP`.

Navržený model významně akcentuje základní charakteristiku metaheuristik, a to jejich obecnost. Metaheuristiky jsou nezávislé na druhu stavového prostoru, a proto ani model nevyžaduje zadání konkrétního typu. Ze stejného důvodu model nevyžaduje ani zadání problémově (prostorově) závislých operátorů. Metaheuristika je pouhým konceptem a jako koncept je modelem i popsána. Definovány jsou pouze ty operátory, které tvoří samotný princip metaheuristiky, tedy např. operátory pracující se seznamem posledních navštívených bodů v metodě *Tabu search* apod.

Celý model je tedy vybudován pouze na dvou základních pojmech. Díky *klíčům* model přistupuje stejným způsobem k rozhodovacím proměnným konkrétního řešení, k vypočítaným hodnotám účelové funkce, k pořadí v posloupnosti, k parametrům celé metaheuristiky, ke všem typům relací, ke skupinám a vůbec ke všem ukládaným informacím. *Operátory* naopak zajišťují stejný způsob zacházení pro problémově závislé funkce, pro obecně definované operátory, pro skupiny operátorů i pro celé metaheuristiky. Sjednocujícím pohledem na zmíněné aspekty metaheuristik tak vznikl model, který dovoluje snadný vývoj nových hybridních postupů. Stejným postupem se vytváří jak základní typy metaheuristik (genetické algoritmy, ...), tak i jejich významně modifikované varianty (odhad nejhoršího scénáře z [72]) a nebo i schémata s od základu odlišnou filosofií (hyperheuristiky). Kromě hybridizace metaheuristik je stejně tak usnadněna i jejich paralelizace a standardizace experimentálních testů chování. Schopnost popisovat obecným modelem různé metaheuristiky je demonstrována několika implementovanými ukázkami na CD přiloženém k disertační práci.

Celá disertační práce je postavena na zkušenostech autora s úspěšnými aplikacemi metaheuristických algoritmů na inženýrské problémy. Je ale potřeba zdůraznit, že se nejedná o všelék, který vyřeší libovolný problém bez větší námahy. Z publikovaných odborných článků lze často získat pocit, že pomocí několika málo změn řídicích parametrů lze většinu metaheuristik přizpůsobit k nevídané efektivitě. Opak je ale pravdou a u komplikovanějších stavových prostorů je zapotřebí velké trpělivosti při hledání vhodné strategie a systematické zpracování mnoha experimentálních dat, k čemuž může být obecný model velmi dobře nápomocný. V mnoha publikací jsou k ověření kvality navržených algoritmů použity sice komplikované účelové funkce, ale pouze na malých ohraničených množinách. Při vyšších počtech vyhodnocení účelové funkce lze ale dosáhnout obdobných výsledků i zcela náhodným způsobem generování bodů. Metaheuristiky se tak vyplatí používat až opravdu v případě, kdy *jde to tuhého*, tedy například pro úlohy třídy NP-těžká nebo i náročnější problémy z třídy P (např. se složitostí $\mathcal{O}\left(n^{123}\right)$). Metaheuristiky nejsou samospasitelné, ale v mnoha případech již prokázaly svou užitečnost a představují tak dobrou volbu při řešení nesnadných úloh.

6 Použitá literatura

[1] GLOVER F.. Future paths for integer programming and links to artificial intelligence. *Computers and Operational Research*, 13(5), 1986.

[2] REEVES C.. *Modern heuristic techniques for combinatorial problems*., 2005.

[3] DORIGO M. and STUTZLE T.. *Ant colony optimization*. MIT Press, 2004.

[4] OSMAN I. H. and KELLY J. P.. *Meta-heuristics: theory & applications*. Springer, 1996.

[5] VOSS S., MARTELLO S., OSMAN I., and ROUCAIROL C.. *Metaheuristics: Advances and trends in local search paradigms for optimization*. Kluwer Academic Publishers, 1999.

[6] WEBER B.. *Distributed Hybrid Metaheuristics for Optimization*..

[7] TALBI E.-G.. *Metaheuristics: From Design to Implementation*. Wiley Publishing, 2009.

[8] YANG X.-S.. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[9] ALBA E.. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[10] BOSCHETTI M. A., MANIEZZO V., ROFFILLI M. and BOLUFé RöHLER A.. Matheuristics: Optimization, simulation and control. In *Proceedings of the 6th International Workshop on Hybrid Metaheuristics* HM '09, pages 171–177, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] RAIDL G. R.. A unified view on hybrid metaheuristics. In *Proceedings of the Third international conference on Hybrid Metaheuristics* HM'06, pages 1–12, Berlin, Heidelberg, 2006. Springer-Verlag.

[12] VAN LUONG T., TAILLARD E., MELAB N. and TALBI E.-G.. Parallelization strategies for hybrid metaheuristics using a single gpu and multi-core resources. In *Proceedings of the 12th international conference on Parallel Problem Solving from Nature - Volume Part II* PPSN'12, pages 368–377, Berlin, Heidelberg, 2012. Springer-Verlag.

[13] POSPICHAL P.. Gpu-based acceleration of the genetic algorithm. In *Proceedings of the 16th Conference Student EEICT 2010 Volume 5*, pages 234–238, 2010. Faculty of Information Technology BUT.

[14] VEGA-RODRíGUEZ M. A., GóMEZ-PULIDO J. A., ALBA E., VEGA-PéREZ D. and PRIEM-MENDES S. et al.. Evaluation of different metaheuristics solving the rnd problem. In *Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog: Applications of Evolutionary Computing*, pages 101–110, Berlin, Heidelberg, 2007. Springer-Verlag.

[15] DEAN J. and GHEMAWAT S.. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6* OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[16] VERMA A., LLORA X., GOLDBERG D. and CAMPBELL R.. Scaling genetic algorithms using mapreduce. In *Intelligent Systems Design and Applications, 2009. ISDA '09. Ninth International Conference on*, pages 13-18, 2009.

[17] TAGAWA K. and ISHIMIZU T.. Concurrent differential evolution based on mapreduce. *International Journal of Computers*, 4:161-168, 2010.

[18] MOSTAFA H., KHADRAGI A. and HANAFI Y.. Hardware implementation of genetic algorithm on fpga. In *Radio Science Conference, 2004. NRSC 2004. Proceedings of the Twenty-First National*, pages C9-1-9, 2004.

[19] DAS S., BISWAS A., DASGUPTA S. and ABRAHAM A.. Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications. In Ajith Abraham, Aboul-Ella Hassanien, Patrick Siarry and Andries Engelbrecht, editors, *Foundations of Computational Intelligence Volume 3*, number 203 in Studies in Computational Intelligence, pages 23-55. Springer Berlin Heidelberg, 2009.

[20] YANG X.-S.. A new metaheuristic bat-inspired algorithm. In Juan R. GonzÃąlez, David Alejandro Pelta, Carlos Cruz, German Terrazas and Natalio Krasnogor, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, number 284 in Studies in Computational Intelligence, pages 65-74. Springer Berlin Heidelberg, 2010.

[21] KAVEH A. and FARHOUDI N.. A new optimization method: Dolphin echolocation. *Advances in Engineering Software*, 59(0):53 - 70, 2013.

[22] OFTADEH R., MAHJOOB M. and SHARIATPANAHI M.. A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. *Computers and Mathematics with Applications*, 60(7):2087 - 2098, 2010.

[23] GANDOMI A., YANG X.-S. and ALAVI A.. Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with Computers*, pages 1–19, 2011.

[24] CHU S.-C., TSAI P.-w. and PAN J.-S.. Cat swarm optimization. In Qiang Yang and Geoff Webb, editors, *PRICAI 2006: Trends in Artificial Intelligence*, number 4099 in Lecture Notes in Computer Science, pages 854-858. Springer Berlin Heidelberg, 2006.

[25] MOZAFFARI A., FATHI A. and BEHZADIPOUR S.. The great salmon run: a novel bio-inspired algorithm for artificial system design and optimisation. *Int. J. Bio-Inspired Comput.*, 4(5):286–301, 2012.

[26] KAMMERDINER A., MUCHERINO A. and PARDALOS P.. Application of monkey search meta-heuristic to solving instances of the multidimensional assignment problem. In MichaelJ. Hirsch, ClaytonW. Commander, PanosM. Pardalos and Robert Murphey, editors, *Optimization and Cooperative Control Strategies*, number 381 in Lecture Notes in Control and Information Sciences, pages 385-397. Springer Berlin Heidelberg, 2009.

[27] YANG X.-S. and DEB S.. Eagle strategy using levy walk and firefly algorithms for stochastic optimization. In Juan R. Gonzalez, David Alejandro Pelta, Carlos Cruz, German Terrazas and Natalio Krasnogor, editors, *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, number 284 in Studies in Computational Intelligence, pages 101-111. Springer Berlin Heidelberg, 2010.

[28] AFSHAR A., HADDAD O. B., MARINO M. and ADAMS B.. Honey-bee mating optimization (hbmo) algorithm for optimal reservoir operation. *Journal of the Franklin Institute*, 344(5):452 - 462, 2007.

[29] KARABOGA D.. An idea based on Honey Bee Swarm for Numerical Optimization. Technical Report, Erciyes University, 2005.

[30] KARABOGA D. and AKAY B.. A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, 31(1-4):61-85, 2009.

[31] COMELLAS F. and NAVARRO M.. Bumblebees: A multiagent combinatorial optimization algorithm inspired by social insect behaviour. , 2009.

[32] MARINAKIS Y., MARINAKI M. and MATSATSINIS N.. A hybrid bumble bees mating optimization - grasp algorithm for clustering. In Emilio Corchado, Xindong Wu, Erkki Oja, Alvaro Herrero and Bruno Baruque, editors, *Hybrid Artificial Intelligence Systems*, number 5572 in Lecture Notes in Computer Science, pages 549-556. Springer Berlin Heidelberg, 2009.

[33] HEDAYATZADEH R., SALMASSI F., KESHTGARI M., AKBARI R. and ZIARATI K.. Termite colony optimization: A novel approach for optimizing continuous problems. In *Electrical Engineering (ICEE), 2010 18th Iranian Conference on*, pages 553-558, 2010.

[34] KRISHNANAND K. and GHOSE D.. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence*, 3(2):87-124, 2009.

[35] ABIDIN Z., ARSHAD M. and NGAH U.. A simulation based fly optimization algorithm for swarms of mini autonomous surface vehicles application. *Indian journal of goe marine sciences*, 2011.

[36] RASHEDI E., NEZAMABADI-POUR H. and SARYAZDI S.. Gsa: A gravitational search algorithm. *Information Sciences*, 179(13):2232–2248, 2009.

[37] FORMATO R. A.. Central force optimization: A new deterministic gradient-like optimization metaheuristic. *OPSEARCH*, 46(1):25-51, 2009.

[38] EROL O. K. and EKSIN I.. A new optimization method: Big bang-big crunch. *Adv. Eng. Softw.*, 37(2):106–111, 2006.

[39] KRIPKA M. and KRIPKA R. M. L.. Big crunch optimization method. In *EngOpt - International Conference on Engineering Optimization*, 2008.

[40] SHAH T. and HOSSEINI H.. Principal components analysis by the galaxy based search algorithm a novel metaheuristic for continuous optimisation. *Int. J. Comput. Sci. Eng.*, 6(1/2):132–140, 2011.

[41] HATAMLOU A.. Black hole: A new heuristic optimization approach for data clustering. *Information Sciences*, 222(0):175 - 184, 2013. Including Special Section on New Trends in Ambient Intelligence and Bio-inspired Systems.

[42] CUEVAS E., OLIVA D., ZALDIVAR D., PéREZ-CISNEROS M. and SOSSA H.. Circle detection using electro-magnetism optimization. *Inf. Sci.*, 182(1):40–55, 2012.

[43] KAVEH A. and TALATAHARI S.. A novel heuristic optimization method: charged system search. *Acta Mechanica*, 213(3-4):267-289, 2010.

[44] LU H.-j., ZHANG H.-m. and MA L.-h.. A new optimization algorithm based on chaos. *Journal of Zhejiang University SCIENCE A*, 7(4):539-542, 2006.

[45] NARAYANAN A. and MOORE M.. Quantum-inspired genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 61-66, 1996.

[46] CHEN Z. and LUO P.. Qisa: Incorporating quantum computation into simulated annealing for optimization problems. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 2480-2487, 2011.

[47] BISWAS A., MISHRA K. K., TIWARI S. and MISRA A. K.. Physics-inspired optimization algorithms: A survey. *Journal of Optimization*, 2013, 2013.

[48] RABANAL P., RODRIGUEZ I. and RUBIO F.. Using river formation dynamics to design heuristic algorithms. In SelimG. Akl, CristianS. Calude, MichaelJ. Dinneen, Grzegorz Rozenberg and H.Todd Wareham, editors, *Unconventional Computation*, number 4618 in Lecture Notes in Computer Science, pages 163-177. Springer Berlin Heidelberg, 2007.

[49] GAO-WEI Y. and ZHANJU H.. A novel atmosphere clouds model optimization algorithm. In *Computing, Measurement, Control and Sensor Network (CMCSN), 2012 International Conference on*, pages 217-220, 2012.

[50] ESKANDAR H., SADOLLAH A., BAHREININEJAD A. and HAMDI M.. Water cycle algorithm - a novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers and Structures*, 110-111(0):151-166, 2012.

[51] SIMON D.. Biogeography-based optimization. *IEEE Transactions on Evolutionary Computation*, 12, 2008.

[52] YANG X.-S.. Flower pollination algorithm for global optimization. In Jerome Durand-Lose and Natasa Jonoska, editors, *Unconventional Computation and Natural Computation*, number 7445 in Lecture Notes in Computer Science, pages 240-249. Springer Berlin Heidelberg, 2012.

[53] MEHRABIAN A. and LUCAS C.. A novel numerical optimization algorithm inspired from weed colonization. *Ecological Informatics*, 1(4):355 - 366, 2006.

[54] REYNOLDS R. G.. An introduction to cultural algorithms. In *Proceedings of the third annual conference on evolutionary programming*, pages 131–139, 1994. World Scientific.

[55] ATASHPAZ-GARGARI E. and LUCAS C.. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 4661-4667, 2007.

[56] XU Y., CUI Z. and ZENG J.. Social emotional optimization algorithm for nonlinear constrained optimization problems. In BijayaKetan Panigrahi, Swagatam Das, PonnuthuraiNagaratnam Suganthan and SubhransuSekhar Dash, editors, *Swarm, Evolutionary, and Memetic Computing*, number 6466 in Lecture Notes in Computer Science, pages 583-590. Springer Berlin Heidelberg, 2010.

[57] KASHAN A.. League championship algorithm: A new algorithm for numerical function optimization. In *Soft Computing and Pattern Recognition, 2009. SOCPAR '09. International Conference of*, pages 43-48, 2009.

[58] SHI L. and ÓLAFSSON S.. Nested partitions method for global optimization. *Oper. Res.*, 48(3):390–407, 2000.

[59] GLOVER F., LAGUNA M. and MARTI R.. Fundamentals of scatter search and path relinking. *Control and cybernetics*, 39:653–684, 2000.

[60] MLADENOVIC N. and HANSEN P.. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 - 1100, 1997.

[61] LOURENCO H. R., MARTIN O. C. and STÃijTZLE T.. Iterated local search: Framework and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, number 146 in International Series in Operations Research & Management Science, pages 363-397. Springer US, 2010.

[62] MOSCATO P.. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report, California Institute of Technology, 1989.

[63] BATTITI R. and BRUNATO M.. *Reactive Search Optimization: Learning while Optimizing*. Springer Verlag, 2009.

[64] KRINK T., FILIPIC B., FOGEL G. and THOMSEN R.. Noisy optimization problems - a particular challenge for differential evolution?. In *In Proceedings of 2004 Congress on Evolutionary Computation*, pages 332–339, 2004. IEEE Press.

[65] DAS S. and KONAR A.. An improved differential evolution scheme for noisy optimization problems. In SankarK. Pal, Sanghamitra Bandyopadhyay and Sambhunath Biswas, editors, *Pattern Recognition and Machine Intelligence*, number 3776 in Lecture Notes in Computer Science, pages 417-421. Springer Berlin Heidelberg, 2005.

[66] VAESSENS R., AARTS E. and LENSTRA J.. A local search template. *Computers and Operations Research*, 25(11):969 - 979, 1998.

[67] CALéGARI P., CORAY G., HERTZ A., KOBLER D. and KUONEN P.. A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, 5(2):145–158, 1999.

[68] GREISTORFER P. and VOSS S.. Controlled pool maintenance for metaheuristics. In Ramesh Sharda, Stefan Voss, CÃĭsar Rego and Bahram Alidaee, editors, *Metaheuristic Optimization via Memory and Evolution*, number 30 in Operations Research/Computer Science Interfaces Series, pages 387-424. Springer US, 2005.

[69] TAILLARD E. D., GAMBARDELLA L. M., GENDREAU M. and POTVIN J.-Y.. Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1 - 16, 2001.

[70] MEIGNAN D., CREPUT J.-C. and KOUKAM A.. An organizational view of metaheuristics. 2008

[71] MEIGNAN D., KOUKAM A. and CRéPUT J.-C.. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics*, 16(6):859–879, 2010.

[72] ŠANDERA Č., POPELA P. and ROUPEC J.. The worst case analysis by heuristic algorithms. *MENDEL 2009 - 15th International Conference on Soft Computing*, pages 109-114.

7 Curriculum vitae

Osobní informace

příjmení, jméno	Šandera, Čeněk
datum narození:	29.07.1984
adresa:	Ečerova 15, Brno 63500, Česká republika
email:	cenek.sandera@gmail.com

Vzdělání

2008 – dosud:	VUT v Brně, Fakulta strojního inženýrství
obor:	doktorský, Aplikovaná matematika
2003 – 2008:	VUT v Brně, Fakulta strojního inženýrství
obor:	magisterský, Matematické inženýrství
1999 – 2003:	Strojní průmyslová škola strojnická, Brno
obor:	Programování CNC strojů

Zahraniční studium

09/2007 – 07/2008:	University of L'Aquila, Itálie
obor:	magisterský, Matematické modelování ve strojírenství
08/2006 – 12/2006:	Molde University College, Norsko
obor:	jednosemestrální, Optimalizace a logistika

Pracovní zkušenosti

09/2012 – dosud:	Honeywell International, s.r.o.
oddělení:	Advanced Technology, Brno
pozice:	Signal Processing Scientists
02/2012 – 03/2012:	Evropský parlament, Brusel
pozice:	asistent europoslankyně Olgy Sehnalové

Odborné workshopy a semináře

03/2011	PhD winter school, Oppdal, Norsko
zaměření	Managing uncertainty in energy infrastructure investments
07/2011	Intensive Programme, L'Aquila, Itálie
zaměření	Mathematical Models in Seismology
09/2009	Intensive Programme, Alba Adriatica, Itálie
zaměření	Mathematical Models in Life and Social Sciences

Ostatní univerzitní aktivity

09/2005 – 08/2007	Člen studentské komory akademického senátu VUT FSI
09/2008 – 08/2011	Výuka předmětů na VUT FSI (Matematika I, Informatika I)

Abstrakt

Hlavním tématem disertační práce jsou metaheuristické algoritmy v obecnějším pojetí. Úvodní kapitoly se věnují popisu širšího kontextu metaheuristik, tedy různým optimalizačním problémům, určování jejich složitosti a samozřejmě přístupům k jejich řešení. Navazující obsáhlá diskuze o metaheuristikách a jejich typických vlastnostech je následována ukázkami několika vybraných metaheuristických konceptů. Na odpozorovaných vlastnostech je vybudován obecný metaheuristický model vhodný pro vývoj nových i hybridních algoritmů. Programová implementace vytvořeného obecného modelu je přiložena k disertaci na CD a tvoří její nedílnou součást.

Abstract

The main topic of this PhD thesis is metaheuristic algorithms in wider scope. The first chapters are dedicated to a description of broader context of metaheuristics, i.e. various optimization classes, determination of their complexity and different approaches to their solutions. The consequent discussion about metaheuristics and their typical characteristics is followed by several selected examples of metaheuristics concepts. The observed characteristics serve as a base for building general metaheuristics model which is suitable for developing brand new or hybrid algorithms. On the CD attached to the disertation, there is also available a program implementation of the created model.