České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra kybernetiky

Diplomová práce

Evoluční algoritmy pro rozvrhování projektů

Bc. Filip Krejčí

Vedoucí práce: Doc. Dr. Ing. Zdeněk Hanzálek

Studijní program: Otevřená informatika (magisterský)

Obor: Umělá inteligence

12. května 2011

Poděkování

Na tomto místě bych rád poděkoval všem, kteří mi pomohli s realizací diplomové práce. V první řadě děkuji svému vedoucímu práce, panu Doc. Dr. Ing. Zdeněku Hanzálkovi, za přínosné konzultace a velké množství nápadů při řešení problému. Obrovský dík patří mé mamince a tatínkovi, kteří mě podporovali během celého studia, zvláště pak při tvorbě této práce.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 12.5.2011 …….….

Annotation

In this thesis the algorithm Particle Swarm Optimization (PSO) deals with the problem of RCPS | *temp*. A new algorithm GPSO that combines good features of PSO with genetic operators unbalanced two-point crossover and random mutation is proposed. The performance of this algorithm is compared with other variants of PSO and GA on the criteria of energetic efficiency and earliness-tardiness.

The concept of energetic efficiency is entirely new in the field of project scheduling with temporal constraints. The optimization is achieved by reducing the number of machines' switching on and off and shortening delays between tasks, in which must be the machines turned on and vice versa creating a sufficiently long delays, so the device to be put into standby mode.

Designed GPSO outperforms other tested variants of PSO and GA. For some test sets GPSO improved the energetic efficiency up to 25 %. The criterion of earliness-tardiness was improved up to 6.7 %, while simultaneously the length of the schedule (C_{max}) was reduced up to 3.7 %.

Keywords: RCPS | *temp*, PSO, GA, GPSO, energetic efficiency, earliness-tardiness

Anotace

Práce se zabývá řešením problému RCPS | *temp* metodou Particle Swarm Optimization (PSO). Vytvořil jsem nový algoritmus GPSO, který kombinuje dobré vlastnosti PSO s genetickými operátory nevyváženého dvoubodového křížení a náhodné mutace. Srovnávám výkonnost tohoto nového algoritmu a různých variant PSO a GA na kritériích energetické efektivity a earliness-tardiness.

Koncept kritéria energetické efektivity je v oblasti rozvrhování projektů s temporálními omezeními zcela nový. Jeho optimalizací se dosahuje redukování počtu zapínání resp. vypínání zařízení a zkracování prodlev mezi úlohami, ve kterých musí být zařízení zapnuté a naopak vytváření dostatečně dlouhých prodlev, aby se zařízení mohlo přepnout do úsporného režimu.

V rozsáhlých testech překonává navržené GPSO ostatní testované varianty PSO i GA. Pro některé testovací množiny dosahuje zlepšení energetické efektivity až o 25 %. Kritérium earliness-tardiness vylepšuje až o 6,7 %, přičemž zároveň zkracuje i délku rozvrhu (C_{max}) až o 3,7 %.

Klíčová slova: RCPS | *temp*, PSO, GA, GPSO, energetická efektivita, earliness-tardiness

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V dnešní době se stále více potvrzuje, že zdroje nejsou nevyčerpatelné a že je třeba s nimi šetřit. Nejde jen o zdroje surovin, ale o zdroje v obecnějším smyslu (lidské zdroje, finanční zdroje, atp.) Jejich spotřeba může být buď fixní, nebo variabilní. Fixní spotřebu nelze ovlivnit (např. materiál nutný při stavbě domu), zatímco variabilní spotřeba závisí na okolnostech (např. mzda dělníků odvíjející se od délky procesu stavby, prostoje). Variabilní zdroje je možné ušetřit (např. vhodným rozvrhnutím jednotlivých činností je možné zkrátit celý proces stavby domu a tím ušetřit peníze). Díky této možnosti je na oblast rozvrhování projektů upřena velká pozornost.

V praxi se velmi často řeší problém rozvrhování projektů s omezenými zdroji, např. při vývoji software, ve stavebnictví, ve strojírenství, při výrobě elektronických spotřebičů, při míchání laků, atp. Efektivní rozvrh může ušetřit zdroje a zvýšit tak např. výrobní kapacitu linky. Ve velkých závodech může zlepšení výrobního procesu v řádu jednotek procent přinést velkou úsporu nákladů nebo zisk.

Při stále se rozšiřujícím počtu oborů uplatnění rostou také nároky na vyjadřovací možnosti jazyka, popisujícího rozvrhovanou doménu. Základní jednotkou rozvrhování je úloha, zastupující konkrétní činnost a mající určité parametry (čas zpracování, nároky na zdroje, atp.). Některé domény vystačí s relacemi následnosti mezi jednotlivými úlohami, kdy zpracování každé úlohy může začít nejdříve po dokončení všech předchůdců dané úlohy. Dalším nástrojem jsou release time a deadline, které určují čas, kdy nejdříve může začít zpracování jednotlivé úlohy a kdy nejpozději musí být zpracování dokončeno. V doménách kde již tyto možnosti nedostačují, musí být použita temporální omezení, changeover časy nebo take-give zdroje (podrobněji popsáno v kapitole 2.2).

Optimálně vyřešit problém znamená najít platný rozvrh (splňující všechna omezení), který je nejlepší z hlediska zvoleného kritéria (spotřeby zdrojů). Bohužel rozvrhovací problémy patří většinou do kategorie NP-těžkých úloh, pro které je velmi těžké najít kvalitní řešení a ani není obecně možné dokázat optimalitu nalezeného řešení (vybraná kritéria kvality jsou popsána v kapitole 2.4).

V současné době má katedra k dispozici IRS heuristiku, která je výborná v řešení rozvrhovacích problémů s temporálními omezeními včetně changeover časů a take-give zdrojů [5] s ohledem na kritérium minimalizace délky rozvrhu. IRS heuristika

byla úspěšně použita na řešení problémů míchání laků [17], který zahrnuje jak temporální omezení, tak i changeover časy a take-give zdroje. V jiných kritériích si však příliš dobře nevedla.

Mým úkolem je studovat schopnosti algoritmu Particle Swarm Optimization (PSO) na problému rozvrhování projektů s temporálními omezeními (RCPS | *temp*) a různými kritérii (přesná definice problému je popsána v následující kapitole). V případě úspěchu může tento PSO nahradit IRS heuristiku pro určitou množinu problémů s různými kritérii (např. energetická efektivita).

Kapitola 2

Popis řešeného problému

Jak již bylo naznačeno v úvodu, rozvrhování projektů s omezenými zdroji a temporálními omezeními (RCPS | *temp*) je NP-těžká úloha [18]. Nejčastěji je řešen problém RCPS | *temp* | C_{max} (dle standardní $\alpha \mid \beta \mid \gamma$ notace, kde α charakterizuje zdroje, β úlohy a γ kritérium), kde v tomto případě je kritériem délka rozvrhu. Metoda Branch and Bound (větví a mezí) řeší tyto problémy optimálně, ale pouze pokud mají méně než 100 úloh [5]. Pro řešení větších problémů je nutné použít heuristiku.

V této kapitole nejprve definuji řešený problém a provedu jeho rozbor: Vymezím typy zdrojů, ilustruji možnosti temporálních omezení, definuji jednotlivá kritéria a nastíním prostor řešení problému.

2.1 Definice problému

Problém rozvrhování projektu s omezenými zdroji a temporálními omezeními je definován takto: Je dáno n nepreemptivních úloh $V = \{1, 2, ..., n\}$. Obvykle je „na začátek“ a „na konec“ přidána dummy úloha s nulovou délkou (nultá a $n+1$ úloha). Nultá úloha předchází všem úlohám, $n+1$ úloha může být vykonána až po dokončení všech ostatních úloh. Tyto dvě dummy úlohy slouží např. k zakódování relativních časů vůči začátku nebo konci rozvrhu nebo k omezení maximální délky rozvrhu.

Úlohy využívají zdroje, které jsou k dispozici v omezeném množství. Je dána množina obnovitelných zdrojů R, každý s kapacitou $r_i \in \mathbf{N}_{>0} \cup \{+\infty\}$. Každá úloha $i \in V$ má přiřazený čas zpracování $p_i \in \mathbf{N}_{\geq 0}$ a nároky na jednotlivé zdroje c_{ik}, $k \in R$, které využívá po celý čas zpracování p_i. Žádná úloha nesmí být během zpracování pozastavena. Úlohy jsou vzájemně svázány temporálními omezeními, zapsanými do matice $W_{n \times n}$. Hodnota $W_{ij} \in \mathbf{Z} \cup \{-\infty\}$ v matici W určuje vztah mezi startovními časy s_i *a* s_j úloh i a j takto:

$$s_j - s_i \geq W_{ij} \tag{2.1}$$

Jednoduše řečeno, nezáporná hodnota W_{ij} říká, že úloha j může začít nejdříve W_{ij} po začátku úlohy i. Naopak, záporná hodnota W_{ij} značí, že úloha i může začít dříve nebo maximálně o $|W_{ij}|$ později, než začalo zpracování úlohy j. Každá dvojice úloh i a j je svázaná dvěma rovnicemi, které vychází ze vztahu (2.1):

$$s_i + W_{ij} \leq s_j \tag{2.2}$$

$$s_j + W_{ji} \leq s_i \tag{2.3}$$

Hodnota $W_{ij} = -\infty$ „vypíná“ rovnici (2.2) (rovnice je splněna pro libovolné hodnoty s_i a s_j). Prakticky to znamená, že není definován nejzazší čas s_i vůči času s_j úlohy j, kdy má být úloha i zařazena. Obdobně hodnota $W_{ji} = -\infty$ „vypíná“ rovnici (2.3). Pokud pro některé $i \neq j$ platí $W_{ij} = W_{ji} = -\infty$, znamená to, že mezi úlohami i a j není žádné omezení.

Problém lze znázornit orientovaným grafem bez smyček. Matici temporálních omezení W převedeme na graf tak, že úlohy budou představovat vrcholy grafu a temporální omezení (hodnoty W_{ij}) délku hran grafu. Každý vrchol je ohodnocen časem zpracování úlohy p_i a nároky na jednotlivé zdroje c_{ik}.

Cílem je najít rozvrh určený vektorem s^* startovních časů všech úloh tak, aby byla hodnota kritéria co nejmenší. Např. pro kritérium C_{max} hledáme vektor s^* minimalizující délku rozvrhu:

$$s^* = \arg\min_{s \in R^n} \max_{i \in 1..n} (s_i + p_i) \tag{2.4}$$

přičemž nesmí dojít k porušení žádného omezení. Prvním omezením je kapacita zdrojů, což znamená, že v žádném okamžiku t ($0 \leq t \leq C_{max}$) pro žádný zdroj k, nesmí být využito více jednotek, než je jeho kapacita r_k. Druhým jsou výše popsaná temporální omezení, kdy pro každou uspořádanou dvojici úloh i a j platí vztah (2.1).

Instance problému *RCPS | temp |* C_{max} je definována trojicí ($V(p, c)$, R, W), kde každá úloha množiny V má přiřazen čas zpracování a nároky na jednotlivé zdroje. Pro jiná kritéria než je C_{max} musí být tato trojice rozšířena o další informace.

2.2 Klasifikace zdrojů (α)

Resource Constrained Project Scheduling (RCPS) je nejobecnější klasifikace zdrojů pro rozvrhovací problémy a zahrnuje velké množství rozdílných charakteristik. Pro přesnější popis zdrojů jsem využil interně používaný formát: pětiznakový řetězec, přičemž každá pozice je vyhrazena pro konkrétní vlastnost. Pokud problém danou vlastnost nemá, je příslušná pozice označena písmenem X. Jednotlivé vlastnosti podle pozic v řetězci jsou:

1. C – Changeover čas je minimální doba mezi dvěma bezprostředně následujícími úlohami i a j zpracovávanými na stejné jednotce zdroje, po kterou nesmí být zdroj využíván. Tento čas je určen pouze konkrétními úlohami i a j a pro různou

kombinaci se může lišit [5]. Tento čas slouží např. pro výměnu nástroje, nebo pro čištění nádoby při míchání laků.

2. D – Dedicated processors značí výskyt různých typů zdrojů, přičemž každá úloha má přesně stanoven typ zdroje, na kterém musí být vykonána.
3. M – Multiprocessor tasks jsou úlohy, které si pro své vykonání mohou nárokovat více typů (nebo jednotek) zdrojů současně.
4. P – Parallel identical processors značí, že od každého typu zdroje může být k dispozici více totožných jednotek.
5. P, O – Take-give resources jsou speciální přídavné zdroje, jejichž využívání není přímo závislé na čase. Fungují tak, že jedna úloha při svém startu zabere take-give resource a využívá ho nejen během času zpracování *p*, ale pokračuje v jeho užívání, dokud ho některá jiná úloha svým dokončením neuvolní. Take-give zdroje umožňují např. simulaci blokujících operací, kdy je stroji znemožněna další práce, dokud není jiná konkrétní úloha dokončena. Toho se využívá např. ve výrobě, jestliže mezi jednotlivými stroji nejsou žádné skladovací prostory [5].

V této práci se budu soustředit pouze na některé charakteristiky zdrojů a nebudu se zabývat changeover časy a take-give zdroji. V pořadí od nejomezenější po nejobecnější se zaměřím na tyto vybrané charakteristiky:

- XDXXX – několik typů zdrojů, každý pouze s unární kapacitou.
- XDXPX – několik typů zdrojů, každý zdroj může mít jinou kapacitu.
- XDMXX – několik typů zdrojů, každý pouze s unární kapacitou, avšak úlohy jsou multiprocesorové.
- XDMPX – několik typů zdrojů, každý zdroj může mít jinou kapacitu a úlohy jsou multiprocesorové.

2.3 Temporální omezení (β)

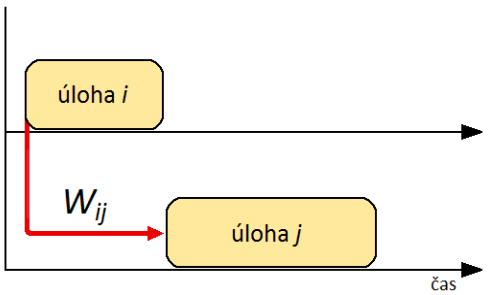
V této sekci budu demonstrovat možnosti temporálních omezení. Jako podklad jsem použil přednášku z předmětu Kombinatorická optimalizace [15], kde je tato problematika velice přehledně vysvětlena. Z tohoto materiálu jsem rovněž čerpal i příklady. Podle způsobu užití lze rozdělit temporální omezení do těchto 5 kategorií:

- $W_{ij} = p_i$
- $W_{ij} > p_i$
- $0 < W_{ij} < p_i$
- $W_{ij} = 0$
- $W_{ij} < 0$

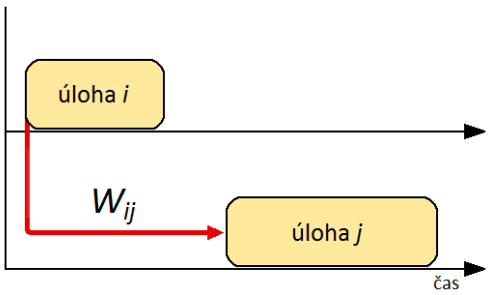
Nejprve budu charakterizovat jednotlivé kategorie a následně uvedu příklady, jak lze provázat úlohy *i* a *j* za použití obou hodnot W_{ij} a W_{ji}.

Hodnota $W_{ij} = p_i$ reprezentuje klasickou relaci následnosti, kdy úloha j může začít nejdříve okamžikem dokončení úlohy i (obr. 1). Např. vybetonování základů stavby může být realizováno až po dokončení výkopů.

Hodnota $W_{ij} > p_i$ představuje situaci, kdy mezi dokončením úlohy i a začátkem úlohy j musí být určitá prodleva (obr. 2). Např. po vybetonování základů stavby musí být určitá časová prodleva pro zatuhnutí betonu. Teprve pak je možné začít se zděním obvodového zdiva. Dalším příkladem (z článku [15]) může být z prostředí lakovny: Po nalakování výrobku (úloha i) musí lak nejprve zaschnout, než bude možné výrobek zabalit (úloha j).



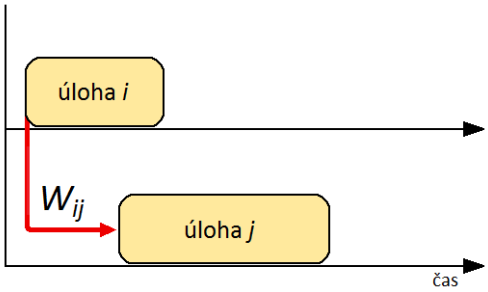
Obr. 1: Znázornění situace $W_{ij} = p_i$



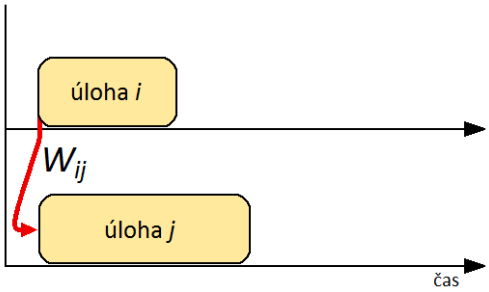
Obr. 2: Znázornění situace $W_{ij} > p_i$

W_{ij} takové, že $0 < W_{ij} < p_i$ se užívá v situaci, kdy úloha j může začít nejdříve určitý čas po začátku úlohy i (obr. 3). Např. při demoličních pracech může začít odvoz suti (úloha j) až určitý čas od zahájení demolice (úloha i), avšak nemusí se čekat, až bude demolice (rozlehlé stavby) dokončena celá.

Hodnota $W_{ij} = 0$ reprezentuje stav, kdy úloha úloha j nesmí začít dříve, než začne zpracování úlohy i (obr. 4). Např. odvoz posekané trávy nezačne dříve než sekání trávy.



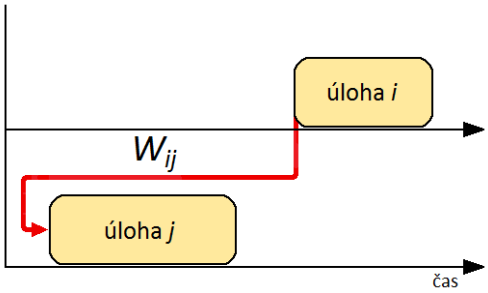
Obr. 3: Znázornění situace $0 < W_{ij} < p_i$



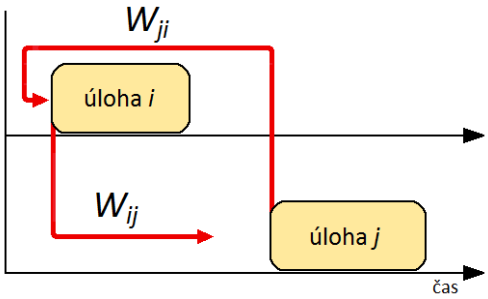
Obr. 4: Znázornění situace $W_{ij} = 0$

Hodnota $W_{ij} < 0$ realizuje situaci, kdy úloha i může být vykonána dříve, nebo nejdéle o $|W_{ij}|$ později než začne zpracování úlohy j (obr. 5). Toto omezení představuje relativní deadline úlohy i vůči úloze j.

Výše uvedené typy temporálních omezení lze kombinovat. Kombinací $W_{ij} > 0 > W_{ji}$, $|W_{ij}| < |W_{ji}|$ vznikne záporný cyklus, kterým se vytvoří relativní časové okno o velikosti $|W_{ji}| - W_{ij}$, uvnitř kterého musí být úloha j zařazena (obr. 6). Velikost časového okna (volnost zařazení úlohy j) se rovná délce záporného cyklu. Existuje-li cyklus s délkou větší než 0, problém nemá řešení. Např. čerstvě nahozená omítka poskytuje určitou volnost v čase, kdy musí být zahájen proces štukování.



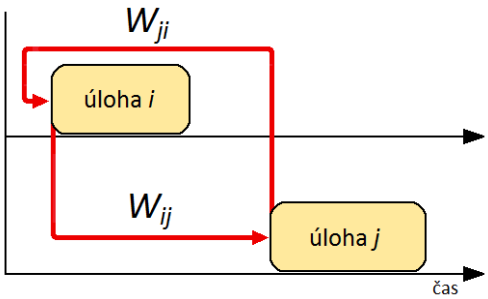
Obr. 5: Znázornění situace $W_{ij} < 0$



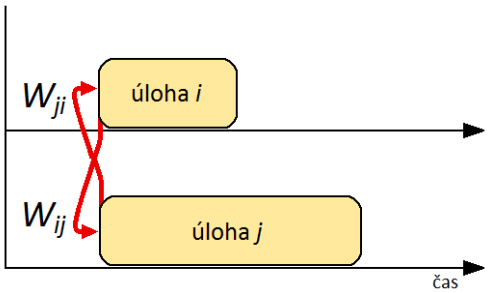
Obr. 6: Znázornění situace $0 < W_{ij} < p_i$

V případě, že časový rozdíl mezi zahájením dvou úloh musí být konstantní, lze tyto úlohy svázat omezením $W_{ij} = -W_{ji}$ = požadované konstantě (obr. 7). Např. po spuštění varovného signálu musí dojít za přesně určenou dobu k vypnutí stroje. Další možností je stav, kdy obě úlohy tvoří jednu úlohu s mezičasem, kdy není zdroj využíván a rozdělením této úlohy je mezičas dán k dispozici jiným úlohám. Speciálním případem je $W_{ij} = -W_{ji} = 0$, kdy je požadováno, aby úlohy byly zahájeny současně (obr. 8).

Svázáním úlohy napevno s nultou úlohou lze simulovat situaci, kdy má být úloha vykonána v přesný čas od začátku rozvrhu. Časovým oknem vůči nulté úloze lze také simulovat relase time a deadline vybrané úlohy.



Obr. 7: Znázornění situace $W_{ij} = -W_{ji}$



Obr. 8: Znázornění situace $W_{ij} = -W_{ji} = 0$

2.4 Kritéria (γ)

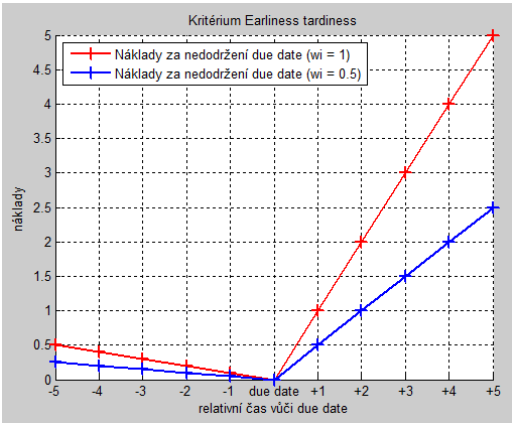
V praxi se často setkáváme s různými hodnotícími kritérii, určujícími kvalitu rozvrhu. V této práci se budu podrobněji zabývat těmito třemi: C_{max}, earliness-tardiness (ET) a energetickou efektivitou (EE).

Nejčastěji používaným kritériem je C_{max}, které se snaží minimalizovat délku rozvrhu (makespan). Čím dříve skončí poslední úloha, tím je rozvrh kvalitnější. Jak již bylo řečeno v kapitole 2.1, kritérium C_{max} lze zapsat jako:

$$s^* = \arg\min_{s \in R^n} \max_{i \in 1..n}(s_i + p_i) \tag{2.5}$$

kde část $\max_{i \in 1..n}(s_i + p_i)$ vyjadřuje konec poslední úlohy. Toto kritérium nalézá uplatnění téměř v jakémkoliv odvětví.

Earliness-tardiness vychází z problému minimalizace nákladů souvisejících s pokutou za pozdní dodání (tardiness) a nákladů na zbytečné skladování zboží (earliness). Prioritou je zařadit úlohu včas, avšak nikoli zbytečně brzy, protože pokuty za pozdní dodání zboží bývají výrazně vyšší než skladovací náklady. Každá úloha i má přiřazen požadovaný termín dokončení (due date) d_i a důležitost (váhu) úlohy w_i. Dále je specifikována konstanta λ_i určující poměr earliness vůči tardiness. Většinou bývá poměr 1:10 (náklady na skladování jsou 10x menší než pokuta za pozdní dodání), tedy $\lambda_i = 0{,}1$ pro všechna i. Náklady na každou úlohu rostou lineárně se vzdáleností času jejího dokončení od due date (obr. 9).



Obr. 9: Lineární růst nákladů na jednu úlohu vzhledem ke vzdálenosti času dokončení od due date pro hodnoty $w_i = 1$ a $w_i = 0{,}5$ za předpokladu $\lambda_i = 0{,}1$.

Optimálním řešení úlohy je vektor s^*, který je určen výrazem:

$$s^* = \arg\min_{s \in R^n} \sum_{i \in 1..n} \left[w_i \cdot \max(s_i + p_i - d_i, \lambda_i \cdot (d_i - (s_i + p_i))) \right] \tag{2.6}$$

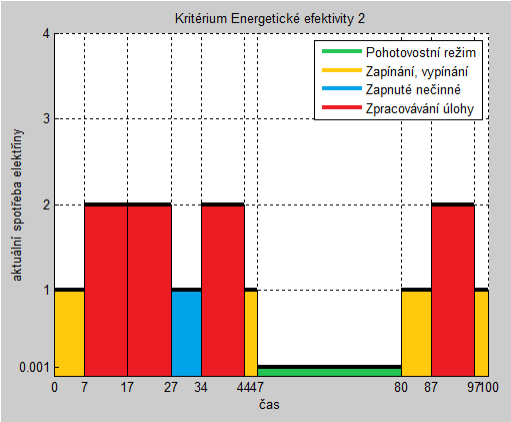
Argumenty funkce max jsou zpoždění a předstih vynásobený λ_i. Z těchto argumentů bude vždy právě jeden kladný nebo oba nulové. Pronásobením vybraného argumentu hodnotou w_i dostaneme náklady na úlohu.

Kritérium energetické efektivity se uplatňuje v oblasti cyklického rozvrhování, kdy např. spolu pravidelně komunikují autonomní zařízení. Každé zařízení může být

buď zapnuté aktivní (odesílá nebo přijímá zprávu), zapnuté nečinné, nebo v pohotovostním režimu. Pouze zapnuté zařízení může přijímat nebo vysílat zprávy. Úsporný režim výrazně šetří baterie a prodlužuje tak dobu funkčnosti zařízení. Zapnuté nečinné zařízení spotřebovává méně energie, než když je aktivní, ale stále výrazně více než v pohotovostním režimu. Přechod z nebo do pohotovostního režimu trvá určitou dobu a spotřebovává energii. Energeticky efektivní je seskupit úlohy do bloků, v rámci kterých je minimální doba nečinnosti, přičemž mezi jednotlivými bloky je dostatečná prodleva, aby se zařízení mohlo přepnout do úsporného režimu. Při cyklickém rozvrhování nejsou používána žádná temporální omezení, jsou pouze stanoveny délky intervalů, ve kterých musí probíhat pravidelná komunikace mezi jednotlivými zařízeními. Pro rozvrhování s temporálními omezeními nebylo zatím kritérium energetické efektivity použito.

Vytvořil jsem dvě verze kritéria energetické efektivity EE1 a EE2. První z nich spočítalo délky nečinných časů mezi úlohami pro všechny jednotky všech zdrojů. Nečinné časy dlouhé alespoň 10 jednotek byly započítány jako pohotovostní režim. K této hodnotě přičte součet intervalů kratších než 10 jednotek započítaných jako zapnuté nečinné. Spotřeba energie při vlastním provádění úloh je konstantní pro zvolený problém, a proto jsem ji do kritéria nezapočítal. Tato verze měla dva problémy: Zanedbávala energii potřebnou na přechod mezi oběma režimy, a tak algoritmus neúměrně prodlužoval délku rozvrhu, protože se snažil mezi každé po sobě následující úlohy vložit dostatečně dlouhou dobu, aby mohla být započítána jako úsporný režim. Druhý problém spočíval v komplikovaném vyhodnocování optimálního přiřazení úloh na jednotlivé jednotky zdrojů s kapacitou větší než 1.

Kritérium EE2 bylo navrženo tak, aby reflektovalo i čas, nutný k přechodu mezi režimy, čímž odpadl i druhý problém při EE1. Kritérium se snaží nejen minimalizovat dobu, kdy je zařízení zapnuté a nečinné, ale také počet zapínání a vypínání zařízení. Na obr. 10 je znázorněna aktuální spotřeba pro každou činnost. Hodnota kritéria je dána součtem integrálů od 0 do C_{max} z aktuální spotřeby energie pro všechny jednotky všech zdrojů.

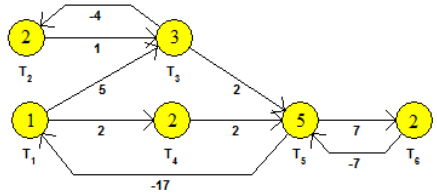


Obr. 10: Spotřeba energie na jedné jednotce zdroje. Červeně je znázorněna spotřeba při zpracovávání úlohy, oranžově přechody z nebo do úsporného režimu, zeleně úsporný režim a růžově nečinné zařízení.

Parametry EE2 byly stanoveny následovně: Přechod z úsporného režimu (zapínání) trvá 7 jednotek, zatímco přechod do úsporného režimu (vypínání) trvá 3 jednotky. Zpracování úlohy je 2x energeticky náročnější než přechody z nebo do úsporného režimu nebo než zapnuté nečinné zařízení. Zapnuté nečinné zařízení je energeticky 1000x náročnější než zařízení v pohotovostním režimu. Stanovené hodnoty byly inspirovány tabulkou na slidu č. 9 v prezentaci [16].

2.5 Prostor řešení

Prostor řešení budu demonstrovat pro dvě jednoduché instance problému se zdroji typu XXXXX (1 zdroj s unární kapacitou) a šesti úlohami (n = 6). První instance je definována grafem na obr. 11, pro kterou jsem zvolil kritérium C_{max}.

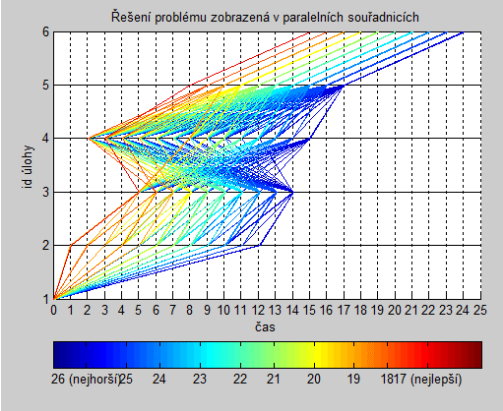


Obr. 11: První instance problému. Úlohy jsou označeny $T_{1\text{-}6}$, čas zpracování je vepsán v uzlu, hodnoty omezení jsou na hranách grafu.

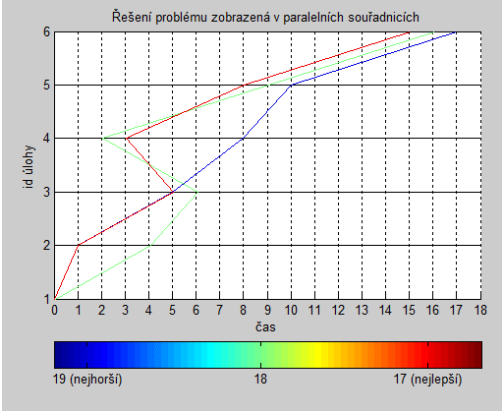
Pro přehlednost jsem omezil délku rozvrhu na 26 jednotek a omezil startovní čas první prováděné úlohy na 0. Pro zobrazení vektorů jsem použil metodu paralelních souřadnic, protože jsem ji považoval pro daný účel za nejpřehlednější (oproti např. PCA). Na svislou osu vynáším dimenze problému (všechny úlohy) a na vodorovnou osu hodnoty startovních časů úloh. Jeden vektor řešení je křivka protínající každou vodorovnou přímkou, která představuje úlohu, právě jednou. Pořadí průsečíků je od první do poslední úlohy. Křivka protíná každou z těchto přímek v bodě, který odpovídá startovnímu času této úlohy.

Temporální omezení a omezení kapacity zdroje splňují 3 permutace indexů úloh, ze kterých lze vygenerovat 784 různých platných celočíselných vektorů startovních časů. Všech 784 vektorů je zobrazeno na obr. 12. Hodnota kritéria C_{max} je znázorněna odpovídající barvou. Z obrázku je patrná pevná vazba úloh 5 a 6, kdy jednomu startovnímu času úlohy 5 přísluší právě jeden startovní čas úlohy 6. Dále je patrné časové okno mezi úlohami 2 a 3. Přestože tento problém obsahuje velké množství omezení, vzniká díky relativní volnosti zařazení některých úloh obrovské množství různých vektorů startovních časů.

Díky charakteristice kritéria C_{max} lze jednoduše z pořadí úloh určit optimální startovní časy pro zvolené pořadí úloh (lze využít algoritmus popsaný v kapitole 4.2.1). Vektor spočítaný algoritmem má jednotlivé startovní časy co nejmenší (úlohy zarovnané vlevo) a je zaručeno, že po nalezení optimálního pořadí úloh bude vektor startovních časů optimální (pro kritérium C_{max}). Proto je možné místo konkrétních startovních časů hledat permutace a redukovat tak prohledávaný prostor. Optimální vektor pro každou permutaci (v tomto případě jsou 3) je zobrazen na obr. 13 a je shodný s optimálním vektorem na obr. 12.

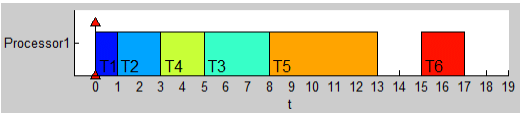


Obr. 12: Zobrazení všech řešení 1. problému, jejichž zpracování začíná v čase 0 a končí nejpozději v čase 26 pro kritérium C_{max}. Vektory jsou barevně odlišeny podle hodnoty kritéria C_{max} od nejhoršího (modrý) po nejlepší (červený).



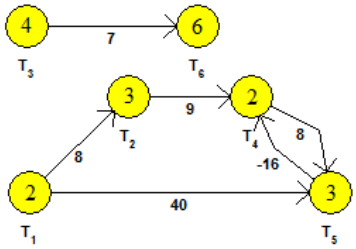
Obr. 13: Zobrazení optimálního řešení 1. problému pro každou ze tří platných permutací úloh.

Rozvrh vytvořený z optimálního vektoru startovních časů je na obr. 14.

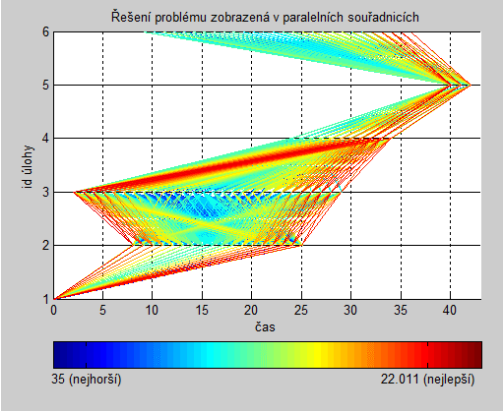


Obr. 14: Rozvrh vytvořený z optimálního vektoru startovních časů.

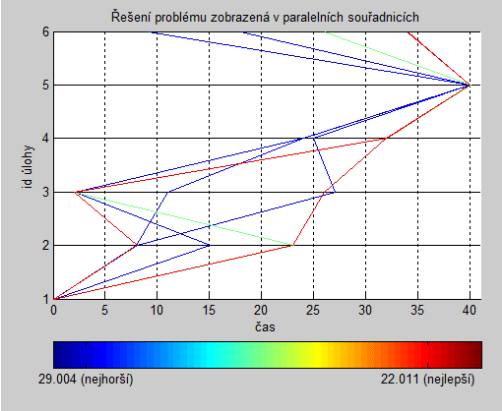
Druhá instance je definována grafem na obr. 15. Zvolil jsem pro ni kritérium EE2 a omezil maximální délku rozvrhu na 45. Temporální omezení a omezení kapacity zdroje v tomto případě splňuje 6 permutací indexů úloh, ze kterých lze vygenerovat 49 953 různých platných celočíselných vektorů startovních časů. Při zvětšení maximální délky rozvrhu o 1 vzroste tento počet o 23 130. Tento relativně velký počet vektorů startovních časů je způsoben větší volností temporálních omezení než v prvním příkladu. Startovní časy jsou zobrazeny na obr. 16. Barevně je odlišena hodnota kritéria EE2. Obr 17. zobrazuje nejlepší vektor startovních časů pro každou ze 6 permutací. Jsou na něm patrny 2 různé vektory se stejnou hodnotou kritéria, patřícím k různým permutacím úloh.



Obr. 15: Druhá instance problému. Úlohy jsou označeny $T_{1\text{-}6}$, čas zpracování je vepsán v uzlu, hodnoty omezení jsou na hranách grafu.



Obr. 16: Zobrazení všech řešení 2. problému, jejichž zpracování začíná v čase 0 a končí nejpozději v čase 45 pro kritérium EE 2. Vektory jsou barevně odlišeny podle hodnoty kritéria EE 2 od nejhoršího (modrý) po nejlepší (červený).



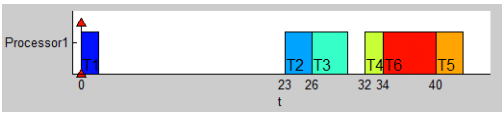
Obr. 17: Zobrazení optimálního řešení problému pro každou z šesti platných permutací úloh.

Instance má celkem 5 optimálních řešení s hodnotou kritéria 22,011. Startovní časy jsou v tabulce 1.

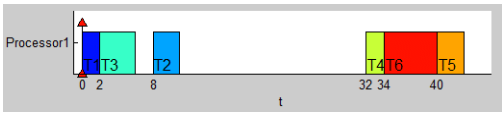
úloha 1	úloha 2	úloha 3	úloha 4	úloha 5	úloha 6
0	23	26	32	40	34
0	23	27	32	40	34
0	8	2	32	40	34
0	8	3	32	40	34
0	8	4	32	40	34

Tabulka 1: Startovní časy všech 5 optimálních řešení s hodnotou kritéria 22,011.

Startovní časy lze rozdělit do dvou skupin, v rámci každé z nich se jednotlivé vektory liší pouze startovním časem úlohy č. 3. První skupina obsahuje vektor v prvním řádku tabulky 1, jehož rozvrh je zobrazen na obr. 18. Druhá skupina obsahuje vektor ve třetím řádku téže tabulky, jehož rozvrh zobrazuje obr. 19. Tyto vektory nemají startovní časy co nejmenší (zarovnané vlevo), jako tomu bylo pro kritérium C_{max}, a nenašel jsem žádné jiné pravidlo umožňující z permutace snadno vytvořit optimální vektor startovních časů. Složitost problému určení optimálních startovních časů se zvyšuje s přidáním dedikovaných (D) nebo paralelních procesorů (P). Z toho důvodu je nutné, aby algoritmus při řešení problémů s tímto kritériem hledal přímo vektor startovních časů, nikoli pouze permutaci úloh, jako tomu je pro C_{max}. Obdobný problém přináší kritérium earliness-tardiness.



Obr. 18: Rozvrh vytvořený z vektoru startovních časů [0, 23, 26, 32, 40, 34].



Obr. 19: Rozvrh vytvořený z vektoru startovních časů [0, 8, 2, 32, 40, 34].

Kapitola 3

Particle Swarm Optimization

Particle swarm optimization (PSO) patří mezi biologicky inspirované algoritmy, které napodobují chování živých organismů. Pracují s populací jedinců, kteří si vzájemně vyměňují informace, vyvíjejí se v čase a hledají řešení problému. Každý jedinec představuje řešení, jehož kvalita je určena hodnotou kriteriální funkce nazývané fitness. Algoritmus se snaží najít globální minimum této funkce. Biologicky inspirované algoritmy (GA, PSO, ACO) zaznamenávají velké úspěchy při řešení NP-těžkých problémů, protože provádí paralelně optimalizaci z více míst prostoru. Jejich velkou výhodou je snadná použitelnost na různá kritéria – stačí pouze změnit fitness funkci. Jejich nevýhodou je většinou suboptimalita nalezeného řešení a větší časová náročnost.

3.1 Rozbor algoritmu PSO

Základním principem PSO algoritmu je „hejno“ částic, které se pohybují po spojitém vícedimenzionálním prostoru řešení. V našem případě prostor není spojitý, a proto bylo nutné provést určité změny. Částice v „hejnu“ mají mezi sebou sociální vztahy, které tvoří relaci sousednosti. Relace sousednosti je vytvořena při spuštění algoritmu a během výpočtu se nemění. Řídí se většinou pořadím při generování částic, skutečná poloha v prostoru a jejich vzájemná blízkost není rozhodující. Každé částici tedy přísluší množina sousedů. Kardinalita (hustota vztahů) a struktura (topologie vztahů) těchto množin má výrazný vliv na chování algoritmu.

Každá částice i se nachází na určité pozici x_i v prostoru a má přiřazenou rychlost v_i (velocity), což je vektor, směřující od předchozí pozice k aktuální. Každá částice i má také paměť, ve které je uchována její doposud nejlepší navštívená pozice p_i (particle-best) a nejlepší pozice g_i nalezená někým ze sousedů (neighborhood-best). Často se využívá plně propojená topologie (množina sousedů každé částice obsahuje všechny částice) a tedy neighborhood-best všech částic je shodná a bývá označena global-best. Pozice částice se mění v čase, přičemž její novou pozici ovlivňuje její aktuální pozice a vektory particle-best a neighborhood-best. Pseudokód algoritmu PSO jsem převzal z článku [12] a po drobných úpravách vypadá následovně:

1 **for** $i = 1$: počet částic
2 inicializuj (x_i)
3 inicializuj (v_i)
4 $p_i = x_i$
5 **end**
6 **for** $i = 1$: počet částic
7 $g_i = \min(p_{sousedi(i)})$
8 **end**
9 **do**
10 **for** $i = 1$: počet částic
11 **for** $d = 1$: počet dimenzí
12 $v_i^d = \varphi_0 \cdot v_i^d + \varphi_1 \cdot rand_1^d \cdot (p_i^d - x_i^d) + \varphi_2 \cdot rand_2^d \cdot (g_i^d - x_i^d)$
13 $v_i^d = sign(v_i^d) \cdot \min(abs(v_i^d), v_{\max}^d)$
14 $x_i^d = x_i^d + v_i^d$
15 **end**
16 **if** $f(x_i) < f(p_i)$ **then** $p_i = x_i$
17 **end**
18 **for** $i = 1$: počet částic
19 $g_i = \min(p_{sousedi(i)})$
20 **end**
21 **while** není splněna ukončovací podmínka

Řádky 1-8 náhodně inicializují počáteční pozice částic x_i, jejich rychlosti v_i a dále určí vektory p_i a g_i. Následuje hlavní cyklus, který se provádí, dokud není splněna ukončovací podmínka. Tou může být např. počet vykonání hlavního cyklu (počet pohybů všech částic), počet vyčíslení fitness funkce, uplynutí časového limitu, nebo dosažení požadované fitness.

Každá složka vektoru částice se mění nezávisle na ostatních složkách. Nejprve se spočítá nová složka rychlosti v_i^d (*d.* složka vektoru rychlosti částice *i*), (řádek 12), která je daná váženým součtem těchto tří faktorů:

- setrvačností ve stávajícím pohybu $\varphi_0 \cdot v_i^d$, kde $0 \leq \varphi_0 \leq 1$, přičemž doporučená hodnota je $\varphi_0 = 0{,}729844$ [6],
- náhodně váženou vzdáleností od particle-best pozice $\varphi_1 \cdot rand_1^d \cdot (p_i^d - x_i^d)$, kde $rand_1^d$ je náhodné číslo náhodně vybrané z intervalu (0; 1) a φ_1 je konstanta s doporučenou hodnotou $\varphi_1 = 1{,}49618$ [6],
- náhodně váženou vzdáleností od neighborhood-best pozice $\varphi_2 \cdot rand_2^d \cdot (g_i^d - x_i^d)$, kde $rand_2^d$ je náhodné číslo náhodně vybrané z intervalu (0; 1) a φ_2 je konstanta s doporučenou hodnotou $\varphi_2 = 1{,}49618$ [6].

Pro takto vypočtenou v_i^d se ověří, zda její absolutní hodnota není větší než maximální změna $v_{\max}^d$ pro aktuálně zpracovávanou dimenzi *d* (*d.* složku vektoru *x*). V případě, že absolutní hodnota v_i^d překračuje $v_{\max}^d$, je nutné v_i^d upravit tak, aby $abs(v_i^d) = abs(v_{\max}^d)$

(řádek 13). Doporučená hodnota $v_{\max}^{d}$ je 10 – 20 % rozsahu přípustných hodnot pro dimenzi *d*. Pak se provede update složky pozice x_i^d (řádek 14).

Po úpravě celého vektoru x_i se spočítá fitness. V případě, že je lepší než fitness particle-best, je proveden update. Po vyčíslení všech částic se provede update i neighborhood-best pozic.

Často se v literatuře [6] objevuje zkrácená, více matematická, rekurentní formulace, která pracuje s vektory *x* a *v* jako s celkem a která využívá při své definici diskrétní čas *t*. Nový vektor *x* částice *i* (v čase *t* + 1) je vypočítán ze současné pozice a z rychlosti této částice (v čase *t*), rovnice 3.1. Nová rychlost (v čase *t* + 1) je pak daná rovnicí 3.2, kde r_1 a r_2 jsou vektory náhodných čísel.

$$x_i(t+1) = x_i(t) + v_i(t) \tag{3.1}$$

$$v_i(t+1) = \varphi_0 \cdot v_i(t) + \varphi_1 \cdot r_1(t) \cdot [p_i - x_i(t)] + \varphi_2 \cdot r_2(t) \cdot [g_i - x_i(t)] \tag{3.2}$$

Dále shrnu výhody a nevýhody PSO algoritmu. Hlavní výhody algoritmu jsou:

- sdílení informací (particle-best pozic) mezi částicemi,
- různé topologie umožňují nastavit rychlost šíření informace o nejlepších particle-best pozicích (uložených v paměti částice jako neighborhood-best) a zabránit tak předčasné konvergenci,
- změnou parametrů φ_0, φ_1, φ_2 je možné docílit výrazně odlišného chování.

Algoritmus má i své nevýhody:

- všem částicím je poskytován stejný výpočetní čas, nejsou upřednostňovány lepší částice ve slibnějších částech prohledávaného prostoru,
- částice nevyužívá informaci, zda aktuálním posunem došlo ke zlepšení či zhoršení aktuální hodnoty kritéria, nelze tak opakovat krok „úspěšným“ směrem,
- algoritmus nezjišťuje ani jinak nereflektuje, zda změna v konkrétní dimenzi kritérium zhoršuje či zlepšuje, používá kriterální funkci jako blackbox,
- určen primárně pro spojitý prostor,
- vyžaduje vhodné nastavení parametrů φ_0, φ_1, φ_2.

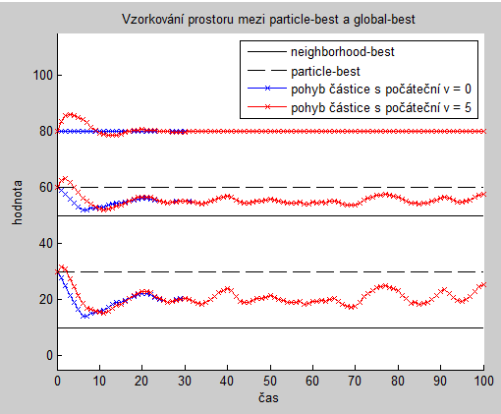
3.2 Diskuse chování algoritmu

V této sekci budu diskutovat, jak nastavení jednotlivých parametrů ovlivňuje chování algoritmu. Rozbor této problematiky je nutný k hlubšímu pochopení chování algoritmu, abych mohl určit vhodné nastavení parametrů, nebo naopak vydedukovat z výsledků jaké chování je nevhodné.

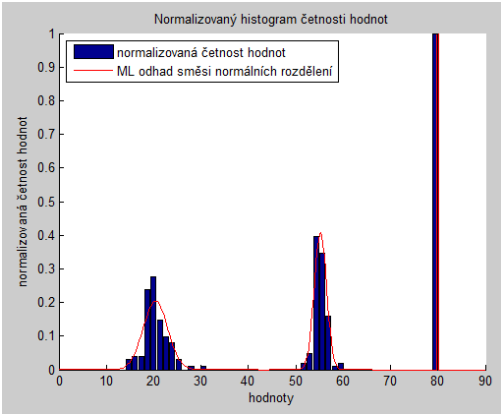
Nejprve jsem simuloval prostředí s jednou částicí o 3 složkách a zobrazil jsem její pohyb vzhledem k času odděleně v každé dimenzi (obr. 20). Testoval jsem dvě

různá nastavení počáteční rychlosti: modrá křivka a červená křivka. Částici jsem inicioval na pozici x = [30, 60, 80] s rychlostmi v = [0, 0, 0] (modrá křivka) a v = [5, 5, 5] (červená křivka). Startovní pozice x byla po celou dobu testu také particle-best pozicí (p = [30, 60, 80]). Neighborhood-best pozice byla po celou dobu testu g = [10, 50, 80]. Obr. 20 také ukazuje chování pro různé vzdálenosti hodnot vektorů particle-best od neighborhood-best, které jsou [20, 10, 0]. Parametry φ jsem určil takto: φ_0 = 0,729844, $\varphi_1 = \varphi_2$ = 0,149618 (φ_1 a φ_2 mají hodnotu 10x menší než doporučenou článkem [6]). Pro srovnání jsem provedl ještě druhý test, jen s rozdílným nastavením parametrů φ (doporučených článkem [6]), které bylo φ_0 = 0,729844, $\varphi_1 = \varphi_2$ = 1,49618.

Testy ukazují, že částice nepravidelně osciluje mezi hodnotami particle-best a neighborhood-best se zvyšující se četností směrem ke středu intervalu, přičemž vzdálenost particle-best a neighborhood-best určuje pouze svislé měřítko a nikoli charakteristiku průběhu. Daleko větší vliv na průběh pohybu mají hodnoty parametrů φ. Tyto parametry určují charakteristiku vzorkovaných bodů (obr. 21 a obr. 23). Zatímco první varianta vzorkuje hodnoty blízko středu hodnot particle-best a neighborhood-best (obr. 21), druhá varianta má rozptyl daleko větší a často vzorkuje i hodnoty mimo interval určený hodnotami particle-best a neighborhood-best.

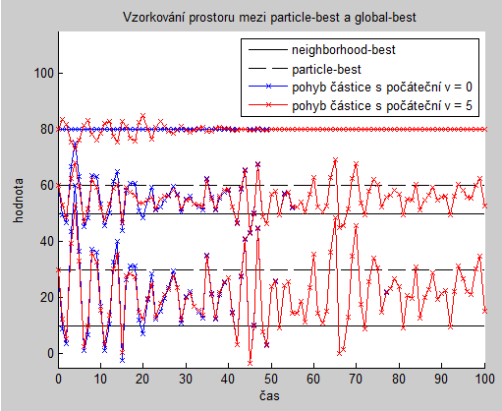


Obr. 20: Vzorkování prostoru mezi particle-best a neighborhood-best pozicí pro částici o 3 složkách a parametrech φ_0 = 0,73, $\varphi_1 = \varphi_2$ = 0,15 (10x menší než na obr. 22)

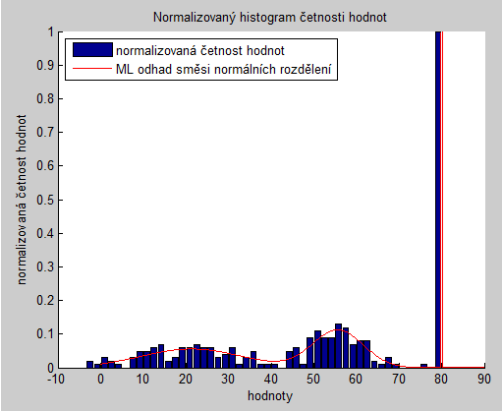


Obr. 21: Histogram četnosti vzorkovaných hodnot pro částici o 3 složkách a parametrech φ_0 = 0,73, $\varphi_1 = \varphi_2$ = 0,15 (10x menší než na obr. 23)

Náhodná inicializace rychlosti v má na částici vliv pouze ze začátku. Především z obr. 22 je patrné, že náhodná inicializace zvětšuje amplitudu kmitů, přičemž postupem času se přibližuje k trajektorii určené částicí s nulovou počáteční rychlostí, až s ní nakonec splyne. Také lze pozorovat, že pro vyšší hodnoty φ_1 a φ_2 (obr. 22) trvá proces splynutí o něco déle. Náhodná inicializace může mít velký význam v případě $x_i^d = p_i^d = g_i^d$ (shodnosti některé složky všech tří vektorů), kde jedině tato inicializace umožní proces explorace v dané dimenzi.



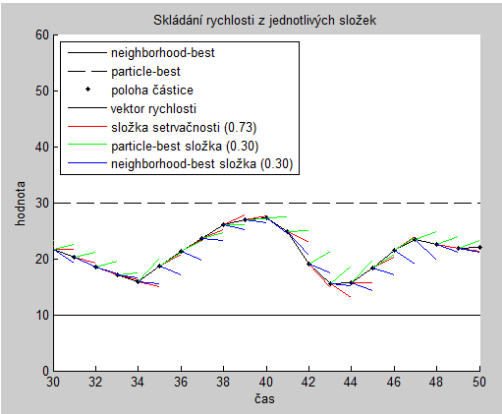
Obr. 22: Vzorkování prostoru mezi particle-best a neighborhood-best pozicí pro částici o 3 složkách a parametrech φ_0 = 0,73, φ_1 = φ_2 = 1,50.



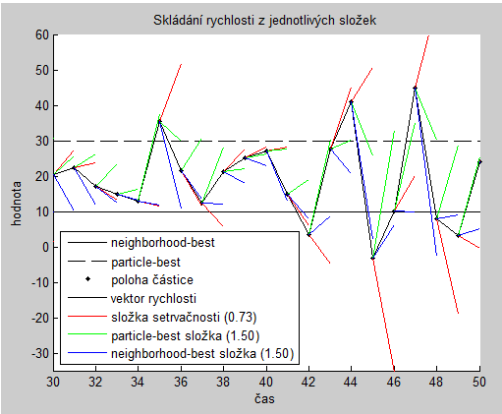
Obr. 23: Histogram četnosti vzorkovaných hodnot pro částici o 3 složkách a parametrech φ_0 = 0,73, φ_1 = φ_2 = 1,50.

Dále mě zajímalo, jak je výsledná rychlost ovlivněna jednotlivými faktory (setrvačnost, particle-best, neighborhood-best). Omezil jsem se pouze na jednu (první) dimenzi a na časový interval [30; 50] a ponechal hodnoty particle-best = 30, neighborhood-best = 10. Použil jsem dvě různá nastavení parametrů. Protože byl pohyb částice v prvním nastavení minulém testu příliš malý, vynásobil jsem pro další testy hodnoty parametrů φ_1 a φ_2 dvěma, tj. φ_0 = 0,73, φ_1 = φ_2 = 0,30 (obr. 24). Druhé nastavení jsem ponechal na doporučených hodnotách (obr. 25).

První varianta (obr. 24) se pohybuje v prostoru mezi best hodnotami velmi pomalu a mění směr pohybu jen velmi zřídka. V čase 35 a 36 jsou vygenerovány relativně velké vektory v opačném směru než je poslední pohyb, ale ke změně směru nedojde, protože byly převáženy součtem setrvačnosti a vektoru směřujícího k particle-best pozici. Změna směru se uskutečňuje postupně a to pouze po dostatečném přiblížení



Obr. 24: Analýza skládání jednotlivých faktorů do nového vektoru pohybu. Barevně jsou odlišeny jednotlivé složky: setrvačnost, particle-best, neighborhood-best a černě pak výsledný vektor. Hodnota parametrů je φ_0 = 0,73, φ_1 = φ_2 = 0,30.



Obr. 25: Analýza skládání jednotlivých faktorů do nového vektoru pohybu. Barevně jsou odlišeny jednotlivé složky: setrvačnost, particle-best, neighborhood-best a černě pak výsledný vektor. Hodnota parametrů je φ_0 = 0,73, φ_1 = φ_2 = 1,50.

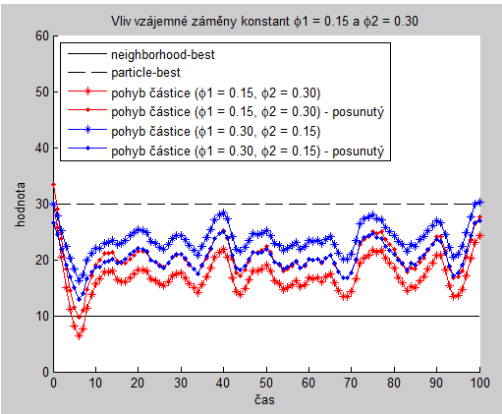
k best pozici. Tuto variantu méně ovlivňuje velikost náhodně generovaných čísel.

Druhá varianta (obr. 25) generuje některé i značně velké vektory (velikost může být i 1,5 násobek vzdálenosti mezi best pozicemi a v případě, že je aktuální pozice vně best intervalu, i větší). Díky velikosti vektorů snadno překonává hranice particle- a neighborhood-best hodnot. Nepřekročí je však příliš, protože následně vektory směřující k oběma best pozicím míří stejným směrem a snadno překonají i velký vektor setrvačnosti, např. v čase 45. Často také nastane kyvadlový efekt, kdy po překonání jedné best pozice dojde k sečtení vektorů a bezprostřednímu překonání i druhé best pozice (čas 44). Divergenci zabraňuje vhodně stanovený parametr φ_0. Přestože se zdají oba běhy značně odlišné, chovají se v některých místech podobně, např. obě změní v časech 34 a 40 svůj vektor pohybu stejným směrem.

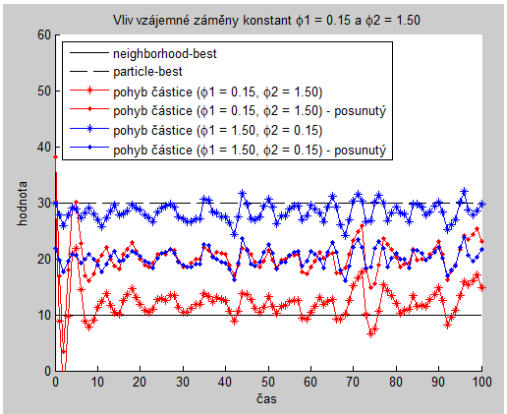
Dále jsem studoval situaci, když bude hodnota φ_1 různá od φ_2. Zvolil jsem jejich hodnoty v poměru 1:2 (obr. 26) a 1:10 (obr. 27) – přesné hodnoty jsou v legendě obrázků. Pro lepší srovnání jsem vzniklé trajektorie vzájemně překryl přičtením posunu (3.3) způsobeného poměrem parametrů (označeno jako „posunutý“).

$$x_{i\_posunutý} = x_i + (p_i - g_i) \cdot (0{,}5 - \varphi_1 / (\varphi_1 + \varphi_2) \tag{3.3}$$

Pro nižší hodnoty φ (obr. 26) jsou křivky téměř totožné, ale pokud je jedna z hodnot výrazně vyšší (obr. 27), pak se trajektorie liší a mezi časy 97 a 100 je rozdíl i ve směru. Částice také více kmitá a pohyb je méně pravidelný. Různý poměr hodnot φ_1 a φ_2 tedy nezpůsobuje jen posun trajektorie, ale může částečně měnit i její charakter.

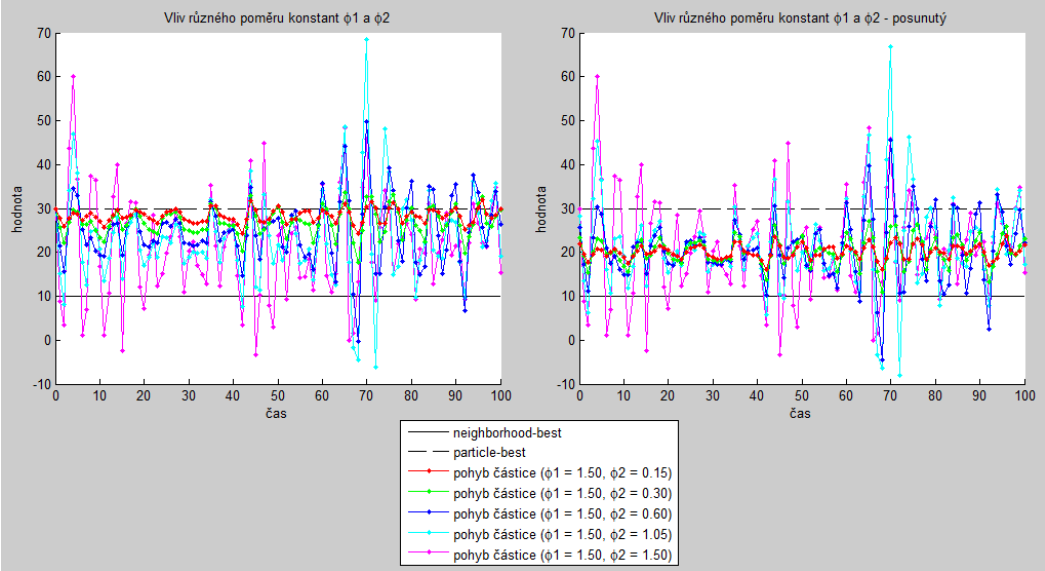


Obr. 26: Vliv záměny hodnot φ_1 a φ_2, které jsou v poměru 1:2. Trajektorie označené „posunutý“ jsou oproštěny o posun způsobený poměrem parametrů.



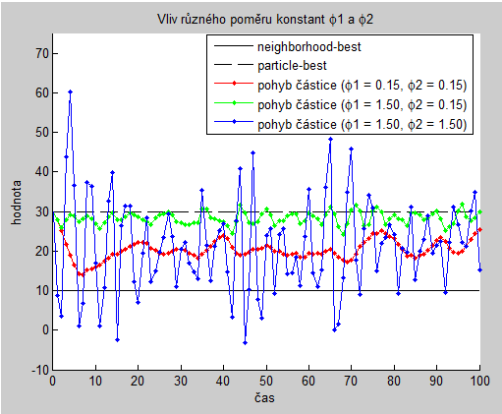
Obr. 27: Vliv záměny hodnot φ_1 a φ_2, které jsou v poměru 1:10. Trajektorie označené „posunutý“ jsou oproštěny o posun způsobený poměrem parametrů.

Při dalším testovacím výpočtu jsem zafixoval hodnotu φ_1 = 1,50 a hodnoty φ_2 jsem vybíral z intervalu [0,15; 1,50]. Trajektorie částice jsou na obr. 28 vlevo a pro přehlednost jsou posunuté na střed intervalu (obr. 28 vpravo). Je patrné, že s rostoucí hodnotou φ_2 se vektory rychlosti nejen prodlužují, ale od hodnoty φ_2 = 0,60 (modrá křivka) vznikají nové změny směru.

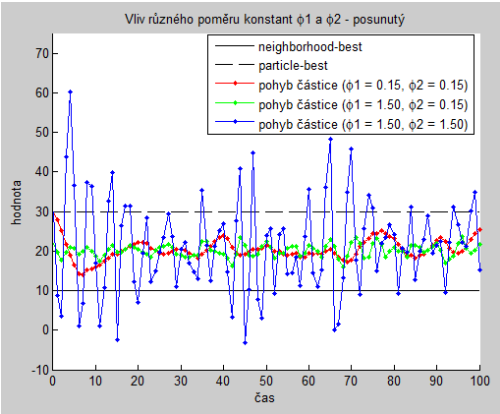


Obr. 28: Trajektorie částice vzniklé nastavením hodnoty φ_1 = 1,50 a různými nastavením hodnoty φ_2 v rozsahu [0,15; 1,50] (vlevo) a posunuté ke středu intervalu [particle-best; neighborhood-best] (vpravo)

Na obr. 29 a jeho posunuté variantě na obr. 30 je vidět, že pro kombinaci různě velkých hodnot φ_1 a φ_2 je pro charakter pohybu určující spíše vyšší hodnota a pro amplitudu nižší hodnota. Nastavením jedné hodnoty φ vyšší a následným posunem křivky do středu je možné docílit chování, vykazující častější změny směru vektoru rychlosti se zachováním amplitudy, která je učená nižší hodnotou.

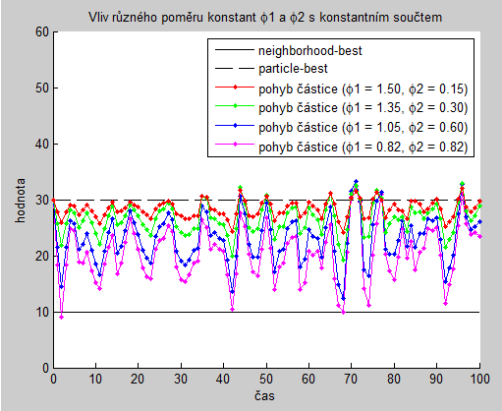


Obr. 29: Trajektorie částice vzniklé nastavením těchto kombinací parametrů: φ_1 = 0,15, φ_2 = 0,15; φ_1 = 1,50, φ_2 = 0,15 a φ_1 = 1,50, φ_2 = 1,50.

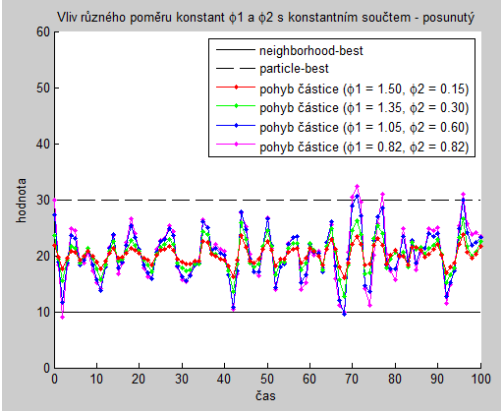


Obr. 30: Trajektorie z obr. 29 posunuté ke středu intervalu.

Obr. 31 a jeho posunutá varianta obr. 32 ukazují, že charakter pohybu je spíše než větším z parametrů ovlivňován jejich součtem. Potvrzuje se, že amplitudu určuje nižší hodnota.

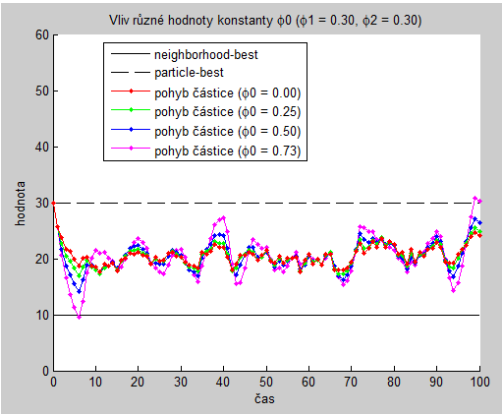


Obr. 31: Vliv různých kombinací parametrů φ_1 a φ_2 s konstantním součtem 1,65.

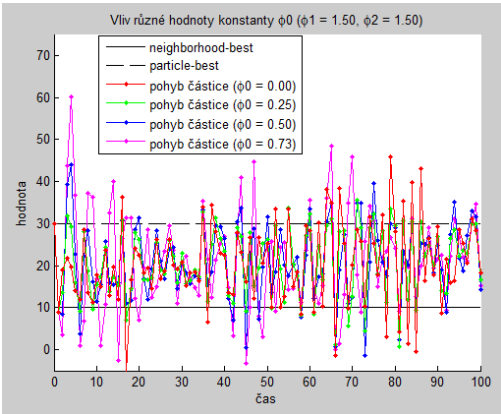


Obr. 32: Trajektorie z obr. 31 posunuté ke středu intervalu.

Posledním testovaným parametrem byl φ_0. Trajektorie pro různé hodnoty φ_0 v rozsahu [0; 0,73] jsou zobrazeny obr. 33 a obr. 34. Rozdíl mezi oběma obrázky je v nastavení ostatních parametrů φ: pro obr. 33 je $\varphi_1 = \varphi_2 = 0{,}30$ zatímco pro obr. 34 je $\varphi_1 = \varphi_2 = 1{,}50$. V prvním případě zvětšování φ_0 jen zvětšuje amplitudu a to tím více, čím blíže je částice best hodnotám. Ve druhém případě (pro větší hodnoty φ_1 a φ_2) jsou rozdíly znatelnější. S menšími hodnotami φ_0 jsou změny směru vektoru rychlosti častější, přičemž je zajímavé, že pro setrvačnost $\varphi_0 = 0$ a $\varphi_0 = 0{,}73$ je amplituda srovnatelná. Pro vyšší hodnoty setrvačnosti φ_0 dochází k občasné divergenci a dočasnému zvětšení amplitudy nad únosnou mez.



Obr. 33: Chování algoritmu pro různé hodnoty parametru φ_0 (0; 0,25; 0,5; 0,73) a hodnoty parametrů $\varphi_1 = \varphi_2 = 0{,}30$.



Obr. 34: Chování algoritmu pro různé hodnoty parametru φ_0 (0; 0,25; 0,5; 0,73) a hodnoty parametrů $\varphi_1 = \varphi_2 = 1{,}50$.

Pro vzdálenost δ particle-best od neighborhood-best ($\delta = 20$) jsem rozdělil obor hodnot částice do 11 disjunktních intervalů:

- 4 intervaly vznikly jako „vnitřní blízké“ a „vnější blízké“ hodnoty best pozic (do vzdálenosti $0{,}25 \cdot \delta$), tj. [5; 10), [10; 15), [25; 30) a [30; 35),

- 1 interval jako „středové“ hodnoty (do vzdálenosti 0,25 · δ od středu best hodnot), tj. [15; 25),
- 2 intervaly jako „mezní“ hodnoty (do vzdálenosti δ od best hodnot), tj. [-10; 5) a [35; 50),
- 2 intervaly jako „vzdálené“ hodnoty (do vzdálenosti 6 · δ od best hodnot), tj. [-110; -10) a [50; 150),
- 2 intervaly jako „nepoužitelné“ hodnoty tj. hodnoty (-∞; -110) a naopak hodnoty [150; +∞).

Časté vzorkování „vzdálených“ nebo „nepoužitelných“ hodnot není vhodné chování algoritmu. Zamezit vzorkování těchto hodnot může vhodné nastavení parametru v_{max}. Na druhou stranu, příliš časté využívání limitu v_{max} však může způsobit nežádoucí chování algoritmu.

Pro každé nastavení φ_0 jsem algoritmus spustil na 100 000 kroků a zaznamenával četnosti vzorkování jednotlivých intervalů. Výsledky pro $\varphi_1 = \varphi_2 = 0{,}30$ jsou v tabulce 2, zatímco pro $\varphi_1 = \varphi_2 = 1{,}50$ v tabulce 3. Z tabulky 2 je vidět, že ke vzorkování hodnot dochází téměř výhradně ve středovém intervalu, což značně snižuje explorační schopnost algoritmu a není tolik žádané.

Mnohem zajímavější výsledky jsou obsaženy v tabulce 3. Pro $\varphi_0 = 0{,}9$ algoritmus diverguje. Pro $\varphi_0 = 0{,}8$ je uvnitř intervalu best hodnot pouze 40% vzorkovaných hodnot. Nastavení $\varphi_0 = 0{,}73$ se jeví ještě příliš vysoké, protože ve „vzdálených“ intervalech se nalézá celých 8 % hodnot. Zajímavé je, že pro $\varphi_0 = 0{,}25$ ($0 < \varphi_0 < 0{,}50$) je četnost hodnot v „mezních“ a „vzdálených“ intervalech výrazně menší než pro $\varphi_0 = 0$ i než pro $\varphi_0 = 0{,}5$. Mezi výsledky pro $\varphi_0 = 0$, $\varphi_0 = 0{,}25$ a $\varphi_0 = 50$ však není příliš velký rozdíl. Všechny vzorkují 38-40 % hodnot ve „středovém“ intervalu, 48-53 % v intervalech blízkých, 6-11 % v „mezních“ intervalech a méně než 1 % v intervalech „vzdálených“ a „nepoužitelných“. Tato nastavení se zdají být velmi slibná.

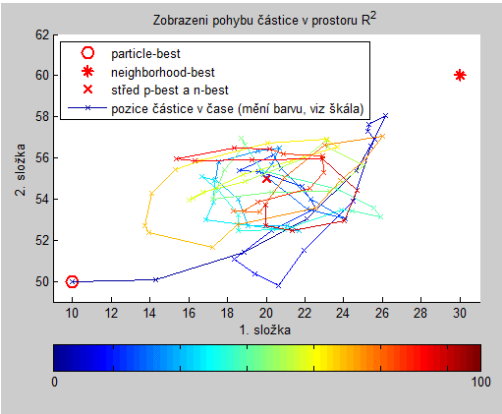
φ_0	< -110	[-110; -10)	[-10; 5)	[5; 10)	[10; 15)	[15; 25)	[25; 30)	[30; 35)	[35; 50)	[50; 150)	≥ 150
0	0	0	0	0	140	99738	122	1	0	0	0
0,25	0	0	0	0	518	98970	512	1	0	0	0
0,50	0	0	0	0	1951	96086	1962	2	0	0	0
0,73	0	0	0	181	6505	86570	6586	158	1	0	0
0,8	0	0	18	700	9350	79786	9487	640	20	0	0
0,9	0	10	949	4099	14391	60951	14631	4017	942	11	0

Tabulka 2: Četnost hodnot v jednotlivých intervalech pro různé hodnoty parametru φ_0 a pro $\varphi_1 = \varphi_2 = 0{,}30$. Zeleně je označen vnitřek intervalu, modře vhodné hodnoty vně intervalu a červeně příliš vzdálené hodnoty.

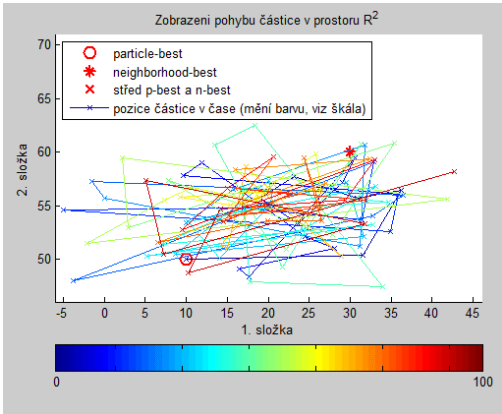
φ_0	< -110	[-110;-10)	[-10; 5)	[5; 10)	[10; 15)	[15; 25)	[25; 30)	[30; 35)	[35; 50)	[50; 150)	≥ 150
0	2	457	4924	7891	16160	40937	16502	7781	4888	459	0
0,25	0	39	2994	9489	16891	40981	17111	9452	3004	40	0
0,50	0	315	5551	9450	15673	37859	15806	9362	5688	297	0
0,73	100	3927	11351	8875	11897	27670	11882	8879	11275	4047	98
0,8	902	8857	12750	7674	9246	21005	9413	7634	12536	9066	918
0,9	45146	2538	1025	435	469	1032	464	405	968	2495	45024

Tabulka 3: Četnost hodnot v jednotlivých intervalech pro různé hodnoty parametru φ_0 a pro $\varphi_1 = \varphi_2 = 1{,}50$. Zeleně je označen vnitřek intervalu, modře vhodné hodnoty vně intervalu a červeně příliš vzdálené hodnoty.

Dále jsem zobrazil pohyb částice ve dvoudimenzionálním prostoru, pro snazší představu, jak se pohyby v jednotlivých dimenzích skládají. Na obr. 35 provádí částice relativně uspořádaný pohyb, připomínající situaci, kdy po velmi nerovném povrchu vyšleme železnou kuličku, která je přitahována slabým magnetem uprostřed. Na obr. 35 je pohyb velmi neuspořádaný a je těžké ho k něčemu přirovnat.



Obr. 35: Chování částice ve dvoudimenzionálním prostoru pro hodnotu parametrů $\varphi_0 = 0{,}73$, $\varphi_1 = \varphi_2 = 0{,}30$. Čas reprezentuje barva vektoru rychlosti měnící se od modré po červenou (viz. škála).



Obr. 36: Chování částice ve dvoudimenzionálním prostoru pro hodnotu parametrů $\varphi_0 = 0{,}25$, $\varphi_1 = \varphi_2 = 1{,}50$. Čas reprezentuje barva vektoru rychlosti měnící se od modré po červenou (viz. škála).

Na závěr shrnu poznatky, ke kterým jsem došel:

- Vzdálenost particle-best od neighborhood-best nemá žádný vliv na charakteristiku vzorkovaných bodů.
- Náhodná inicializace má vliv pouze ze začátku, je však jediným faktorem, který umožňuje exploraci prostoru v případě, že $x_i^d = p_i^d = g_i^d$ pro některé d.
- Pro hodnoty parametrů $\varphi_1 = \varphi_2 = 0{,}30$ vykazuje částice pozvolný pohyb soustředěný kolem středu intervalu [particle-best; neighborhood-best], zatímco pro hodnoty $\varphi_1 = \varphi_2 = 1{,}50$ realizuje částice velké posuny v po sobě následujících časech a snadno tak vzorkuje i body vně best intervalu.

- Pro $\varphi_1 \neq \varphi_2$ dochází k posunu trajektorie pohybu blíže k best pozici příslušející větší z hodnot φ, přičemž se může měnit i charakter pohybu. Charakter pohybu (četnost změn směru) určuje velikost součtu obou parametrů, zatímco amplitudu určuje nižší hodnota, která působí jako brzda pohybu.
- Vhodný rozsah hodnot parametru φ_0 (v kombinaci s $\varphi_1 = \varphi_2 = 1{,}50$) je interval [0; 0,73]. Pro hodnoty 0; 0,25 a 0,50 vykazuje algoritmus podobné chování, tj. téměř stejnou četnost vzorkování jednotlivých intervalů a vybírá si nejen hodnoty uvnitř best intervalu, ale i blízko za jeho hranicemi. Pro hodnotu $\varphi_0 = 0{,}73$ je však již 8 % hodnot značně vzdálených.

Zajímavá nastavení parametr φ jsou:

- $\varphi_0 = 0{,}73$, $\varphi_1 = \varphi_2 = 1{,}50$ (doporučené článkem [6]),
- $\varphi_0 = 0{,}73$, $\varphi_1 = \varphi_2 = 0{,}30$ (menší váhy particle-best a neighborhoood-best),
- $\varphi_0 = 0{,}25$, $\varphi_1 = \varphi_2 = 1{,}50$ (menší setrvačnost),
- $\varphi_0 = 0{,}9$, $\varphi_1 = \varphi_2 = 0{,}30$ (větší setrvačnost a menší váhy particle-best a neighborhoood-best).

3.3 Existující algoritmy

V oblasti rozvrhování projektů jsou často využívány GA [14, 1, 2, 10] a algoritmy PSO [3, 4, 6, 7]. Pro rozvrhování s temporálními omezeními nejsou nasazovány tak často [14, 8, 9], jako pro jednodušší problémy s relacemi následnosti [1, 2, 6] nebo pro problémy s kritériem earliness-tardiness [4, 7], kdy jsou všechny úlohy k dispozici v čase 0. Nejčastěji užívaným kritériem bývá C_{max} [1, 2, 3, 6]. Dalším kritériem může být flexibilita rozvrhu, tj. schopnost rozvrhu absorbovat změny a promptně na ně reagovat [8]. Dále podrobněji přiblížím přínosy jednotlivých prací.

Zajímavý výzkum byl realizován v oblasti GA. Článek [2] srovnává tři různé metody reprezentace jedinců: permutaci, priority value a priority rule. Permutace obsahuje na *i*-té pozici vektoru *x* pořadí, v jakém má být úloha *i* zařazena. Priority value (též activity list) je reprezentace, obsahující *n*-posloupnost reálných čísel z intervalu (0; 1), reprezentujících prioritu zařazení dané úlohy. Při konstrukci rozvrhu z hodnot priority value se v každém kroku vybírá ze všech zařaditelných úloh ta, která má nejvyšší prioritu. Vybrané úloze se přiřazuje co nejnižší čas startu, za dodržení kapacity zdrojů a relací následnosti. Při užívání reprezentace priority value se vyhneme problémům, které přináší křížení permutací. Reprezentace priority rule šlechtí pouze pravidla výběru úlohy, tj. jak ze všech aktuálně zařaditelných úloh vybrat jednu (např. pravidlo LST – úloha s nejnižší hodnotou nejzazšího možného startu, MTS – největší počet následníků, ...).

Pro jednotlivé metody byly navrženy různé mutace: pro permutaci výměna sousedních úloh, pro prority value vygenerování nové priority a pro priority rule zvolení nové priority. Testováno bylo jednobodové, dvoubodové a uniformní křížení. Metody výběru přežívajících jedinců do další generace byly: výběr těch nejlepších, proporcionální výběr (pravděpodobnost přežití jedince je přímo úměrná jeho fitness) a výběr turnajem (vybráni 2 nebo 3 jedinci, kteří se utkají v turnaji a jedinec s nejhorší fitness nepřežije). Článek srovnával výkonnost při generování počáteční populace

náhodně a heuristikou. Testoval i různé velikosti populace, kterým umožnil 1000 vyčíslení fitness funkce.

V testech se autorovi nejvíce osvědčila reprezentace permutací a to i v časově limitovaném výpočtu. Nejlépe si vedla heuristikou inicializovaná populace 40 jedinců vyvíjejících se po 25 generací s dvoubodovým křížením, pravděpodobností mutace 0,05 a selekcí, kdy přežívají jen ti nejlepší jedinci. Naopak reprezentace „priority rule“ vykazovala téměř nulovou schopnost se vyvíjet. Nevýhodou jsou pouze malé velikosti problémů, na kterých byly algoritmy testovány – jen 30 a 60 úloh. Problémy stejné velikosti byly zvoleny i v článku [1].

Daleko větší problémy (až 1000 úloh) řeší autoři článku [14]. Před spuštěním algoritmu provádí předzpracování, ve kterém detekují a opravují „forbidden sety“, tj. dvě úlohy, které nemohou být zpracovány současně, avšak kombinace omezení určují jejich přesné pořadí. Oprava se realizuje přidáním hrany patřičné délky mezi těmito úlohami, aby se vynutilo správné pořadí, čímž se zmenší prohledávaný prostor.

Autoři navrhli paralelní GA vyvíjející současně populace na 5 oddělených ostrovech (na každém 10 jedinců), mezi kterými dochází v každé 10. generaci k migraci jednoho nejlepšího jedince. Místo generačního modelu používají model steady-state. V generačním modelu se pomocí selekce, křížení a mutace vytváří z aktuální populace nová, po jejímž naplnění dojde k nahrazení aktuální populace populací novou. Steady-state model považuje za generaci jedno provedení operací selekce, křížení a mutace (na každém ostrově), po nichž dochází k okamžitému zapojení vytvořeného jedince do populace, a to na úkor jiného jedince (nahrazení). Autoři ukončují výpočet po splnění jedné z podmínek: provede se 5000 vyčíslení jedinců, po 20 generací se nevygeneruje platný jedinec (proveditelný rozvrh), nebo 100 generací bez zlepšení. Nepříjemným faktem je však výpočetní čas jedné instance, který je pro množinu rovnoměrně složenou z úloh velikosti 200, 500 a 1000 v průměru více než 70 minut, přičemž výpočet přímou metodou s o 2 % horším výsledkem trvá 32 s. Kromě GA testují autoři aproximační metodu, filtered beam search, dekompoziční metody a tabu search.

Práce [10] řeší problém rozvrhování s více módy (MRCPSP), kdy každá úloha dává na výběr několik módů zpracování, přičemž každý z nich vyžaduje jiné množství zdrojů a trvá různě dlouho. Autor přichází se zajímavou myšlenkou dvoupopulačního generačního modelu GA. Tento algoritmus převádí po každé generaci populaci vlevo zarovnaných jedinců (startovní časy úloh jsou co nejblíže začátku rozvrhu) na populaci vpravo zarovnaných jedinců (startovní časy úloh jsou co nejblíže konci rozvrhu) a po další generaci zpět. Dosahuje tím lepších výsledků, než GA jednopopulační. Článek se také zaměřuje na různé optimalizační metody (operující módy úloh) a testuje různé pokutové (penalty) funkce pro neplatné jedince. Zkoumá metodu Artificial Immune System (AIS), využívající hypermutaci měnící méně kvalitní jedince daleko více než ty kvalitní, a Scatter search, starající se nejen o kvalitu jedinců v populaci, ale i o jejich diverzitu.

U algoritmů PSO se zkoumají především možnosti jednotlivých topologií a jejich přizpůsobení na diskrétní prostor. Základní algoritmus PSO operuje s plně propojenou topologií, kde existuje jediná neighborhood-best pozice nazývaná global-best. Článek [6] přichází s novým a ve výsledku lepším konceptem IPSO (Improved PSO), který místo jediné global-best g zavádí g_k jakožto k. nejlepší pozici ze všech částic. Update rychlosti částice i se pak (ve zkráceném zápise) řídí vzorcem (3.4).

$$v_i(t+1) = \varphi_0 \cdot v_i(t) + \varphi_1 \cdot r_1(t) \cdot [p_i - x_i(t)] + \frac{1}{k} \cdot \sum_{j=1}^{k} [\varphi_{2j} \cdot r_{2j}(t) \cdot [g_j - x_i(t)]] \qquad (3.4)$$

Protože řeší problém s kritériem earliness-tardiness (a relacemi následnosti), používají pro reprezentaci částic přímo vektory startovních časů.

Autoři článku [3] testují tři statické relace sousednosti: plně propojenou topologii (global-best), řídce propojenou topologii (značeno local-best) (částice i sousedí s částicemi $i - 2$, $i - 1$, $i + 1$ a $i + 2$), topologii hvězda s jedním uzlem uprostřed a jednu dynamicky se měnící topologii (hierarchické PSO). Částice jsou při hierarchickém PSO uspořádány do úplného kořenového stromu s konstantním větvícím faktorem (v jejich případě 3). Sousedem každé částice je pouze její předek v hierarchii (s výjimkou částice v kořeni). Po každé generaci se může částice posouvat v hierarchii nahoru či dolů. Pokud některý z následníků reprezentuje kvalitnější rozvrh než jeho rodič, jsou částice prohozeny. Vyhodnocování probíhá od kořene, což umožňuje posun každé částice směrem ke kořeni pouze o jednu úroveň. Nejlépe si v testech na 30, 60, 90 a 120 úlohách s 1 000, 5 000 a 50 000 vyčíslení jedinců počínala topologie lbest.

Autoři článku [4] představují UPSO (unified PSO) a ukazují, že kombinací local-best (řídké topologie – sousedy částice i jsou $i - 1$ a $i + 1$) a global-best (plně propojené topologie) metod lze dosáhnout zajímavých výsledků. Vektor rychlosti UPSO se počítá dle rovnice (3.5).

$$v_i(t+1) = u \cdot gbest_i(t) + (1-u) \cdot lbest_i(t) \qquad (3.5)$$

$$lbest_i(t+1) = \varphi_0 \cdot v_i(t) + \varphi_1 \cdot r_1(t) \cdot [p_i - x_i(t)] + \varphi_2 \cdot r_2(t) \cdot [g_i - x_i(t)] \qquad (3.6)$$

$$gbest_i(t+1) = \varphi_0 \cdot v_i(t) + \varphi_1 \cdot r_1'(t) \cdot [p_i - x_i(t)] + \varphi_2 \cdot r_2'(t) \cdot [g - x_i(t)] \qquad (3.7)$$

Součet reprezentuje konvexní kombinaci příspěvků z local-best (3.6) a global-best (3.7) vektorů řízenou parametrem u. Proměnná g_i značí nejlepšího ze sousedů částice i v topologii local-best, zatímco g značí nejlepšího jedince v celé populaci. Problémy o velikosti 50 úloh řešilo nejlépe UPSO s $u = 0{,}2$.

Práce [7] ukazuje na problému rozvrhování s kritériem tardiness a changeover časy funkci algoritmu DPSO (discrete PSO). Využívá topologie local-best, global-best i jejich kombinaci. Částice reprezentuje vektorem, kde hodnota na i-té pozici určuje index úlohy, která bude zařazena na i-té pozici. Pro tuto diskrétní reprezentaci navrhli autoři operátory odčítání, sčítání a násobení. Rozdíl (vzdálenost) částic $x_1 - x_2$ je vektor uspořádaných dvojic (posunů úloh), kde první z dvojice označuje číslo úlohy j a druhé rozdíl $(m - k)$ takový, že m označuje pozici úlohy j ve vektoru x_2 a k pozici úlohy j ve vektoru x_1, neboli $x_1(k) = x_2(m) = j$. Pro $x_1 = (1, 2, 3, 4)$ a $x_2 = (2, 3, 1, 4)$ je $x_1 - x_2 = \{(1, 2), (2, -1), (3, -1), (4, 0)\}$, kde dvojice (1, 2) znamená, že úloha číslo 1 má být posunuta doprava o 2 pozice (zpožděna). Sčítání dvou takto vzniklých vektorů se realizuje po jednotlivých dvojicích se stejným číslem úlohy i tak, že $(i, k) + (i, m) = (i, k + m)$. Obdobně pro násobení $c \cdot (i, k) = (i, rr(c \cdot k))$, kde rr je náhodné zaokrouhlení nahoru nebo dolů. Vektor posunů úloh se pak aplikuje na vektor pozice částice. Takto vzniklý vektor nemusí být platný a proto je nutné ještě provést opravnou dokončovací proceduru.

Autoři ukazují důležitost intenzifikační fáze, varianty lokálního optimalizátoru, náhodně provádějící operace prohození dvou úloh nebo zařazení dvou úloh bezprostředně za sebe. Tato reprezentace dosahovala značných vylepšení na Cicirello's a ORLIB benchmarcích obsahujících problémy o velikosti až 100 úloh. Počet částic byl stanoven na 120 a počet vyčíslení fitness funkce na neuvěřitelných 20 000 000.

Další oblast rozvrhování, na kterou se upírá velká pozornost, je vytváření robustních plánů. Zajímavé kritérium flexibility rozvrhu (schopnost absorbovat změny a promptně na ně reagovat) na problémech s temporálními omezeními řeší autoři [8]. Flexibilitu definují dvěma způsoby: počet vzájemně přímo nesvázaných dvojic nebo vzájemná volnost vazeb všech úloh do maximální délky plánu. Srovnávají dvě metody pro tvorbu robustního, částečně seřazeného plánu (POS). Metoda EBA (Envelope Based Algorithm) – založená na PCP (Precedence Constraint Posting) – vytvoří rozvrh s respektem k temporálním omezením (time layer), najde špičky porušení kapacity zdrojů (resource layer) a přidá nové omezení řešící konflikt do time layeru, který ho zpracuje a vytvoří nový POS, atd. Metoda ESTA (Earliest Start Time Algorithm) vytvoří jeden EST plán, který zrobustní pomocí chaining procedury. V testech si lépe počínal ESTA (až 4x více vyřešených úloh, řádově menší časové nároky, téměř o polovinu kratší rozvrhy, téměř poloviční počet generovaných omezení). S kritériem flexibility si na menších instancích lépe poradil ESTA, zatímco na těch větších EBA.

Další problém, kdy je nutný robustní rozvrh, je popsán v článku [9]. Problém obsahuje temporální omezení a úlohy s proměnlivou délkou zpracování určenou minimální a maximální hodnotou. Rozvrh, reprezentující řešení, musí být platný pro všechny doby zpracování. Problém reprezentují pomocí časového modelu – grafu se dvěma typy hran. Hrana $[a, b]$ označuje stav, kdy musí existovat hodnota z intervalu $[a, b]$, které může hrana nabývat. Hrana $[a: b]$ určuje, že hrana smí nabýt všech hodnot z intervalu $[a, b]$. V tomto grafu pak propagují startovní časy a časy dokončení jednotlivých úloh. Detekci MCS (Minimal Critical Set) převádí na problém minimálního toku, který řeší Edmond-Karpovým algoritmem. Ve srovnání s CS (complete search) potřebuje algoritmus řádově méně času i paměti.

Kapitola 4

Specifika vytvořeného PSO

V této kapitole se zaměřím na jednotlivé aspekty implementace. Nejprve shrnu výhody a nevýhody zvoleného prostředí Matlab. V další části se zaměřím na kód vlastního algoritmu a především na opravnou funkci, starající se o nápravu chyb ve vektoru startovních časů vzniklých pohybem částice podle schématu PSO. Zaměřím se i na analýzu možností generování počáteční populace IRS heuristikou a nakonec vysvětlím nové koncepty, které jsem implementoval.

4.1 Volba implementačního prostředí

Pro implementaci jsem zvolil Matlab, který je vhodný pro vývoj menších projektů, nebo návrh a testování prototypů. Během implementace jsem objevil tyto výhody a nevýhody zvoleného prostředí:

- Vizualizační prostředí poskytuje uživateli velké možnosti, přičemž je jednoduché na ovládání a umožňuje okamžité zobrazení vypočítaných dat.
- Základní jednotkou je matice, podporující širokou škálu již implementovaných operací. Matice různých rozměrů lze navíc ukládat do buňkových polí (cell arrays).
- Nabízí struktury, do kterých lze jednoduše přidávat další elementy.
- Výhodou je snadné debugování, při kterém je možné vypsat obsah matic a spouštět další kód.
- Funkci lze předávat jako parametr. Navíc lze určit, které parametry předané funkce budou při jejím volání nastavovány uvnitř obsluhující funkce (budou se měnit) a které budou nastaveny z vnějšku (zůstanou konstantní a při volání předané funkce budou automaticky doplněny).
- Umožňuje využití toolboxů – rozsáhlých knihoven šikovných funkcí (např. TORSCHE toolbox [19]).
- Výpočty jsou řádově pomalejší než v jiných jazycích.

- Nepodporuje objekty, pouze jednoduché struktury.
- Nelze předávat proměnné „odkazem“.

Po celkovém zhodnocení pozitiv a negativ, která přináší implementační prostředí Matlab, bych opět volil stejně a oželel výpočetní výkon za pohodlnou práci v prostředí Matlabu.

4.2 Popis algoritmu

Hledaným řešení instance problému s n úlohami (instance velikosti n) je vektor délky n, obsahující celá čísla, která odpovídají startovním časům jednotlivých úloh (prostor řešení je diskrétní). Vektor startovních časů se vyhodnotí fitness funkcí, která spočítá hodnotu kritéria a optimalizace probíhá hledáním jejího minima.

V kapitole 3.3 se ukázaly jako slibné reprezentace jedinců: permutace (hodnota na i-té pozici určuje pořadí úlohy i), priority value (activity list), a reprezentace užívaná při DPSO (hodnota na i-té pozici určuje, která úloha bude odstartovaná jako i-tá). Pro zjištění hodnoty kritéria je nutné z pořadí úloh (určující výše zmíněné reprezentace) vytvořit vektor startovních časů. Jak bylo řečeno v kapitole 2.5, je pro kritérium energetické efektivity a earliness-tardiness výpočetně velmi obtížné určit z daného pořadí úloh nejoptimálnější startovní časy. Proto jsem zvolil reprezentaci vektorem startovních časů.

Algoritmus provádí během svého výpočtu s částicí operace, které mohou (a často tak i činí) vytvořit vektor startovních časů, který porušuje temporální omezení nebo omezení daná kapacitou zdrojů. Z toho důvodu je před vyčíslením hodnoty kritéria (fitness funkce) nutné provést opravu vektoru startovních časů (funkce *fix_start_times*). Funkce *fix_start_times* nejprve opraví zjevná porušení kladných hran temporálních omezení a následně se snaží pro každou úlohu i nalézt novou hodnotu $sFixed_i$, která bude co nejmenší a přitom $sFixed_i \geq s_i$. Pokud oprava vektoru není možná, je vektor označen za neplatný a *fix_start_times* vrátí prázdný vektor. Opravná funkce je přesněji popsána v sekci 4.2.1.

Vyčíslení vlastní fitness funkce, jak ukazuje následující kód, se realizuje ve dvou krocích. Prvním krokem je report generující funkce, která spočítá vlastnosti rozvrhu (součet všech dob zapínání, dob vypínání, dob nečinného běhu, ...). Druhým krokem je určení fitness, které se provádí pouhým sečtení hodnot jednotlivých vlastností rozvrhu pronásobených patřičnými konstantami. Díky této modularitě je možné změnit konstanty bez zásahu do kódu algoritmu pouhou změnou *fceFitness*.

```
1   function f (sFixed)
2       report = fceReport (sFixed)
3       return fceFitness (report)
4   end
```

Zdrojový kód celého algoritmu PSO vypadá následovně:

```
1   populace = inicializuj_částice ()
2   for i = 1 : počet generací
3       for j = 1 : počet částic
4           s = urči_novou_pozici_částice (populace(j))
5           sFixed = fix_start_times (round (s))
6           if (sFixed != Ø & f (sFixed) ≤ f (populace (j).s))
7               aktualizuj_pozici (populace (j), sFixed)
8               if (f (sFixed) < f (populace (j).particle-best_fitness)
9                   změň_particle-best_pozice (populace (j), sFixed)
10              end
11          end
12      end
13      for j = 1 : počet částic
14          přepočítej_hodnotu_neighborhood-best (populace (j))
15      end
16  end
```

První řádek inicializuje celou populaci (počátečních pozice, vektory rychlosti, particle-best a neighborhood-best pozice). Hlavní cyklus se provádí pro předepsaný počet generací (pohybů částic). V hlavním cyklu se pro každou částici nejprve spočítá nová pozice, na kterou se spustí opravná procedura (řádky 4-5). Pokud je nová pozice po opravě platná a její fitness není horší, než aktuální fitness částice, provede se aktualizace pozice částice (řádek 7). Pokud nová fitness překonává i fitness particle-best pozice, je particle-best nahrazena novým vektorem (řádek 9). Hodnota fitness funkce každého vektoru je počítána pouze jednou a je pro urychlení výpočtu cacheována. Po jednom pohybu všech částic se přepočítají hodnoty neighborhood-best (řádek 14).

4.2.1 Opravná funkce

Pro opravu vektoru startovních časů po uskutečnění pohybu částice jsem navrhl opravnou funkci s odrozvrhovacím krokem. Podobnou funkci popisují autoři článku [14]. Vstupem algoritmu je vektor s, určující startovní časy, které mohou porušovat temporální omezení nebo kapacitu zdrojů. Výstupem algoritmu je vektor, který splňuje omezení (platný rozvrh), nebo prázdný vektor, pokud se nepodařilo startovní časy opravit.

```
1  function fix_start_times (s)
2      inicializuj_zdroje ()
3      unscheduleAvailable = n
4      negativeError (1 : n) = 0

5      idxs = pořadí_úloh_podle_rostoucích_startovních_časů (s)
6      idxs = oprav_indexy (idxs)

7      counter = 1
8      while counter ≤ n
9          uloha = idxs (counter)
10         earliestStart = max (nejdřívější startovní časy určené temporálními
                                omezeními od již zařazených úloh, s (uloha),
                                negativeError (uloha), nejdřívější možné zařazení
                                úlohy na zdroje)
11         if zařazením úlohy uloha na startovní čas earliestStart by došlo k porušení
                                záporné hrany
12             if unscheduleAvailable ≤ 0
13                 return Ø
14             else
15                 unscheduleAvailable – –
16             end

17             negUloha = vyber úlohu s nejnižším časem, ke které vede porušená
                                záporná hrana
18             posun = hodnota, o kterou musí být negUloha posunuta dále od startu,
                                aby již nebyla porušena záporná hrana, vzhledem
                                ke startovnímu času earliestStart úlohy uloha
19             negativeErr (negUloha) = negativeErr (negUloha) + posun
20             odrozvrhni všechny úlohy až po negUloha včetně
21             counter = counter – počet odrozvrhnutých úloh
22         else
23             sNew (uloha) = earliestStart
24             zařaď úlohu uloha v čase earliestStart na zdroje
25             counter + +
26         end
27     end
28     return sNew
29 end
```

Funkce nejprve inicializuje zdroje a nastaví maximální počet odrozvrhovacích kroků, který je roven počtu úloh (doporučuje [14]). Následně se určí pořadí úloh podle rostoucích startovních časů. V případě rovnosti startovních časů je pořadí těchto úloh určeno náhodně. Na takto vzniklé pořadí se zavolá funkce *oprav_indexy*, která toto pořadí upraví tak, aby vyhovovalo kladným temporálním omezením.

V hlavním cyklu se pro každou úlohu určují jednotlivé startovní časy tak, aby bylo vyhověno i záporným temporálním omezením a kapacitě zdrojů. Zařazování se uskutečňuje dle pořadí určující výstup z funkce *oprav_indexy*, které se již během výpočtu nemění. Startovní čas každé úlohy se vybere tak, aby byl minimální a splňoval tyto podmínky (řádek 10):

- vyhovuje všem kladným temporálním omezením od již zařazených úloh,
- kapacita zdrojů umožní zpracování úlohy,
- musí být větší nebo roven hodnotě požadované ve vstupním vektoru, což umožňuje, aby úloha nemusela být zařazena okamžitě, kdy to umožňují temporální omezení a kapacita zdrojů, protože optimální zařazení úlohy může být jiné,
- musí být větší nebo roven minimálnímu času, který požaduje některá záporná hrana, kvůli které nemohla být jiná (odrozvrhnutá) úloha zařazena.

Pro takto určený startovní čas se zjistí, zda nedojde k porušení některé záporné hrany vedoucí z této úlohy k některé již zařazené úloze. V případě, že nevznikne konflikt, je úloha zařazena a výpočet pokračuje následující úlohou (řádky 23-25). Pokud konflikt vznikne, zjistí se, zda nebyl vyčerpán maximální počet odrozvrhování. Pokud ne, vybere se nejdříve zařazená úloha, která ho způsobuje (řádek 17), a spočítá se pro ni nejdřívější možný (zpožděný) startovní čas (řádky 18-19). Všechny úlohy od zařazované až po zpožděnou včetně jsou odrozvrhnuty (řádek 19) a pokračuje se od začátku pokusem o zařazení zpožděné úlohy na nový startovní čas. Složitost funkce je $O(n^3)$.

Funkci opravující pořadí úloh (řádek 6), popisuje následující kód:

```
1  function idxsNew = oprav_indexy (idxs)
2      used (1 : n) = false
3      for i = 1 : n
4          pred (i) = počet kladných hran vedoucích do úlohy i
5      end

6      cnt = 1
7      i = 1
8      while i ≤ n
9          if used (idxs (i)))
10             i = i + +
11         else
12             for j = 0 : n – i
13                 if pred (idx (i + j)) == 0 & ! used (idxs (i + j))
14                     pred (k) = pred (k) – 1, kde k jsou indexy úloh, do kterých
                                   vede kladná hrana z úlohy s indexem i + j
15                     idxsNew (cnt) = idxs (i + j)
16                     used (idxs (i + j)) = true
17                     cnt + +
18                     if j == 0
19                         i + +
20                     end
21                 end
22             end
23         end
24     end
25     return idxsNew
26 end
```

Pro všechny úlohy se spočítá počet kladných hran, které do ní vedou – *pred* (řádky 3-5). Nové řazení úloh se provádí vybíráním úloh, do kterých nevedou žádné kladné hrany (řádek 13) podle pořadí určeného vstupním pořadím úloh. Po zařazení každé úlohy se provádí aktualizace vektoru *pred*. Složitost funkce je $O(n^3)$.

4.2.2 Generování inicializační populace

Počáteční populaci algoritmu PSO jsem nejprve generoval pomocí pravidel (LST, MST, ...), dále jsem zkoušel generovat populaci s relaxací vždy na jedno omezení (nezáporné hrany, záporné hrany, kapacitu zdrojů). Takto vygenerované populace však nedosahovaly příliš velkých úspěchů.

Daleko lépe si počínala populace platných jedinců generovaných pomocí IRS heuristiky [5]. IRS heuristika generuje velmi kvalitní jedince vzhledem ke kritériu C_{max}. Je známým faktem, že populační algoritmy potřebují populaci různorodých jedinců. K získání různých řešení jsem využil časovou symetrii popsanou v článku [5] a zmenšení problému. K vytvoření časově invertovaného (symetrického) problému je nutné „otočit“ všechny hrany problému a upravit jejich váhu vzhledem k časům zpracování vázaných úloh. *K*-krát zmenšený problém se vytvoří vydělením časů zpracování úloh a vah všech hran zvolenou konstantou *k* a výsledky se zaokrouhlí (časy zpracování a kladné hrany nahoru, záporné hrany dolů). Takto upravené problémy pak vyřeší IRS heuristika. Kombinací časové inverze a polovičního problému (k = 2) lze vytvořit až 4 jedince:

- řešení původního problému (IRS),
- řešení časově invertovaného problému (IRS inverzní),
- řešení polovičního problému (IRS ½),
- řešení invertovaného polovičního problému (IRS ½ inverzní).

Rozdílnost takto vzniklých řešení jsem testoval na 100 problémech o velikosti n = 30 (50_MIX_n30 a 50_XDMPX_n30) a n = 100 (50_MIX_n100 a 50_XDMPX_n100). Pro každou metodu jsem spočítal průměrnou relativní odchylku kritéria C_{max} od nejlepšího řešení vygenerovaného těmito 4 metodami. Dále jsem spočítal průměrnou rozdílnost rozvrhů oproti původnímu rozvrhu vygenerovaného IRS heuristikou, která pro každou úlohu vybere menší hodnotu z rozdílu startovních časů vůči začátku nebo konci rozvrhu a tyto hodnoty sečte:

$$rozdílnost = \sum_{i=1}^{n} \min\left[abs(sIRS_i - s_i), abs(C_{\max}IRS - sIRS_i - (C_{\max} - s_i))\right] \qquad (4.1)$$

IRS heuristika přiřazuje úlohám co nejmenší startovní časy (úlohy zarovnané vlevo), což může způsobit, že vektor startovních časů počítaný pro časově inverzní problém (po zpětné inverzi úlohy zarovnané vpravo) bude značně odlišný, přestože se oba rozvrhy budou lišit jen v zarovnání úloh. Z tohoto důvodu jsem přidal informace o počtu shodných vektorů startovních časů, zarovnaných vlevo, vzniklých řešením

modifikovaných problémů s řešením původního problému a o počtu úloh, které (po zarovnání vlevo) startují v jiný čas. Výsledky shrnuje tabulka 4.

	50_MIX_n30 a 50_XDMPX_n30, n = 30					
metoda	# vyřešených problémů	ø C_{max}	ø relativní odchylka od nejlepšího C_{max}	rozdílnost rozvrhu od IRS	ø # shodných řešení po zarovnání vlevo	ø # úloh zařazených v jiný čas (po zarovnání vlevo)
IRS	100 %	273,26	0,19 %	–	–	–
IRS inverzní	100 %	275,39	1,01 %	9,07	2	10,77
IRS ½	92 %	286,23	2,99 %	0,49	62	3
IRS ½ inverzní	92 %	288,66	4,03 %	9,45	2	11,09

	50_MIX_n30 a 50_XDMPX_n30, n = 30					
metoda	# vyřešených problémů	ø C_{max}	ø relativní odchylka od nejlepšího C_{max}	rozdílnost rozvrhu od IRS	ø # shodných řešení po zarovnání vlevo	ø # úloh zařazených v jiný čas (po zarovnání vlevo)
IRS	100 %	593,23	2,06 %	–	–	–
IRS inverzní	99 %	590,06	1,71 %	130,92	0	67,46
IRS ½	94 %	613,02	4,77 %	6,46	15	30,57
IRS ½ inverzní	92 %	609,97	5,02 %	137,35	0	67,97

Tabulka 4: Výsledky testování rozdílnosti vektorů startovních časů při modifikaci problému (časovou inverzí a zmenšením na polovinu) oproti vektoru řešení původního problému.

Tabulka 4 ukazuje, že po zmenšení problémů na polovinu přestane být až 8 % z nich řešitelných IRS heuristikou, což znamená, že použitím těchto 4 metod nelze vždy získat 4 vektory startovních časů. Rozdílnost těchto vektorů je pro časovou inverzi řádově větší, než pro poloviční problém. Časovou inverzí vznikají výrazně rozdílné rozvrhy, ve kterých pro n = 30 startuje více než 1/3 úloh v jiný čas a pro n = 100 je to více než 2/3. Naopak při zmenšení problému jsou vzniklé rozvrhy po zarovnání vlevo shodné (62 % pro n = 30, 15 % pro n = 100). Kombinací časové inverze a zmenšeného problému vznikají (po zarovnání vlevo) rozvrhy jen o málo odlišné, než při samotné časové inverzi. Pro 100 úloh dává IRS s časovou inverzí v průměru lepší řešení (pro kritérium C_{max}), než pro původní problém. Je zřejmé, že pro poloviční problém s prodlouženými kladnými hranami a časy zpracování úloh a naopak zkrácenými zápornými hranami, dává IRS řešení horší (pro kritérium C_{max}).

Další možnosti v generování počáteční populace se naskytly, když p. Šůcha rozšířil IRS heuristiku [20] o možnost náhodně modifikovat prioritu zařazování úloh. Díky tomuto zásahu je možné generovat více jedinců do inicializační populace, u nichž lze řídit jejich kvalitu parametrem určujícím míru náhody. Čím vyšší je hodnota parametru, tím rozdílnější a méně kvalitní (hodnota C_{max} roste) jedinci vznikají a také klesá schopnost algoritmu generovat řešení. IRS heuristiku jsem testoval na 10 instancích problému typu XDMPX se 100 úlohami. Pro každé nastavení parametru (10, 20, 50, 100, 500, 1000 a 100000) jsem zaznamenával počet neúspěšných pokusů, dokud nebylo vygenerováno 10 platných rozvrhů, dále hodnotu C_{max}, a Manhattanskou vzdálenost (součet absolutních hodnot rozdílů startovních časů odpovídajících úloh) od nejbližšího vektoru (tabulka 5). Pro každé nastavení je generováno 10 platných

rozvrhů, což odpovídá 70 řešením pro každý problém, přičemž Manhattanská vzdálenost je počítána vždy ke zbývajícím 69 řešením problému.

	Počet neúspěšných pokusů									
Parametr	10 Instancí XDMPX, n = 100									
10	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
100	0	0	3	2	0	0	0	0	0	0
500	7	21	14	326	3	34	7	1	1	1
1000	21	31	51	331	4	48	18	1	1	1
100000	10	26	60	656	4	104	8	5	5	5

	ø hodnota C_{max}									
Parametr	10 Instancí XDMPX, n = 100									
10	284	267	241	237	335	220	221	269	228	260
20	284	266	245	247	335	231	224	269	228	260
50	289	273	260	273	335	234	229	269	229	260
100	291	282	268	293	336	255	239	271	229	260
500	302	305	279	306	342	273	258	279	236	277
1000	300	301	271	328	341	290	259	277	236	275
100000	310	317	276	329	345	289	256	280	234	284

	ø Manhattanská vzdálenost k nejbližšímu rozvrhu									
Parametr	10 Instancí XDMPX, n = 100									
10	24	124	130	878	2	481	87	1	14	78
20	60	141	184	1174	14	700	192	7	37	114
50	217	251	418	1296	63	891	370	38	112	152
100	292	433	655	1558	82	1104	463	90	126	261
500	325	594	663	1395	166	1163	472	192	293	438
1000	307	613	644	1482	153	1307	464	170	249	476
100000	430	907	692	1694	198	1257	526	227	239	547

Tabulka 5: Výsledky testování vlivu hodnoty parametru určující míru náhody IRS heuristiky na schopnost generovat řešení, hodnotu C_{max} a rozdílnost řešení. Každý sloupec je 1 instance, přičemž ve 2. a 3. části tabulky je každá hodnota průměr počítaný z 10 platných rozvrhů.

Přestože byly testované instance generované se stejným nastavením generátoru, vliv hodnoty parametru na jejich řešení IRS heuristikou se značně liší. Pro některé problémy funguje generování rozvrhu úspěšně jen do hodnoty 100, pro jiné není problém generovat řešení při hodnotě 1000. Do hodnoty parametru 50 byly všechny pokusy na testovaných instancích úspěšné. Rychlost růstu hodnoty kritéria C_{max} je také závislá na konkrétní instanci, avšak pro hodnoty parametru menší nebo rovny 50 je nárůst 0 – 15 %. Vzdálenost jednotlivých řešení může být pro některé instance a hodnotu parametru 10 téměř žádná, a pro jiné značná. Pro zvyšující se hodnoty parametru je růst vzdálenosti pozvolný.

Z předešlé analýzy vyplývají 2 způsoby generování počáteční populace. První možností, kterou používám ve většině testů, je vygenerovat 1-4 jedince IRS heuristikou za pomoci časově invertovaného a zmenšeného problému. Vzniklé jedince pak rozkopíruji na celou populaci. Výhodou této varianty je vysoká kvalita jedinců,

nevýhodou je jejich malá diverzita. Druhá varianta je vygenerovat větší množství různě kvalitních jedinců pomocí vhodné volby parametru IRS heuristiky. Do základu jsem použil 1-4 jedince z první varianty a další jsem vytvářel na původním a invertovaném problému s rovnoměrně používanými hodnotami parametrů z tabulky 6, dokud neměla populace celkem 30 jedinců.

Hodnoty parametru													
10	15	20	30	40	50	60	80	90	100	150	200	300	500

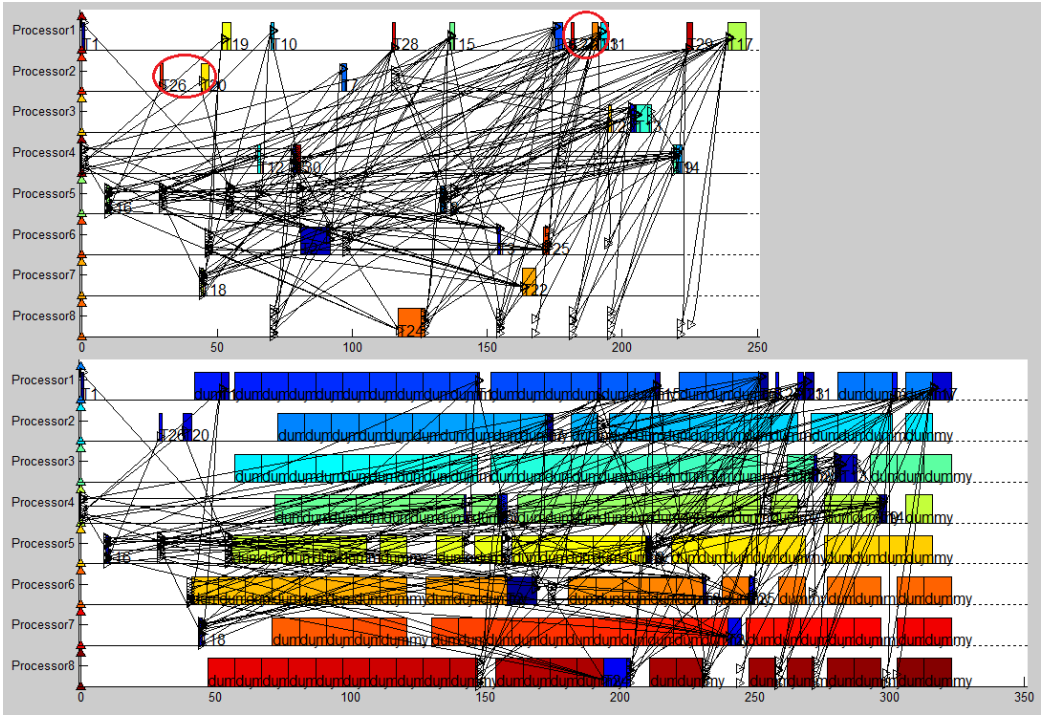
Tabulka 6: Hodnoty parametru IRS heuristiky určující míru náhody, které byly rovnoměrně používány ke generování různorodé populace.

Výhodou této varianty je různorodost generovaných jedinců. Nevýhodami jsou menší kvalita populace jako celku (nižší průměrná C_{max}) a větší časová náročnost, způsobená generováním více jedinců a také snižující se rychlostí generování rozvrhu s rostoucí hodnotou parametru. V části testů obě varianty porovnám.

4.3 Implementované varianty

4.3.1 Bloková IRS

Jedním z nápadů bylo řešit kritérium energetické efektivity 1 (EE1 zmiňované v sekci 2.4) pomocí samotné IRS heuristiky. Bezprostředně po sobě zařazené úlohy na stejnou jednotku zdroje jsou tlačeny kritériem k tomu, aby prodleva mezi nimi buď nebyla žádná (nebo co nejmenší) nebo naopak, aby byla dlouhá minimálně 10 časových jednotek. Nápad spočíval v tom, že by se do problému přidaly další nijak neomezené dummy úlohy, které by při snaze minimalizovat kritérium C_{max} vkládala IRS heuristika do „mezer“ mezi úlohami. Předpoklady se ukázaly jako mylné a jak je vidět na obr. 37, IRS heuristika značně prodloužila C_{max} a dokonce pro některé problémy vytvořila rozvrh s horší hodnotou kritéria EE1. Na obr. 37 jsou červeně označena místa, kde došlo k prodloužení nečinných intervalů s délkou menších než 10 a tím ke zhoršení kritéria EE1. Pro jiná kritéria se tento nápad příliš nehodí.



Obr. 37: Ukázka problému typu XDXXX s n = 30 úlohami, 150 pouze kladnými hranami, a 8 unárními zdroji, kde bloková IRS (rozvrh dole) dosáhla horší hodnoty kritéria EE1, než pro původní problém (rozvrh nahoře). V původním rozvrhu jsou červeně označena místa, kde ke zhoršení hodnoty kritéria došlo. Navíc kvůli absenci záporných hran by měla jít hodnota kritéria snadno zmenšit.

4.3.2 Modifikace základního PSO

U PSO jsem vycházel ze základní varianty pohybu částic popsané v kapitole 3. Implementoval jsem různé varianty vztahu sousednosti (neighborhood-best): global-best (všechny částice jsou sousedé), local-best (částice i sousedí s částicemi $i - 2$, $i - 1$, $i + 1$ a $i + 2$) a jejich kombinaci nazývanou unifikované PSO (popsané v sekci 3.3). Váhu unifikačního parametru u jsem zvolil 0,2 (doporučovanou [4]) a 0,5.

Vyzkoušel jsem i variantu, kdy mají částice namísto náhodně generované počáteční rychlosti rychlost nulovou, jak je diskutováno v kapitole 3.2.

Dalším testovaným nápadem bylo měnit vektor startovních časů po částech. Na začátku algoritmu jsem určil 20 % startovních časů, na které budou aplikovány operace PSO (budou se vyvíjet), zbylých 80 % jsem zafixoval. Při každém pohybu je 4% šance, že se některá ze zafixovaných hodnot začne vyvíjet, nebo naopak, některá z vyvíjejících se hodnot se zafixuje. Tím se zmenší počet změn ve vektoru startovních časů během jednoho kroku.

Varianta, kterou v testech označuji jako standardní PSO, používá topologii global-best s náhodnou inicializací vektoru rychlosti. Dále jsou k popisu PSO připojeny tři hodnoty parametrů φ, velikost populace označená písmenem p a počet pohybů

označených g (konkrétní hodnoty jsou p44 g91, p63 g63 nebo p91 g44, kde p44 značí velikost populace 44 částic, g91 označuje 91 pohybů, obdobně další). Jiné varianty nastavení jsou označeny přidáním některých z následujících popisků (není-li označeno jinak, jedná se o nastavení jako u standardního PSO):

- u=0 – local-best,
- u=0,25 nebo u=0,5 – unifikované PSO,
- v=0 – nulová počáteční rychlost,
- onOff – změny vektoru startovních časů po částech.

4.3.3 GPSO

Zamyslel jsem se, jak si lépe poradit s tím, že PSO je navrženo pro spojitý prostor, zatímco řešení rozvrhovacích problémů je diskrétní. Výsledkem bylo GPSO, které mění způsob pohybu částice, přičemž zachovává ostatní principy PSO. Nový vektor pohybu částice se určí obdobně, jako se provádí dvoubodové křížení tří jedinců v GA: Určování nového vektoru se účastní vektory současné pozice, particle-best a neighborhood-best (v tomto případě global-best). Náhodně se vyberou body křížení k, m ($0 \leq k \leq m$). Náhodně se zvolí jeden ze tří vektorů a vybere se k úloh s nejnižšími startovními časy. Tyto startovní se zapíší do nového vektoru. Následně se zvolí druhý vektor, ze kterého se vybere $m - k$ úloh s nejmenšími startovními časy, které nebyly vybrány v předchozím kroku. Startovní časy vybraných úloh se přidají do nového vektoru. Startovní časy zbylých úloh se určí podle posledního vektoru.

Další vylepšení této metody spočívá v tom, že vektor aktuální pozice bude mít větší váhu než ostatní vektory, tzn. je z něj určován průměrně dvojnásobný počet startovních časů než z jednotlivých vektorů particle-best a neighborhood-best. Poměr vybíraných startovních časů z aktuálního vektoru, particle-best a neighborhood-best je 2:1:1.

Jinou možností určení nové pozice částic je uniformní křížení. V tomto případě jsou startovní časy úloh náhodně vybrány z jednotlivých vektorů (aktuální, particle-best, neighborhood-best). Nově vzniklý vektor propojuje jednotlivé vektory daleko složitěji, protože je kombinuje ve větším počtu bodů.

Nový jedinec vzniklý jednou z výše popsaných „genetických“ operací nepřináší žádné nové startovní časy – neprovádí exploraci prostoru. To je stejně jako u GA řešeno mutací. Navrhl jsem dvě různé mutace. První z nich posouvá startovní čas pouze vybrané úlohy, zatímco druhá mutace posouvá i startovní čas všech úloh s vyšším startovním časem. Velikost posunu je vybrána náhodně z povoleného intervalu a zaokrouhlena na celé číslo. Testoval jsem tyto intervaly mutace: [-5; 5], [$-p_{ø}$ / 2; $p_{ø}$ / 2], [$-p_{ø}$; $p_{ø}$], kde $p_{ø}$ označuje průměrný čas zpracování úlohy v problému. Dalším vylepšením byla nevyvážená mutace, upřednostňující posun úlohy blíže k začátku oproti posunu ke konci rozvrhu v poměru 2:1.

Standardní varianta GPSO používá dvoubodové křížení se stejným poměrem jednotlivých vektorů bez mutace. Další varianty nastavení jsou označeny přidáním

některých z následujících popisků (není-li označeno jinak, jedná se o nastavení jako u standardního GPSO):

- 2bod 2:1:1 – nevyvážené 2bodové křížení v poměru 2:1:1,
- unif. – uniformní křížení,
- mut.5 – mutace [-5; 5],
- mut.p/2 – mutace [$-p_{\emptyset}$ / 2; $p_{\emptyset}$ / 2],
- mut.p – mutace [$-p_{\emptyset}$; $p_{\emptyset}$],
- nevyv. – nevyvážená mutace upřednostňující posun úlohy blíže k začátku rozvrhu v poměru 2:1,
- vš.násl. – mutace nejen vybrané úlohy, ale i všech úloh s vyšším startovním časem.

4.3.4 GA

Pro porovnání výkonnosti PSO jsem vytvořil genetický algoritmus. Testoval jsem generační a steady-state schéma (vysvětleno v části 3.3). Pro výběr (selekci) jedinců jsem využil 2-turnaj a ruletovový výběr. Selekce 2-turnaj vybere náhodně dva jedince z populace, kteří se utkají v turnaji, což znamená, že je vybrán ten lepší z nich. Ruletová selekce vybírá jedince pomocí rulety s různě velkými políčky pro každého jedince. Velikost políčka odpovídá fitness jedince – čím je jedinec kvalitnější, tím větší políčko mu přísluší.

Pro křížení dvou jedinců jsem testoval jednobodové, dvoubodové a uniformní křížení, která se realizují obdobně, jak je popsáno v kapitole 4.3.3 s tím, že pro dvoubodové křížení je místo třetího jedince vybrán opět první jedinec. Křížení dvou jediců vznikají dva noví jedinci – druhý jedinec se skládá z nevybraných startovních časů do prvního jedince. Všechny varianty mutace, které byly navrženy pro GPSO, byly testovány i na GA. Nový jedinec ve steady-state modelu nahrazuje v populaci vždy toho nejhoršího.

Standardní varianta GA používá generační model, selekci 2-turnaj, jednobodové křížení s pravděpodobností 0,75 a mutaci vybrané úlohy v rozsahu [-5; 5] s pravděpodobností 0,2. Další varianty nastavení jsou označeny přidáním některých z následujících popisků (není-li označeno jinak, jedná se o nastavení jako u standardního GA):

- st-st – steady-state model,
- ruleta – ruletový výběr,
- 2bod. – dvoubodové křížení,
- unif. – uniformní křížení,
- mut.p/2 – mutace [$-p_{\emptyset}$ / 2; $p_{\emptyset}$ / 2],
- mut.p – mutace [$-p_{\emptyset}$; $p_{\emptyset}$],

- nevyv. – nevyvážená mutace upřednostňující posun úlohy blíže k začátku rozvrhu v poměru 2:1,
- vš.násl. – mutace nejen vybrané úlohy, ale i všech úloh s vyšším startovním časem,
- pk=0,5 nebo pk=1 – pravděpodobnost křížení,
- pm=0,1 nebo pm=0,3 – pravděpodobnost mutace.

Kapitola 5

Testování

V této kapitole nejprve popíši parametry testovacích množin instancí. Na vygenerovaných množinách problémů provedu ladění algoritmů PSO a GA, aby dosahovaly co možná nejlepších výsledků. Porovnám vliv různorodosti počáteční populace na zlepšení vybraných algoritmů. Dále se budu zabývat rychlostí poklesu kritéria během vývoje a budu analyzovat, které operace jsou v algoritmu časově nejnáročnější. Provedu test výkonnosti algoritmů na kritériu C_{max}. Nejdůležitější část této kapitoly spočívá v testování vybraných algoritmů na kritériích energetické efektivity 2 a earliness-tardiness na problémech až do velikosti 200 úloh. Kapitolu uzavřu testy na problému, který pochází z reálného prostředí výroby laků.

5.1 Generování testovacích problémů

Testovací množiny problémů jsem generoval dodaným generátorem [21], který umožňuje nastavení následujících parametrů (v kulaté závorce je uvedeno označení a ve složené závorce vybrané hodnoty parametru):

- počet úloh (n) {30, 100, 200}
- maximální čas zpracování (p_{max}) {20, 35, 50}
- počet kladných hran (pos_{num}) $\{1{,}5 \cdot n, 2 \cdot n\}$
- maximální délka kladné hrany (pos_{max}) {20, 50}
- počet záporných hran (neg_{num}) $\{0{,}2 \cdot pos_{num}\}$
- maximální délka záporné hrany (neg_{max}) $\{3 \cdot pos_{max}\}$
- maximální počet dedikovaných procesorů ($proc_{num}$) {8}
- maximální kapacita procesoru ($proc_{cap}$) {1, 3}
- maximální počet zdrojů požadovaných jednou úlohou ($multiproc$) {1, 4}

Pro každou specifikaci problému (XDXXX, XDXPX, XDXMP, XDMPX) jsem generoval instance o velikosti $n = 30$, $n = 100$ a $n = 200$ úloh s kombinacemi parametrů podle tabulky 7. Pokud specifikace problémů určuje, že mají obsahovat paralelní procesory (P), je nastaveno $proc_{cap} = 3$. Mají-li problémy obsahovat multiprocesorové úlohy (M), je nastavení $multiproc = 4$.

kombinace	p_{max}	pos_{num}	neg_{num}	pos_{max}	neg_{max}	$proc_{num}$
1.	20	$1{,}5 \cdot n$	$0{,}2 \cdot pos_{num}$	20	$3 \cdot pos_{max}$	8
2.				50		
3.		$2 \cdot n$		20		
4.				50		
5.	35	$1{,}5 \cdot n$		20		
6.				50		
7.		$2 \cdot n$		20		
8.				50		
9.	50	$1{,}5 \cdot n$		20		
10.				50		
11.		$2 \cdot n$		20		
12.				50		

Tabulka 7: Kombinace nastavení parametrů pro generování instancí

Generátorem byly vytvořeny různé množiny testovacích úloh se stejným počtem instancí pro každé nastavení z tabulky 7, pokud to počet generovaných instancí umožňuje. Pro přehlednost byly označeny ve formátu A_B_nC, kde A určuje počet problémů v testovacím setu, B určuje charakteristiku problémů (XDXXX, XDXPX, XDXMP, XDMPX, MIX) a C počet úloh. Charakteristika MIX znamená, že v množině jsou rovnoměrně zastoupeny problémy všech 4 charakteristik, z nichž pro každou bylo dodrženo rovnoměrné zastoupení kombinací z tabulky 7. Např. charakteristika testovací množiny 50_XDMPX_n100 určuje, že se skládá z 50 problémů s paralelními zdroji a 100 multiprocesorovými úlohami.

5.2 Hledání vhodné konfigurace PSO

V sekci 4.3 bylo popsáno velké množství různých konfigurací PSO. Cílem této části je najít nastavení parametrů, při kterých bude algoritmus dosahovat nejlepších výsledků. Pro testování jsem použil dvě množiny problémů: 50_MIX_n30 a 50_MIX_n100 (50 problémů o 30 a 100 úlohách) a kritérium energetické efektivity 2 (EE2). Do počáteční populace jsou namnoženi 1-4 jedinci, vygenerovaní IRS heuristikou za pomoci časově invertovaného a zmenšeného problému. Populace je velmi kvalitní, ale relativně málo různorodá. Vliv různé počáteční populace bude diskutován v sekci 5.4.

K řešení každého problému bylo poskytnuto algoritmu 4004 vyčíslení fitness funkce, což je průměr často se vyskytujících hodnot a zároveň počet, který zvládne Matlab spočítat v rozumném čase. Při posuzování vhodnosti vybraných konfigurací byla větší váha přikládána výsledkům na problémech se 100 úlohami, než s 30 úlohami. Rozhodujícím parametrem bylo průměrné relativní zlepšení oproti nejlepšímu rozvrhu,

nalezeném IRS heuristikou v počáteční populaci. Pro úplnost ještě uvádím, jak velkou část instancí se algoritmu nepodařilo vylepšit.

Prvními testovanými parametry byl počet částic a počet optimalizačních kroků. Jejich součin byl omezen 4004. Z tabulky 8 je patrné, že pro každý algoritmus je vhodná jiná velikost populace. Populace 91 částic byla vybrána pro:

- PSO 0,73 0,30 0,30,
- PSO 0,90 0,30 0,30,
- GPSO.

Pro PSO 0,73 1,50 1,50 byla vybrána velikost populace 44 a pro PSO 0,25 1,50 1,50 velikost 63 jedinců. Vzhledem k rozdílným velikostem populace, pro které GPSO unif. s mutací a bez ní poskytovalo nejlepší výsledky, jsem vybral obě velikosti – 91 i 44. Nyní se zaměřím na testy PSO, testy GPSO provedu později.

	50_MIX_n30, EE2						50_MIX_n100, EE2					
metoda	pop size = 91, kroků = 44		pop size = 63, kroků = 63		pop size = 44, kroků = 91		pop size = 91, kroků = 44		pop size = 63, kroků = 63		pop size = 44, kroků = 91	
	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
PSO 0,73 1,50 1,50	6,88	4	6,55	4	7,11	4	2,29	28	2,40	32	2,40	36
PSO 0,73 0,30 0,30	5,48	8	5,28	6	4,80	16	2,43	28	2,41	32	2,16	32
PSO 0,25 1,50 1,50	8,26	2	8,06	2	7,66	0	3,70	6	4,11	14	3,51	8
PSO 0,90 0,30 0,30	5,57	10	5,41	16	5,33	14	2,45	34	2,05	34	1,69	42
GPSO unif.	6,94	0	6,84	0	6,87	0	6,37	10	6,21	6	6,33	12
GPSO unif. + mut.5	8,76	0	9,26	0	9,06	0	5,88	0	5,74	2	6,30	0
GPSO	5,62	2	5,34	2	5,21	6	7,63	0	7,31	0	6,95	0

Tabulka 8: Výsledky testů PSO pro různý počet částic a počet optimalizačních kroků. Zeleně jsou označeny velikosti populace, se kterými bylo dosaženo nejlepších výsledků.

Pro všechny čtyři vybrané varianty PSO s patřičnou velikostí populace jsem vyzkoušel vliv nulové počáteční rychlosti. Výsledky v tabulce 9 ukazují, že pro 30 úloh došlo pro nejlepší PSO ke zlepšení, zatímco pro 100 úloh ke značnému zhoršení. Zatím je nejlepší nastavení PSO 0,25 1,50 1,50 p63 g63.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
PSO 0,73 1,50 1,50 p44 g91	7,11	4	2,40	36
PSO 0,73 1,50 1,50 p44 g91 v=0	5,98	4	2,61	22
PSO 0,73 0,30 0,30 p91 g44	5,48	8	2,43	28
PSO 0,73 0,30 0,30 p91 g44 v=0	5,14	4	2,50	16
PSO 0,25 1,50 1,50 p63 g63	8,06	2	4,11	14
PSO 0,25 1,50 1,50 p63 g63 v=0	8,31	0	3,09	26
PSO 0,90 0,30 0,30 p91 g44	5,57	10	2,45	34
PSO 0,90 0,30 0,30 p91 g44 v=0	5,61	4	2,43	20

Tabulka 9: Výsledky testů PSO pro náhodnou a nulovou počáteční rychlost.

Tabulka 10 srovnává globální, lokální a unifikované varianty PSO. Nejhůře dopadlo lokální PSO. Unifikované PSO dosahovalo na problémech se 100 úlohami pro obě hodnoty unifikačního parametru (u = 0,2 a u = 0,5) lepších výsledků než globální verze. Přestože pro 30 úloh byla nejvhodnější hodnota u = 0,2, je rozhodující chování na větších problémech, kde si nejlépe počínalo unifikované PSO s u = 0,5.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
PSO 0,25 1,50 1,50 p63 g63	8,06	2	4,11	14
PSO 0,25 1,50 1,50 p63 g63 u=0,5	7,67	0	4,84	2
PSO 0,25 1,50 1,50 p63 g63 u=0,2	8,39	0	4,29	8
PSO 0,25 1,50 1,50 p63 g63 u=0	7,86	2	3,58	10

Tabulka 10: Výsledky testů globální, lokální a unifikované varianty PSO.

V dalším testu jsem zkoušel vyvíjet vektor po částech. Pro problémy s 30 úlohami došlo ke zlepšení. Dokonce se zlepšily výsledky i pro 100 úloh a hodnotu u = 0,2. Nejlepším výsledkem pro 100 úloh však zůstalo původní nastavení s u = 0,5 bez parametru onOff (tabulka 11).

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
PSO 0,25 1,50 1,50 p63 g63 u=0,5	7,67	0	4,84	2
PSO 0,25 1,50 1,50 p63 g63 u=0,5 onOff	8,95	0	4,40	6
PSO 0,25 1,50 1,50 p63 g63 u=0,2	8,39	0	4,29	8
PSO 0,25 1,50 1,50 p63 g63 u=0,2 onOff	8,82	2	4,73	8

Tabulka 11: Výsledky testů vyvíjení vektoru po částech.

Zajímalo mě, zda problémy neobsahují příliš velké silné komponenty, které by bylo nutné vyvíjet současně. Zafixování části silné komponenty by mohlo znemožňovat vývoj zbylé části komponenty. Před návrhem algoritmu, který by zajišťoval vývoj celých komponent současně, jsem spočítal průměrný počet komponent v problémech s 30, 100 a 200 úlohami, tabulka 12. Výsledky ukazují, že počet silných komponent je jen o málo menší než počet úloh, což znamená, že velkých silných komponent je zanedbatelný počet a tento nový algoritmus by nepřinášel téměř žádné zlepšení.

	50_MIX_n30	50_MIX_n100	50_MIX_n200
Průměrný počet silných komponent	25,66	91,98	185,78

Tabulka 12: Výsledky testů počtu silných komponent.

Nejlepší variantou klasického PSO je PSO 0,25 1,50 1,50 p63 g63 u0,5, které se tak kvalifikuje do závěrečných testů. Tímto uzavírám sekci PSO a vracím se zpět ke GPSO.

Pro GPSO jsem nejprve vyzkoušel jaký bude mít vliv nevyvážené dvoubodové křížení upřednostňující aktuální pozici oproti particle-best a neighborhood-best v poměru 2:1:1 ve srovnání s vyváženým poměrem jednotlivých vektorů. Výsledky v tabulce 13 ukazují, že nevyvážené křížení je na obou testovacích množinách lepší než vyvážené.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GPSO p91 g44	5,62	2	7,63	0
GPSO p91 g44 2bod 2:1:1	5,69	4	7,89	0

Tabulka 13: Výsledky testů dvoubodového křížení upřednostňující aktuální pozici.

Pro obě testované velikosti populace uniformního GPSO a variantu GPSO 2bod. 2:1:1 jsem vyzkoušel všechny tři intervaly mutace: [-5; 5], [-$p_ø$ / 2; $p_ø$ / 2], [-$p_ø$; $p_ø$] (tabulka 14). Na problémech o 100 úlohách si nejlépe počínalo GPSO 2bod. 2:1:1 s mutací v intervalu [-$p_ø$ / 2; $p_ø$ / 2]. Pro velikost n = 30 se výsledky lišily, avšak rozhodující je velikost problémů n = 100.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GPSO p91 g44 unif.	6,94	0	6,37	10
GPSO p91 g44 unif. mut.5	8,76	0	5,88	0
GPSO p91 g44 unif. mut.p/2	8,92	0	6,65	2
GPSO p91 g44 unif. mut.p	9,33	0	5,81	0
GPSO p44 g91 unif.	6,87	0	6,33	12
GPSO p44 g91 unif. mut.5	9,06	0	6,30	0
GPSO p44 g91 unif. mut.p/2	8,97	0	6,76	2
GPSO p44 g91 unif. mut.p	9,37	0	6,05	4
GPSO p91 g44 2bod 2:1:1	5,69	4	7,89	0
GPSO p91 g44 2bod 2:1:1 mut.10	8,08	0	8,51	0
GPSO p91 g44 2bod 2:1:1 mut.p/2	8,40	0	8,76	0
GPSO p91 g44 2bod 2:1:1 mut.p	9,11	0	7,93	0

Tabulka 14: Výsledky testů různých intervalů mutace.

Pro nejlepší nastavení z předchozího testu jsem vyzkoušel vliv nevyvážené mutace, upřednostňující přesun úloh k začátku rozvrhu a vliv mutace, která kromě vybrané úlohy posouvá i všechny úlohy s vyšším startovním časem. Vlastnost mutace, zda se přesouvá jen vybraná úloha nebo i všechny úlohy s vyšším startovním časem, může mít velký vliv na kritérium earliness-tardiness. Proto jsem provedl testy i pro toto kritérium. Pro obě kritéria a obě velikosti testovacích problémů si nejlépe počínala varianta GPSO s nevyváženou mutací a posunem všech úloh s vyšším startovním časem včetně vybrané (tabulka 15).

	50_MIX_n30, EE2		50_MIX_n100, EE2		50_MIX_n30, ET		50_MIX_n100, ET	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GPSO p91 g44 2bod 2:1:1 mut.p/2	8,40	0	8,76	0	2,08	0	0,76	2
GPSO p91 g44 2bod 2:1:1 mut.p/2 vš.násl.	9,03	0	9,67	0	3,16	0	3,28	0
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv.	8,35	0	9,21	0	2,50	0	1,11	2
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl.	9,66	0	12,11	2	3,67	0	3,73	0

Tabulka 15: Výsledky testů vlivu nevyvážené mutace a mutace odsouvající nejen vybranou úlohu ale i všechny s vyšším startovním časem.

Posledním testovaným parametrem byla pravděpodobnost mutace. Tabulka 16 ukazuje, že z hodnot 0,1; 0,2 a 0,3 byla nejvhodnější 0,1. Nejlepším nastavením GPSO, které se kvalifikuje do závěrečných testů, je GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl. pm=0,1.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl pm=0,1	8,94	0	12,36	0
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl.	9,66	0	12,11	2
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl. pm=0,3	9,89	0	11,62	2

Tabulka 16: Výsledky testů různých pravděpodobností mutace.

5.3 Hledání vhodné konfigurace GA

Stejné testy jako v předchozí sekci jsem provedl i pro GA. V testu velikosti populace a počtu generací jasně vítězí GA s populací 63 jedinců vyvíjejících se po 63 generací (tabulka 17).

	50_MIX_n30, EE2						50_MIX_n100, EE2					
metoda	pop size = 91, kroků = 44		pop size = 63, kroků = 63		pop size = 44, kroků = 91		pop size = 91, kroků = 44		pop size = 63, kroků = 63		pop size = 44, kroků = 91	
	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA	7,95	2	8,11	4	7,89	0	6,52	0	6,89	2	6,77	4

Tabulka 17: Výsledky testů velikosti populace a počtu generací.

V porovnání generačního a steady-state modelu si na problémech s 30 úlohami počínal lépe generační model, zatímco pro 100 úloh, což je rozhodující, vítězil steady-state model (tabulka 18).

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA p63 g63	8,11	4	6,89	2
GA p63 st-st	7,31	0	8,23	0

Tabulka 18: Výsledky testů generačního a steady-state modelu GA.

Tabulka 19 ukazuje, že selekce jedinců ruletou je nevhodná jak pro generační, tak i steady-state model GA.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA p63 g63	8,11	4	6,89	2
GA p63 g63 ruleta	6,74	0	7,10	2
GA p63 st-st	7,31	0	8,23	0
GA p63 st-st ruleta	7,39	4	6,16	4

Tabulka 19: Výsledky testů ruletové selekce.

Pro oba modely GA jsem testoval vliv způsobu křížení. Testoval jsem jednobodové, dvoubodové a uniformní. Tabulka 20 ukazuje, že pro obě sady testovacích problémů je nejvhodnější jednobodové křížení.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA p63 g63	8,11	4	6,89	2
GA p63 g63 2bod.	7,92	2	6,66	2
GA p63 g63 unif.	7,14	4	4,44	6
GA p63 st-st	7,31	0	8,23	0
GA p63 st-st 2bod.	7,53	0	8,11	2
GA p63 st-st unif.	7,35	0	6,03	2

Tabulka 20: Výsledky testů jednobodového, dvoubodového a uniformního křížení.

Pro další testy jsem vybral jen steady-state model GA, který dosahoval nejlepších výsledků na množině problémů se 100 úlohami. Vyzkoušel jsem tři intervaly mutace: [-5; 5], [$-p_ø$ / 2; $p_ø$ / 2], [$-p_ø$; $p_ø$] (tabulka 21). Nejoptimálnější byl pro obě testovací množiny interval [$-p_ø$; $p_ø$].

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA p63 st-st	7,31	0	8,23	0
GA p63 st-st mut.p/2	7,04	0	8,03	0
GA p63 st-st mut.p	8,77	0	8,48	0

Tabulka 21: Výsledky testů velikosti intervalu mutace.

Testoval jsem vliv nevyvážené mutace, upřednostňující přesun úloh k začátku rozvrhu a vliv mutace, která kromě mutované úlohy posouvá i všechny úlohy s vyšším startovním časem. Stejně jako u GPSO, i zde jsem provedl test i pro kritérium earliness-tardiness. Výsledky v tabulce 22 ukazují, že pro obě kritéria a obě velikosti testovacích problému je nejlepší volbou GA s nevyváženou mutací a posunem všech úloh s vyšším startovním časem včetně mutované.

	50_MIX_n30, EE2		50_MIX_n100, EE2		50_MIX_n30, ET		50_MIX_n100, ET	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA p63 st-st mut.p	8,77	0	8,48	0	2,76	0	0,19	12
GA p63 st-st mut.p vš.násl	9,27	0	10,29	0	2,74	0	2,27	0
GA p63 st-st mut.p nevyv.	7,97	0	8,74	0	3,09	0	0,47	2
GA p63 st-st mut.p nevyv. vš.násl.	9,27	0	12,12	2	3,55	0	2,94	0

Tabulka 22: Výsledky testů vlivu nevyvážené mutace a mutace odsouvající nejen mutovanou úlohu ale i všechny úlohy s vyšším startovním časem.

Test pravděpodobnosti mutace (tabulka 23) prokázal, že z hodnot pravděpodobnosti 0,1; 0,2 a 0,3 je pro problémy se 100 úlohami nejvhodnější hodnota 0,1.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1	9,13	0	12,87	0
GA p63 st-st mut.p nevyv. vš.násl.	9,27	0	12,12	2
GA p63 st-st mut.p nevyv. vš.násl. pm=0,3	9,87	0	11,56	4

Tabulka 23: Výsledky testů pravděpodobnosti mutace.

V testu pravděpodobnosti křížení (tabulka 24) se pro 100 úloh ukázala nejlepší původní hodnota 0,75, což znamená, že na čtvrtinu vybraných dvojic jedinců se před zpětným zařazením do populace aplikuje pouze mutace. Další testované hodnoty byly 0,5 a 1. Posledním algoritmem, který se kvalifikuje pro závěrečné testy je GA p63 st-st mut.p nevyv. vš.násl. pm=0,1.

metoda	50_MIX_n30, EE2		50_MIX_n100, EE2	
	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1 pk=0,5	8,63	0	11,44	0
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1	9,13	0	12,87	0
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1 pk=1	9,31	0	12,77	0

Tabulka 24: Výsledky testů pravděpodobnosti křížení.

5.4 Vliv inicializace populace

V této kapitole se budu zabývat vlivem různorodosti a kvality počáteční populace na výkon algoritmu. První testovaná populace (používaná při ladění algoritmů) byla naplněna 1-4 různými jedinci vygenerovanými IRS heuristikou za pomoci časově invertovaného a zmenšeného problému. Do druhé populace bylo IRS heuristikou vygenerováno 30 různě kvalitních jedinců pomocí časové inverze a hodnot parametru, určující míru náhodného zařazování úloh (více v sekci 4.2.2). Takto vzniklá populace obsahuje značně dlišné jedince, avšak její průměrná hodnota C_{max} je vyšší (zatímco hodnota kritéria EE2 nižší). Tato populace samozřejmě obsahuje i jedince, kterými byla plněna první populace.

Tabulka 25 obsahuje výsledky pro první populaci a tabulka 26 pro druhou, obě s kritériem EE2. Důležité je zmínit, že samotné generování 30 jedinců do druhé populace oproti 1-4 do populace první přineslo zlepšení kritéria EE2. Pro n = 30 to bylo 1,56 %, zatímco pro n = 100 přineslo 1,97 %. Tyto hodnoty nejsou v tabulce 26 zohledněny, protože hodnota zlepšení se určuje vůči počáteční populaci. Výsledky ukazují, že po přičtení těchto hodnot je absolutní kvalita získaných rozvrhů o něco lepší. Pro srovnání výkonnosti algoritmu je však významnější hodnota relativního zlepšení.

Je zajímavé, že algoritmy s různorodější populací, obsahující více „genetického materiálu“, dosahují menšího zlepšení než s málo diverzifikovanou populací, přičemž tato populace je obsažena i v té různorodější. Důvodem by mohlo být to, že výsledky, které podávají algoritmy, jsou již na hranici, kde dosáhnout sebemenšího zlepšení je velmi těžké. Generování více jedinců zřejmě posouvá inicializační populaci blíže k této hranici, a tudíž je dosahované zlepšení menší.

Za povšimnutí také stojí, že nejlepším pro první populaci byl GA, zatímco pro druhou populaci GPSO, z čehož lze usoudit, že zrůznorodnění populace pomohlo GPSO více než GA, nebo naopak z pohledu zlepšení uškodilo GPSO méně než GA.

metoda	50_MIX_n30, EE2		50_MIX_n100, EE2	
	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
PSO 0,25 1,50 1,50 p63 g63 u=0,5	7,67	0	4,84	2
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl. pm=0,1	8,94	0	12,36	0
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1	9,13	0	12,87	0

Tabulka 25: Výsledky testů s populací obsahující 1-4 kvalitních jedinců.

	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
PSO 0,25 1,50 1,50 p63 g63 u=0,5	6,65	2	3,14	16
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl. pm=0,1	8,46	0	11,31	0
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1	8,33	0	10,86	0

Tabulka 26: Výsledky testů s populací obsahující 30 různorodých jedinců včetně 1-4 kvalitních z první populace.

Vyzkoušel jsem, jaký bude mít na algoritmy vliv, jestliže počáteční populace bude obsahovat pouze jednoho jedince, který byl vybrán jako nejlepší z první varianty populace. Výsledky v tabulce 27 potvrzují, že GPSO je závislejší na různorodosti populace více než GA. Úplné odstranění různorodosti z populace poškodilo výkon GA daleko méně než výkon GPSO.

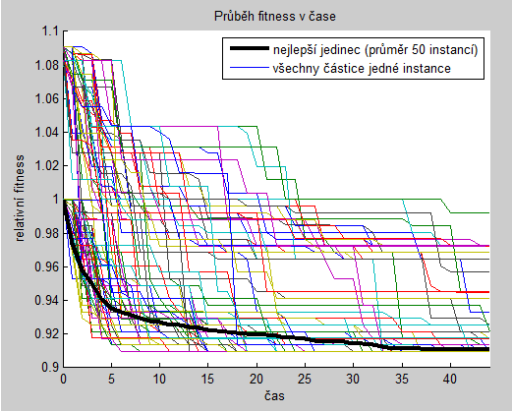
	50_MIX_n30, EE2		50_MIX_n100, EE2	
metoda	Ø zlepšení [%]	bez zlepšení [%]	Ø zlepšení [%]	bez zlepšení [%]
PSO 0,25 1,50 1,50 p63 g63 u=0,5	2,98	28	3,64	26
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl. pm=0,1	5,93	0	9,32	0
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1	7,45	0	10,30	0

Tabulka 27: Výsledky testů s populací obsahující pouze 1 nejlepšího jedince z první varianty populace.

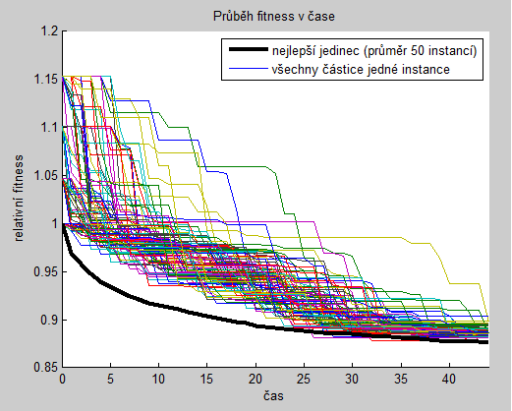
5.5 Pokles kritéria v čase

Dalším důležitým aspektem je rychlost poklesu kritéria v čase. Pro snazší vizualizaci jsem normoval fitness nejlepšího jedince v inicializační populaci na hodnotu 1. Na obr. 38 je zobrazen pokles normované hodnoty fitness všech 91 částic realizujících 44 kroků na problému s 30 úlohami. Jsou patrná různě velká skoková vylepšení s různě dlouhými intervaly, kdy ke zlepšení fitness částice nedošlo. Jako referenční hodnota je černě zobrazen průměr nejlepších hodnot fitness pro každou jednotku času, počítaný na 50 problémech. Obdobně je na obr. 39 zobrazeno chování pro 100 úloh.

Hodnota průměru nejlepších fitness nejprve rychle klesá, ale s přibývajícím časem se rychlost poklesu snižuje. Pro n = 100 (obr. 39) je patrné, že částice mají ještě potenciál snižovat hodnotu fitness i na konci výpočtu. Zdá se, že přidáním počtu kroků by mohla být hodnota kritéria ještě snížena.



Obr. 38: Graf vývoje všech částic GPSO pro 1 instanci v porovnání s průměrem nejlepších částic z 50 instancí MIX_n30 a kritériem EE2.



Obr. 39: Graf vývoje všech částic GPSO pro 1 instanci v porovnání s průměrem nejlepších částic z 50 instancí MIX_n30 a kritériem EE2.

5.6 Statistika profileru

Výpočet algoritmu je relativně časově náročný, proto mě zajímalo, které operace spotřebovávají nejvíce času. Pro testování jsem zvolil algoritmus GPSO a kritérium EE2. Nejprve jsem použil profiler na problémech 10_MIX_n30. Téměř 44 % času spotřebovala funkce generující report (hlavní část výpočtu fitness), další třetinu využila opravná procedura *fix_start_times*. Celých 9 % času spotřebovala mutace. Generování počáteční populace 1 – 4 jedinců přispělo méně než 8 % (obr. 40).

Function Name	Calls	**Total Time**	Self Time*	Total Time Plot (dark band = self time)
profiler	1	266.842 s	0.051 s	
pso	10	266.770 s	4.408 s	
...eport_energeticky2(problems(i),s,7,3)	40043	117.158 s	6.196 s	
generuj_report_energeticky2	40043	110.962 s	109.496 s	
fix_start_times	40040	91.651 s	76.450 s	
...ozptyl,2,1,1,0.1,mean(problems(i).p))	40040	24.779 s	6.668 s	
generuj_init_populaci_irs	10	21.824 s	0.069 s	
imsm_cdmpp_old	40	19.121 s	0.051 s	

Obr. 40: Výpis profileru pro problémy 10_MIX_n30.

Pro problémy 10_MIX_n100 byly výsledky podobné (obr. 41). Mírně klesl podíl času spotřebovaný funkcí generující report (na necelých 38 %) a mutací. Vzrostl naopak podíl času, který využívala opravná funkce (na 45 %) a funkce generující inicializační populaci (na 11 %). Z toho je zřejmé, že případné snahy o urychlení

algoritmu by měly být zaměřeny především na zrychlení výpočtu fitness funkce a opravné procedury *fix_start_times*.

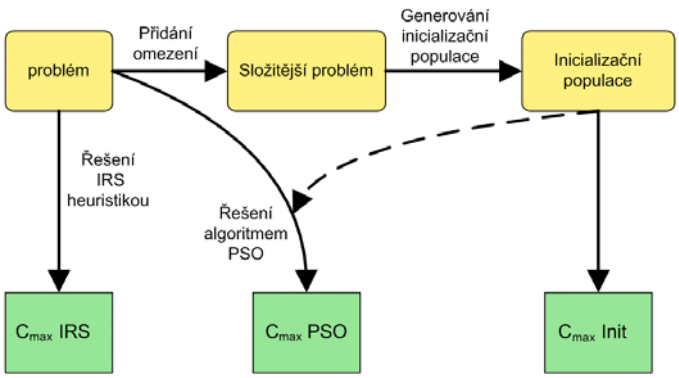
Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
profiler	1	981.950 s	0.108 s	
pso	10	981.817 s	5.119 s	
fix_start_times	40040	442.701 s	396.675 s	
...eport_energeticky2(problems(i),s,7,3)	37719	369.098 s	12.612 s	
generuj_report_energeticky2	37719	356.487 s	354.682 s	
generuj_init_populaci_irs	10	111.346 s	0.087 s	
imsm_cdmpp_old	40	102.340 s	0.100 s	

Obr. 41: Výpis profileru pro problémy 10_MIX_n100.

5.7 C_{max}

Další test jsem provedl na kritériu C_{max}. Motivací tohoto testu bylo srovnat výkonnost algoritmů s IRS heuristikou, což by mohlo dát odhad, jak kvalitní řešení poskytují naše algoritmy pro jiná kritéria. Vzhledem k charakteristice kritéria jsem vektor řešení v průběhu algoritmu reprezentoval activity listem.

Problémem je, že implementované algoritmy PSO, GPSO i GA využívají pro vygenerování počáteční populace IRS heuristiku. Uspokojivé řešení tohoto problému je zobrazeno na obr. 42: Řešená instance problému se nejprve zesložití přidáním dalších temporálních omezení. Z tohoto problému se IRS heuristikou vygeneruje inicializační populace, která se použije pro naše algoritmy.



Obr. 42: Schéma generování počáteční populace a získávání jednotlivých hodnot C_{max}.

Vzniknou tak tři hodnoty C_{max}:

- $C_{max}IRS$ – řešení původního problému IRS heuristikou,
- $C_{max}PSO$ – řešení algoritmem PSO, GPSO nebo GA s handicapovanou inicializační populací,
- $C_{max}Init$ – nejlepší řešení v handicapované inicializační populaci.

Z těchto tří hodnot jsem počítal zlepšení testovaných algoritmů vůči IRS heuristice podle vzorce (5.1).

$$zlepšení = (C_{\max}Init - C_{\max}PSO)/(C_{\max}Init - C_{\max}IRS) \cdot 100\% \qquad (5.1)$$

Dále jsem zaznamenal počet instancí bez zlepšení (přičemž zlepšení některých instancí nemusí být možné) a počet lepších řešení našich algoritmů oproti IRS heuristice ($C_{max}IRS < C_{max}PSO$), tabulka 28. Algoritmy dosahují velmi dobrých výsledků především pro instance se 100 úlohami, na kterých dosahují až průměrného zlepšení 93,5 %, přičemž zlepšení IRS heuristikou odpovídá 100 %. Pro problémy s 30 úlohami je průměrné zlepšení pouze 85 %, nicméně až pro 14 % instancí našel algoritmus PSO lepší řešení než IRS heuristika. Menší úspěšnost na problémech s n = 30 úlohami je způsobena upřednostňováním nastavení algoritmů, vhodných pro problémy se 100 úlohami. Výsledky pro 100 úloh jsou velmi pěkné a lze tedy očekávat, že hodnoty zlepšení, které algoritmy generují pro jiná kritéria, se pohybují blízko optima.

	50_MIX_n30, C_{max}			50_MIX_n100, C_{max}		
metoda	Ø zlepšení [%]	bez zlepšení [%]	lepších než IRS [%]	Ø zlepšení [%]	bez zlepšení [%]	lepších než IRS [%]
PSO 0,25 1,50 1,50 p63 g63 u=0,5	82,13	18	14	91,76	0	2
GPSO p91 g44 2bod 2:1:1 mut.p/2 nevyv. vš.násl. pm=0,1	79,29	16	12	93,54	0	6
GA p63 st-st mut.p nevyv. vš.násl. pm=0,1	85,06	14	12	92,80	0	4

Tabulka 28: Výsledky testu pro kritérium C_{max}.

5.8 Energetická efektivita 2

Tato sekce obsahuje výsledky závěrečných testů kritéria energetické efektivity 2 pro instance s n = 30, n = 100 a n = 200 úlohami. Přidal jsem statistiku úspěšného generování, což je poměr počtu vektorů, které byly po opravě funkcí *fix_start_times* platné, k počtu všech vygenerovaných vektorů. Přidal jsem i statistiku relativního zhoršení C_{max} nejlepšího nalezeného rozvrhu pro EE2, průměrný čas pro optimalizaci jedné instance a počet nejlepších rozvrhů nalezených každým algoritmem. Každou vybranou charakteristiku zdrojů řešených problémů jsem testoval odděleně. Výsledky testovaných charakteristik jsou v následujících tabulkách:

- XDXXX – tabulka 29,

- XDMXX – tabulka 30,
- XDXPX – tabulka 31,
- XDMPX – tabulka 32.

Pro přehlednost jsem testované varianty PSO, GPSO a GA s optimální konfigurací označil po řadě PSO*, GPSO* a GA*.

	50_XDXXX_n30, EE2						25_ XDXXX _n100, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	11,29	0	20	93,9	17,0	21	8,47	0	0	45,0	26,3	140
GPSO*	11,14	0	38	98,2	1,4	24	20,97	0	28	87,6	9,8	83
GA*	10,95	0	50	97,3	3,6	22	22,77	0	72	87,0	9,9	82

	12_ XDXXX _n200, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	4,66	0	0	14,8	19,8	550
GPSO*	25,16	0	50	74,3	9,9	281
GA*	24,41	0	50	72,2	8,5	260

Tabulka 29: Výsledky testu pro problémy typu XDXXX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem EE2.

	50_XDMXX_n30, EE2						25_ XDMXX _n100, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	9,35	0	14	82,3	16,7	32	3,28	6	0	14,3	8,6	215
GPSO*	12,07	0	50	95,4	2,9	34	15,17	0	44	70,7	2,8	147
GA*	11,49	0	36	93,9	4,5	32	15,28	0	56	67,1	2,1	150

	12_ XDMXX _n200, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	0	100	0	0,5	0	806
GPSO*	8,21	0	58	38,3	1,6	538
GA*	6,97	0	42	31,0	4,3	582

Tabulka 30: Výsledky testu pro problémy typu XDMXX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem EE2.

	50_XDXPX_n30, EE2						25_ XDXPX _n100, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	3,07	0	34	98,9	3,9	20	4,91	0	20	88,0	10,9	75
GPSO*	2,82	0	58	99,9	0,9	24	6,62	0	36	98,0	1,6	65
GA*	2,80	0	58	99,5	0,7	22	7,62	0	44	96,3	3,0	67

	12_ XDXPX _n200, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	1,42	33	0	61,3	5,7	378
GPSO*	10,26	0	50	93,9	0,4	185
GA*	9,75	0	50	88,4	2,2	224

Tabulka 31: Výsledky testu pro problémy typu XDXPX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem EE2.

	50_XDMPX_n30, EE2						25_ XDMPX _n100, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	4,81	0	20	97,4	9,4	28	2,92	8	0	51,8	6,4	187
GPSO*	5,45	0	30	99,0	0,7	32	9,60	0	60	89,8	0,8	113
GA*	5,51	0	50	98,0	1,8	32	8,54	0	40	84,3	2,6	126

	12_ XDMPX _n200, EE2					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	0,21	92	0	16,7	0,4	717
GPSO*	10,42	0	58	73,3	1,3	394
GA*	7,51	8	42	63,7	2,2	446

Tabulka 32: Výsledky testu pro problémy typu XDMPX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem EE2.

Výsledky ukazují, že PSO* si v testech vedlo velmi špatně, a to především na větších problémech se 100 a 200 úlohami. Jeho dalšími nevýhodami jsou výrazné prodlužování C_{max} a delší výpočet.

Pro problémy o velikosti 100 úloh byl ze čtyř testovaných množin na třech nejlepší GA*. S instancemi velikosti n = 200 si výrazně nejlépe poradilo GPSO*, přičemž generovalo o trochu více nejlepších rozvrhů než GA*. Na instancích o velikosti 100 úloh dosahovalo na jedné testovací množině nejlepších výsledků, přičemž pro ostatní tři byly výsledky jen o málo horší než pro GA*. Další zajímavostí je, že relativní zlepšení roste s velikostí problémů, což odpovídá tomu, že ve větších problémech s více úlohami je i větší prostor pro optimalizaci.

Výsledky také ukazují, že největšího zlepšení lze dosáhnout na problémech XDXXX, které je pro 200 úloh více než 25% oproti hodnotě generované IRS heuristikou. Velkého zlepšení lze dosáhnout i na problémech XDMXX. Nejmenšího, avšak pro 200 úloh stále většího než 10 %, pro problémy XDMPX a XDXPX.

Za povšimnutí také stojí fakt, že pro problémy XDMXX s rostoucím počtem úloh klesá poměr úspěšného generování výrazně rychleji než pro ostatní typy problémů. Pro problémy s 200 úlohami je téměř poloviční!

Dobrou vlastností GPSO* i GA* je, že příliš nezhoršují C_{max}. Intuitivním očekáváním je, že energeticky efektivní rozvrh nebude o mnoho delší než rozvrh vytvořený pro kritérium C_{max}. Průměrný přírůstek C_{max} je pro problémy typu XDXXX max. 10 %, pro XDMXX max. 5 % a pro XDXPX a XDMPX max. 3 %. Průměrný čas optimalizace jedné instance algoritmy GPSO* a GA* nepřekročí 10 minut pro žádný typ problému o velikosti do 200 úloh.

5.9 Earliness tardiness

V této části uvádím výsledky závěrečných testů kritéria earliness-tardiness. Vektory vah w a požadované termíny dokončení (due date) d byly generovány náhodně, přičemž due date byly generovány maximálně do hodnoty 1,05 · C_{max} nalezeného IRS heuristikou.

Stejně jako v předchozí sekci, i zde jsem přidal do tabulek stejné rozšiřující informace (statistiku úspěšného generování, relativní zhoršení C_{max}, atd.). Každou vybranou charakteristiku zdrojů řešených problémů jsem opět testoval odděleně. Výsledky testovaných charakteristik jsou v následujících tabulkách:

- XDXXX – tabulka 33,
- XDMXX – tabulka 34,
- XDXPX – tabulka 35,
- XDMPX – tabulka 36.

	50_XDXXX_n30, ET						25_XDXXX_n100, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	4,23	2	50	93,5	1,8	12	0,48	4	0	55,1	0,0	125
GPSO*	4,13	0	34	98,9	0,2	22	1,70	0	56	89,6	-0,6	77
GA*	3,76	0	16	98,0	0,3	20	1,39	0	44	84,6	-0,1	82

	12_XDXXX_n200, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	0,06	50	0	25,5	-0,0	518
GPSO*	1,68	0	92	77,8	-0,3	247
GA*	1,08	0	8	67,5	-0,3	292

Tabulka 33: Výsledky testu pro problémy typu XDXXX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem ET.

	50_XDMXX_n30, ET						25_XDMXX_n100, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	2,32	6	12	87,3	0,7	26	0,02	76	0	24,6	0	208
GPSO*	7,01	0	64	96,0	-0,9	27	6,74	0	56	73,3	-3,7	131
GA*	4,62	0	24	94,4	-0,5	26	5,66	0	44	70,7	-3,5	130

	12_XDMXX_n200, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	0	100	0	0,7	0	803
GPSO*	3,44	0	83	39,2	-1,7	536
GA*	2,33	0	17	32,2	-1,0	571

Tabulka 34: Výsledky testu pro problémy typu XDMXX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem ET.

	50_XDXPX_n30, ET						25_XDXPX_n100, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	3,82	0	78	98,8	1,6	17	1,59	0	64	91,2	0,1	59
GPSO*	2,65	0	18	99,9	0,2	21	1,47	0	28	98,5	0	60
GA*	2,17	0	4	99,6	0,5	19	1,05	0	8	96,5	0	62

	12_XDXPX_n200, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	0,43	8	25	69,5	0,1	323
GPSO*	1,00	0	67	94,2	-0,1	176
GA*	0,54	0	8	88,9	0,1	204

Tabulka 35: Výsledky testu pro problémy typu XDXPX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem ET.

	50_XDMPX_n30, ET						25_ XDMPX _n100, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	3,35	0	50	96,6	0,9	21	0,38	8	8	62,5	0,0	152
GPSO*	3,36	0	34	99,6	-0,6	25	3,32	0	60	93,2	-1,1	91
GA*	2,60	0	16	98,7	-0,2	24	2,51	0	24	88,5	-0,8	99

	12_XDMPX_n200, ET					
metoda	Ø zlepšení [%]	bez zlepšení [%]	nejlepších rozvrhů [%]	Ø úspěšné generování [%]	Ø zhoršení C_{max} [%]	Ø čas / 1 inst. [s]
PSO*	0,00	83	0	15,4	0	746
GPSO*	1,39	0	83	74,9	-1,0	361
GA*	1,01	8	17	61,3	-0,3	445

Tabulka 36: Výsledky testu pro problémy typu XDXPX o velikosti n = 30, n = 100 a n = 200 úloh s kritériem ET.

Celkové zlepšení kritéria bylo pro earliness-tardiness výrazně menší než pro EE2. To může být způsobeno nevhodností náhodného generování požadovaných termínů dokončení d. S rostoucí velikostí problémů relativní zlepšení klesá. Nejhůře si v testu vedlo PSO*, které nebylo schopno některé množiny problémů s 200 úlohami vůbec zlepšit. Jediný úspěch pro problémy se 100 úlohami zaznamenalo na problémech typu XDXPX, kde překonalo ostatní metody.

Nejlépe si v testu vedlo GPSO*, které dosahovalo vždy lepších výsledků než GA* pro všechny velikosti problémů a charakteristiky zdrojů. GPSO* také generovalo většinou výrazně více nejlepších řešení, než ostatní algoritmy. Největšího zlepšení dosahuje algoritmus GPSO* pro problémy typu XDMXX, které je pro n = 200 úloh a to 3,44 %. Pro n = 100 úloh to bylo 6,74 % a pro n = 30 úloh 7,01 %. Nejmenší zlepšení realizovaly algoritmy na problémech typu XDMPX.

Poměr úspěšného generování klesá s rostoucí velikostí problémů. I pro kritérium ET se ukazuje, že pro problémy typu XDMXX klesá až 2x rychleji než pro ostatní charakteristiky problémů. Hledání platného vektoru je pro tuto charakteristiku obtížnější, přičemž nezáleží na kritériu.

Algoritmy GPSO* i GA* při hledání efektivního řešení pro kritérium ET dokonce ve většině případů vylepšují C_{max} původního rozvrhu. To naznačuje, že termíny due date jsou stanoveny příliš malé a často se uplatňuje tardiness, což nutí algoritmy minimalizovat délku rozvrhu. Délka rozvrhu však již byla optimalizována IRS heuristikou. Nevhodnost stanovení termínů dokončení by vysvětlovala, proč je zlepšení pro earliness-tardiness menší než pro EE2. Velikost C_{max} rozvrhu vygenerovaného GPSO* může být v průměru až o 3,7 % menší než původní hodnota spočítaná IRS heuristikou (100 úloh XDMXX). Průměrný čas optimalizace jedné instance algoritmy GPSO* a GA* nepřekročí pro žádný typ problému o velikosti do 200 úloh 10 minut.

5.10 Problém míchání laků

V této části budu testovat algoritmy na problému „ze života“. Bohužel se mi nepodařilo nalézt problémy *RCPS | temp*, které nejsou generovány náhodně a pro které existuje řešení jiným algoritmem, se kterým bych mohl výsledky pro kritérium ET srovnat.

Jako nejvhodnější pro tento test jsem zvolil problém z produkce laků podrobně popisovaný v článku [17]. Vybral jsem problém obsahující 2 zakázky od každého ze tří typů laků (železný, bronzový a uni). Každý typ laku má svůj specifický proces výroby, přičemž délka jednotlivých úloh závisí na množství vyráběného laku. Rozpisem procesů výroby zakázek vznikl problém (označen LACQ) obsahující 46 úloh, 6 unárních zdrojů a jeden zdroj s neomezenou kapacitou. Problém jsem upravil tak, aby neobsahoval take-give zdroje ani changeover časy. Problém jsem testoval pro kritérium EE2.

Úlohy mají relativně dlouhé časy zpracování, a proto jsem testoval dvě různé délky zapínání resp. vypínání zdrojů: původní hodnoty 7 resp. 3 a další hodnoty 70 resp. 30. Algoritmy jsem na tento vybraný problém spustil 100x a zaznamenal jsem průměrné zlepšení a nejlepší dosažené zlepšení (tabulka 37).

Výsledky ukazují, že algoritmus GPSO* vykazuje nejlepší průměrné zlepšení energetické efektivity 2, které dosahovalo 17,8 %, resp. 19,78 %. Absolutně nejlepší rozvrh našel pro délky zapínání = 7 a vypínání = 3 algoritmus GPSO*, zatímco pro hodnoty 70 a 30 algoritmus GA*. Výsledky ukazují robustnost GPSO* a GA*, protože dosažená průměrná hodnota zlepšení je velmi blízko nejlepší nalezené hodnotě. Výpočet řešení problému trvá algoritmům méně než 3 minuty.

	LACQ_n46 (100x), délka zapínání = 7, délka vypínání = 3				LACQ_n46 (100x), délka zapínání = 70, délka vypínání = 30			
metoda	Ø zlepšení [%]	nejlepší zlepšení [%]	Ø úspěšné generování [%]	Ø čas / 1 inst. [s]	Ø zlepšení [%]	nejlepší zlepšení [%]	Ø úspěšné generování [%]	Ø čas / 1 inst. [s]
PSO*	2,95	5,03	50,3	167	14,90	18,34	24,5	179
GPSO*	17,80	18,91	76,5	164	19,78	21,84	68,1	171
GA*	16,47	18,01	79,9	156	19,32	22,28	79,7	157

Tabulka 37: Výsledky testu pro 100 spuštění algoritmů na problém výroby laků s 46 úlohami s kritériem EE2 a rozdílnými délkami zapínání a vypínání zdrojů.

Kapitola 6

Závěr

Úkolem této práce bylo seznámit se s metodou PSO, provést rešerši existujících algoritmů, implementovat algoritmus PSO pro řešení rozvrhování projektů s temporálními omezeními (RCPS | *temp*) a ověřit jeho výkonnost pro různá kritéria.

V práci jsem nejprve popsal řešený problém, diskutoval možnosti poskytované temporálními omezeními a vizualizoval prostor řešení. Provedl jsem rozsáhlou analýzu vlivu různých nastavení parametrů na chování PSO a přiblížil pokročilá vylepšení popisovaná jinými autory (UPSO, DPSO, IPSO, atp.). Implementoval jsem vylepšení, použitelná pro řešený problém.

Hlavním přínosem práce je vytvoření nového algoritmu GPSO kombinujícího dobré vlastnosti PSO s genetickými operátory nevyváženého dvoubodového křížení a náhodné mutace, využívaných v GA. Pro porovnání výsledků jsem implementoval i GA. Provedl jsem rozsáhlé testování různých konfigurací PSO, GPSO a GA (různé hodnoty parametrů φ, různé velikosti a způsoby inicializace populace, unifikované PSO, nulování počáteční rychlosti, optimalizaci po částech, vyvážené a nevyvážené křížení, jednobodové, dvoubodové a uniformní křížení, různou velikost a charakter mutace, generační a steady-state model GA).

Nejlepší konfigurace jsem otestoval na celkem dvanácti množinách problémů čtyřech typů o velikostech 30, 100 a 200 úloh s kritérii energetické efektivity a earliness-tardiness. Kritérium energetické efektivity (EE2) je zcela nové. Navrhl jsem ho tak, aby nejlépe hodnotilo rozvrhy, v nichž jsou úlohy na jednotlivých zdrojích seskupeny do bloků, v rámci kterých je minimální doba nečinnosti, přičemž mezi jednotlivými bloky je dostatečná prodleva, aby se zařízení mohlo přepnout do úsporného režimu. V testech dosáhl nejlepších výsledků algoritmus GPSO, který překonal PSO i GA. Na jedné množině testovacích problémů s velikostí 200 úloh zmenšil hodnotu kritéria EE2 v průměru o 25 %. Tím o čtvrtinu redukoval počet zapínání resp. vypínání zdrojů a délku prodlev mezi úlohami, které jsou příliš krátké pro přepnutí zařízení do úsporného režimu. Pro kritérium earliness-tardiness dosáhl algoritmus na jedné množině problémů o velikosti 100 úloh průměrného zlepšení o 6,7 %, přičemž zkrátil i průměrnou délku rozvrhu o 3,7 %. Menší zlepšení pro kritérium earliness-tardiness může být způsobeno generováním příliš brzkých termínů požadovaného dokončení (due

date). Test na problému výroby laků potvrzuje robustnost algoritmu GPSO, který pro tento problém dosáhl zlepšení až 21,8 % a v průměru 19,8 %.

Vzhledem k velmi dobrým výsledkům pro problémy RCPS | *temp* s kritérii EE2 a slibným výsledků pro earliness-tardiness by bylo vhodné algoritmus do budoucna rozšířit pro řešení více typů rozvrhovacích problémů. Rozšíření může být provedeno buď přidáním changeover časů a take-give zdrojů nebo úpravou pro odlišnější rozvrhovací problémy, jako je např. Job-shop.

Literatura

[1] Franco, E. G. – Zurita, F. L. T. – Delgadillo, G. M. A Genetic Algorithm for the Resource Constrained Project Scheduling Problem (RCPSP). 2007.

[2] Hartmann, S. A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling. 1997.

[3] Czogalla, J. – Fink, A. Particle Swarm Topologies for Resource Constrained Project Scheduling. 2009.

[4] Parsopoulos, K. E. – Vrahatis M. N. Studying the Performance of Unified Particle Swarm Optimization on the Single Machine Total Weighted Tardiness Problem. 2006.

[5] Hanzálek, Z. – Šůcha, P. Time Symmetry of Project Schedulig with Time Windows and Take-give Resources. 2009.

[6] Wang, Q. – Qi, J. Improved Particle Swarm Optimization for RCP Scheduling Problem. 2009.

[7] Anghinolfi, D. – Paolucci M. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. 2007.

[8] Policella, N. – Cesta, A. – Oddi, A. – Smith, S. F. From Precedence Constraint Posting to Partial Order Schedules: A CSP approach to Robust Scheduling. 2007.

[9] Lombardi, M. – Milano, M. A Precedence Constraint Posting Approach for the RCPSP with Time Lags and Variable Durations. 2009.

[10] Peteghem, V., v. PhD thesis. 2010. p. 27-66.

[11] Anghinolfi, D. – Boccalatte, A. – Paolucci, M. – Vecchiola, Ch. Performance Evaluation of an Adaptive Ant Colony Optimizatiom Applie to Single Machine Scheduling. 2008.

[12] Clerc, M. – Kennedy, J. The Particle Swarm–Explosion, Stability, and Convergence in Multidimensional Complex Space. 2002.

[13] Šochman, J. – Franc, V. Cvičení z RPZ – Maximálně věrohodný odhad. 2006. Dostupné z: http://cmp.felk.cvut.cz/cmp/courses/recognition/Labs/mlodhad/ml.pdf [stav z 1.5.2011]

[14] Franck, B. – Neumann, K. – Schwindt, Ch., Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. 2000.

[15] Hanzálek, Z. – Šůcha, P. Přednáška z Kombinatorické optimalizace: Rozvrhování. 2010.

[16] Rowe, A. – Lakshmanan, K. – Zhu, H. – Rajkumar, R. Rate-Harmonized Scheduling for Saving Energy.

[17] Kelbel, J. – Hanzálek, Z. Solving production scheduling with earliness/tardiness penalties by constraint programming. 2009.

[18] Bartusch, M. – Möhring, R. H. – Radermacher, F. J. Scheduling project networks with resource constraints and time windows. 1988.

[19] TORSCHE Scheduling Toolbox for Matlab.
Dostupné z: http://rtime.felk.cvut.cz/scheduling-toolbox/ [stav z 1.5.2011]

[20] Šůcha P. IRS heuristika (funkce pro Matlab).

[21] Hanzálek – Z. Kelbel, J. – Šůcha P. Generátor náhodných instancí. Instance problému výroby laků.

Dodatek A

Seznam použitých proměnných a zkratek

δ	vzdálenost particle-best od neighborhood-best
φ, φ_0, φ_1, φ_2	parametry PSO určující váhu vektorů v_i, p_i a g_i
λ_i	poměr earliness vůči tardiness úlohy i
2bod 2:1:1	nastavení – nevyvážené 2bodové křížení v poměru 2:1:1
2bod.	nastavení – dvoubodové křížení
ACO	Ant Colony Optimization
activity list	viz. priority value
c_{ik}	požadavky úlohy i na zdroj k
C_{max}	kritérium délky rozvrhu
d_i	požadovaný termín dokončení úlohy i (due date)
DPSO	Discrete Particle Swarm Optimization
EE1	kritérium energetické efektivity 1
EE2	kritérium energetické efektivity 2
ET	kritérium earliness-tardiness
g44, g63, g91	nastavení – počet generací
GA	genetický algoritmus
GA*	GA s optimální konfigurací
g_i	neighborhood-best pozice částice i
global-best	nejlepší pozice nalezená někým ze sousedů v PSO s plně propojenou topologií
GPSO	Genetic Particle Swarm Optimization
GPSO*	GPSO s optimální konfigurací
changeover čas	minimální doba mezi dvěma bezprostředně následujícími úlohami i a j zpracovávanými na stejné jednotce zdroje, po kterou nesmí být zdroj využíván.
IPSO	Improved Particle Swarm Optimization
LACQ	problém míchání laků
local-best	nejlepší pozice nalezená někým ze sousedů v PSO s řídce propojenou topologií

LST	pravidlo výběru úlohy s nejnižší hodnotou nejzazšího možného startu
MRCPSP	Multi Mode Resource Constrained Project Scheduling
MTS	pravidlo výběru úlohy s největším počtem následníků
multiproc	maximální počet zdrojů požadovaných jednou úlohou
mut.5	nastavení – mutace v intevalu [-5; 5]
mut.p	nastavení – mutace v intevalu [$-p_ø$; $p_ø$]
mut.p/2	nastavení – mutace v intevalu [$-p_ø/2$; $p_ø/2$]
n	počet úloh
neg_{max}	maximální délka záporné hrany
neg_{num}	počet záporných hran
neighborhood-best	nejlepší pozice nalezená někým ze sousedů
nevyv.	nastavení – nevyvážená mutace upřednostňující posun úlohy blíže k začátku rozvrhu v poměru 2:1
onOff	nastavení – změny vektoru startovních časů po částech
$p_ø$	průměrný čas zpracování úlohy v problému
p44, p63, p91	nastavení – velikost populace
particle-best	nejlepší pozice nalezená částicí
PCA	Principal Component Analysis
p_i	čas zpracování úlohy p; particle-best pozice částice i
pk=0,5, pk=1	nastavení – pravděpodobnost křížení
pm=0,1, pm=0,3	nastavení – pravděpodobnost mutace
p_{max}	maximální čas zpracování
pos_{max}	maximální délka kladné hrany
pos_{num}	počet kladných hran
$proc_{cap}$	maximální kapacita procesoru
$proc_{num}$	maximální počet dedikovaných procesorů
priority rule	n-posloupnost pravidel výběru úlohy
priority value	n-posloupnost reálných čísel z intervalu (0; 1), reprezentujících prioritu zařazení dané úlohy
PSO	Particle Swarm Optimization
PSO*	PSO s optimální konfigurací
R	množina zdrojů
RCPS | *temp*	problém rozvrhování projektů s omezenými zdroji a temporálními omezeními
r_i	kapacita zdroje i
ruleta	nastavení – ruletový výběr
s	vektor startovních časů
s^*	optimální vektor startovních časů
s_i, s_j	startovní čas úlohy i resp. j
st-st	nastavení – steady-state model
steady-state	model GA, vysvětleno v kapitole 3.3
t	čas
take-give zdroje	speciální přídavné zdroje, jejichž využívání není přímo závislé na čase. Fungují tak, že jedna úloha při svém startu zabere take-give resource a využívá ho nejen během času zpracování p, ale pokračuje v jeho užívání, dokud ho některá jiná úloha svým dokončením neuvolní.
u	unifikační parametr pro UPSO
u=0	nastavení – local-best

u=0,25, u=0,5	nastavení – unifikované PSO
unif.	nastavení – uniformní křížení
UPSO	Unified Particle Swarm Optimization
V	množina úloh
v=0	nastavení – nulová počáteční rychlost
v_i	rychlost (velocity) částice i
v_{max}	vektor maximálních rychlostí v každé dimenzi
vš.násl.	nastavení – mutace nejen vybrané úlohy, ale i všech úloh s vyšším startovním časem
W	matice temporálních omezení
w_i	důležitost (váha) úlohy i
XDMPX	specifikace problémů s několika typy zdrojů, každý zdroj může mít jinou kapacitu a úlohy jsou multiprocesorové
XDMXX	specifikace problémů s několika typy zdrojů, každý zdroj může mít jinou kapacitu
XDXPX	specifikace problémů s několika typy zdrojů, každý pouze s unární kapacitou, avšak úlohy jsou multiprocesorové
XDXXX	specifikace problémů s několika typy zdrojů, každý pouze s unární kapacitou
x_i	vektor řešení částice i
XXXXX	specifikace problémů s jedním zdrojem s unární kapacitou

Dodatek B

Obsah CD

Na přiloženém CD se nalézají tyto adresáře:

Literatura obsahuje použité zdroje,

Text obsahuje text diplomové práce ve formátu PDF a DOC,

Zdrojove_kody Všechny zdrojové kódy algoritmů jsou samostatně spustitelné v prostředí Matlab (přípona .m). Před spuštěním je nutné nastavit cestu do všech podadresářů se zdrojovými soubory. Zdrojové kódy jsou rozděleny do těchto podadresářů:

- **Externi** soubory, které nebyly vytvořeny autorem práce (generátor instancí, IRS heuristika),
- **Fitness** generování reportů a výpočet fitness funkcí (C_{max}, EE2, ET),
- **GA** funkce pro genetický algoritmus a operátory pro GPSO,
- **Generator** obslužné funkce generátoru testovacích problémů, včetně předpočítávání inicializační populace,
- **IRS** obslužné funkce IRS heuristiky,
- **Pomocne** další nezařazené funkce,
- **Problemy** testovací množiny problémů,
- **PSO** funkce pro PSO a GPSO,
- **Testy** testovací funkce, výsledky testů a kódy pro generování obrázků použitých v práci.