Paralelní návrh pro řešení Resource-Constrained Project Scheduling Problem

Martin Lukeš

Kombinatorická optimalizace

2013

Popis problému a kategorizace

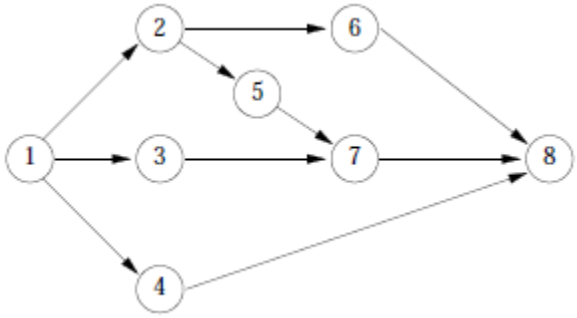
Resource Constraint Project Sheduling problém (RCPSP) je instanciací obecného Resource Constraint Scheduling Problem(RCSP), který je podtřídou problémů scheduling problémů, jež řadíme do kategorie NP složitých problémů. Kromě RCPSP jsou v této kategorii problémy job-shop flow-shop a production scheduling.

RCSP je v obecné formě definován takto: Mějme množinu aktivit, množinu úloh, množinu úloh, množinu zdrojů a míru vykonávání rozvrhu. Jakým způsobem nejlépe přiřadit aktivity zdrojům (nebo zdroje aktivitám) v dané časy, aby byl rozvrh co nejlepší?

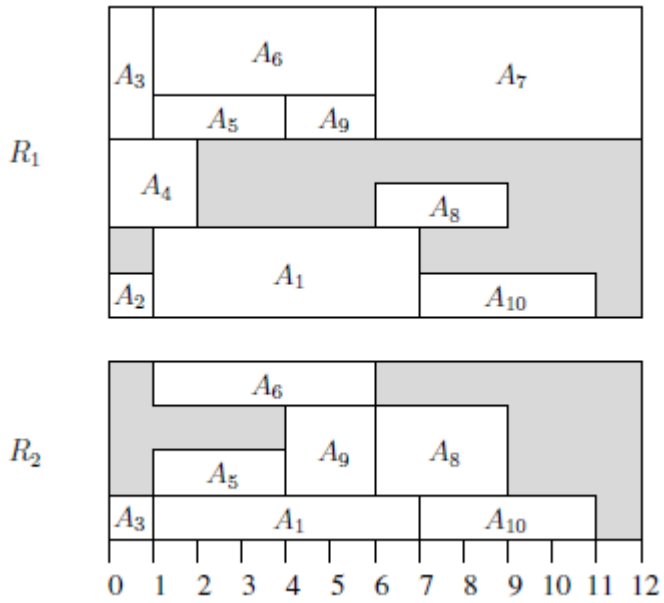
RCPSP potom zavádí další omezení, kterým je nepreemptivnost jednotlivých aktivit a následnost jednotlivých aktivit. Nepreemptivnost vyjadřuje, že po započetí dané aktivity tato aktivita nemůže být pozastavena před jejím dokončením. Následnost aktivit vyjadřuje, že daná aktivita nemůže být v rozvrhu dříve než jsou dokončeny všechny její předchůdci. Následnost aktivit je nejčastěji zobrazována pomocí acyklického grafu aktivit, kde uzly symbolizují jednotlivé aktivity a hrany následnost. Existují dvě varianty RCPSP : multi – mode a single – RCPSP. Single mode varianta předpokládá v jediný způsob vyřešení aktivity, kdežto multi-mode metoda umožňuje vykonávání aktivity ve více módech oddělených například podle délky trvání aktivity a jí přiřazených zdrojů.

Formální definice Single mode RCPSP

Nechť T je množina tasků $T = \{t_1, \dots t_n\}$
Activity on network (AON) síť tasků vyjádřenou pomocí acyklického grafu, na každé cestě od počátečního tasku do koncového tasku platí, že mezi tasky je následnost, tzn task t_i předchází tasku t_j, pokud je $i < j$
délku tasků $R = \{r_1, \dots r_m\}$
závislost tasků na zdrojích definovaných maticí K = m x n, kdy hodnota K_{ji} definuje počet jednotek zdroje r_j potřebných pro dokončení tasku t_i



Obr 1 . Ukázka AON – obrázek převzaný z [Sprecher97]



Obr 2 – Ukázka Rozvrhu s zobrazenými závislostmi tasků na zdrojích R_1 a R_2, obr byl přejat z [Christian ARTIGUES]

Cílem této práce bylo nalezení paralelního single-mode algoritmu pro řešení tohoto problému. Problém RCPSP byl řešen například pomocí enumerativní metody, Branch and bound algoritmu, metody CPM (Critical Path method), genetických algoritmů a Ant Colony Optisation přístupu.

Model

Před řešením úlohy je nutné definovat si struktury/model instance tasků, zachytit precedenční vztahy mezi tasky, množství zdrojů K_{ji} potřebných k provedení tasku *i* pro každý task *j*, délku trvání tasku a množství obnovitelných zdrojů jednotlivých typů, které jsou dostupném v každém časovém okamžiku.

Nejprve reprezentujeme statické prvky úlohy. Precedenční závislost tasků se dá namodelovat jak již bylo zmíněno acyklickým grafem. V implementaci budeme tento graf reprezentovat pomocí pole seznamů následníků A, protože matice sousednosti takového grafu může být maximálně horní trojúhelníková a typicky je tento graf řidší. Pro grafy s horní trojúhelníkovou maticí sousednosti existuje pouze jedno řešení – viz sekce inicializace, pro hustější precedenční grafy se celkový počet řešení limitně blíží jedné.

Množství obnovitelných zdrojů v daném čase se dá reprezentovat pomocí celočíselného pole hodnot o délce *n*.

Délku trvání jednotlivých tasků P – tzv. processing time můžeme taktéž reprezentovat pomocí pole celočíselných hodnot o délce *m.*

Závislosti tasků na zdrojích se typicky reprezentují maticí K o

rozměrech *n x m* celočíselných hodnot, kdy závislost tasku *k* na zdroji *l* najdeme v K na pozici [l, k], přičemž *n* je počet zdrojů a *m* počet tasků. V literatuře jsou typicky spojeny matice K s polem délek trvání tasků P.

Jsou dvě možné reprezentace instance dané úlohy. První z nich reprezentuje úlohy pomocí uspořádané k-tice celočíselných hodnot, reprezentující počáteční časy jednotlivých tasků, druhý způsob je reprezentace také uspořádanou k-ticí reprezentující pořadí jednotlivých tasků v rozvrhu. Tato k-tice na rozdíl od první je jen permutací hodnot *1*... *n* , kdy n je počet tasků. Vybral jsem si druhý způsob, protože se mi s ním lépe pracuje – viz sekce genetický algoritmus – definice operátorů.

Paralení algoritmy

V rámci této práce jsem prostudoval materiály profesora Tvrdíka k předmětu MI-PAR - Paralelní algoritmy a systémy a materiály k předmětu z FJFI Paralelní algoritmy a architektury. Následující text je vytažen z materiálů prof Tvrdíka.

Profesor Tvrdík uvádí měřítka výkonnosti sekvenčních algoritmů – asymptotickou dobu výpočtu algoritmu - T(n), paměťovou náročnost Sp(n) při výpočtu algoritmu a porovnává je s měřítky výkonnosti paralelních algoritmů.

Sekvenční algoritmus:

T(n)

Sp(n)

Paralelní algoritmy mají kromě asymptotické doby výpočtu a paměťové náročnosti i počet procesorů, které vykonávají danou úlohu. Délka trvání výpočtu je potom závislá nejen na velikosti vstupních dat, ale i na počtu procesorů p, které úlohu počítají.

T(n, p)

Formální definice prof Tvrdíka: T(n, p) je čas, který uplynul od začátku paralelního výpočtu do okamžiku, kdy poslední (nejpomalejší) procesor skončil výpočet.

T(n, p) je měřen čítáním:
1. výpočetních kroků: aritmetické, logické, paměťové operace
+
2. komunikačních kroků: přenos a výměna dat mezi procesory

Pokud si definujeme horní mez výpočetní složitosti SU(n), pak můžeme definovat paralelní zrychlení S(n, p):

S(n, p) =SU(n)/T(n, p) <= p

Lineární zrychlení:

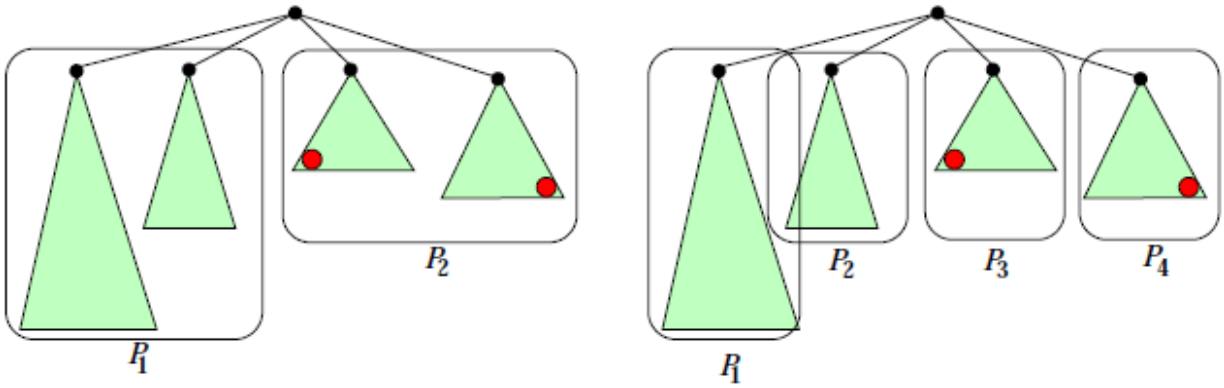
S(n, p) = Q(p)

Superlineární zrychlení :

• Způsobeno určitými charakteristikami HW, které staví sekvenční algoritmy do nevýhody. Typická situace: algoritmus je paměťově náročnější, než je kapacita paměti na 1-procesorovém systému, kdežto souhrnná

kapacita pamětí paralelního systému stačí.

- Vzniká díky anomáliím při paralelním prohledávání kombinatorického stavového prostoru



Obrázek 3 Anomální prohledávání stavového prostoru

Na obr 3 vidíme ukázku statického prohledávání stavového prostoru na 2 a 4 procesorech. Anomálie spočívá v tom, že řešení je nalezeno stejně rychle 2 jako 4 procesory. Obrázek byl přejat z [Tvrdík2009]

Paralelní cena

$C(n, p) = p \times T(n, p)$.

Cenově optimální algoritmus

$C(n, p) = Q(SU(n)))$.

Původním cílem této práce bylo nalezení paralelního cenově optimálního algoritmu pro nalezení minimální délky rozvrhu RCPSP.

V přednáškách z FJFI jsem našel následující popis vývoje paralelního algoritmu

„Vývoj paralelního algoritmu je nunto chápat jako optimalizaci.

1. nejprve vždy vyvíjíme co nejjednodušší sekvenční algoritmus bez optimalizací za každou cenu se vyhýbáme předčasné optimalizaci, ta může zcela zbytečně poničit čistý návrh algoritmu bez základní jenoduché verze kódu nemůžeme poměřit přínos paralelizace
2. máme-li základní funkční kód, který není dostatečně výkonný, přistupujeme k optimalizacím,pokud je to možné, optimalizujeme jen sekvenční kód, pokud to nepostačuje, přikročíme k paralelizaci
3. během implementace optimalizací provádíme průběžné testy a kontrolujeme, zda optimalizovaný kód dává stále správné výsledky
4. nakonec poměřujeme přínos optimalizace tj. Výslednou efektivitu

paralelizace“

Problémem se v této chvíli stala neznalost sekvenčního algoritmu pro řešení dané úlohy, proto jsem se snažil najít takový algoritmus.

Výzkum v oblasti RCPSP

Většina článků pojednávající o nadmnožině RCPSP - RCPS (Resource constraint scheduling problem) se zabývá rozdělením a specifikováním problému podle všemožných omezení. Kritérii rozdělení jsou multi-mode vs single-mode, obnovitelné vs neobnovitelné zdroje, preemptivita jednotlivých tasků.

Tyto problémy jsou řešeny pomocí matematických metod a ILP, enumerativních, neuronových sítí, genetických algoritmů a jiných heureistik.

Vzhledem k nutnosti adaptability plánu v aplikaci algoritmů na reálné problémy se většina autorů odklání od využívání exaktních metod sloužících k nalezení optimálního řešení a přiklání se k heureistickým metodám, jejichž použití sice nezaručuje nalezení optimálního řešení(pokud existuje), ale hledá „dobré“ řešení vyhovující daným podmínkám.

Sprecher popsal v článku [Sprecher97] Branch and Bound algoritmus a množinu predikátů, jež pomáhají ořezávat prostor řešení(Extended and Simplified Single Enumeration Rule, Local Left shift rule, Contraction Rule, Set Base Dominance, Simple Permutation Rule, Non-optimality shift rule a Cutset dominance).

Článek Methewa Bartschi Walla je velice dobrým úvodem do celé problematiky RCSP, nadmnožiny RCPSP. Autor uvádí problémy z rodiny RCSP jako je flow-shop, job-shop a open-shop. Autor dále vysvětluje pojmy single-mode a multi-mode scheduling. Článek se věnuje i reálné aplikaci algoritmů na problémy a autor uvádí, že exaktní metody nejsou adaptabilní pokud se změní přidají omezující podmínky na výsledný rozvrh. Článek cituje Lawrence Davise, který nepovažuje ILP, matematické metody řešení a metody tradičního operačního výzkumu za vhodné k řešení problému kvůli velkým omezením na reálné rozvrhy. Autor se dále věnuje popisu Steady-state a struggle genetic algoritmů a jejich testování na benchmarcích. Výsledky čistého genetických algoritmů nejsou na benchmarkových instancích problému nijak excelentní, algoritmus nalézá 2 až 5 x horší řešení než nejlepší řešení problém. Výjimkou je multi mode varianta algoritmu, kde má genetický algoritmus srovnatelné výsledky. Autor očekává vylepšení výkonnosti algoritmů, pokud by byl genetický algoritmus používán hybridně s jiným typem algoritmu nebo pokud by byly parametry algoritmu vhodněji vybrány.

Selcuk Colak v článku [Selcuk colak et al] popsal hybridní přístup HNA k problému pomocí adaptive-learning approach (ALA) a augmented neural network (AugNN), přičemž druhý genetický algoritmus pracuje paralelně ve dvou oddělených populacích. AugNN nicméně nezaručuje optimalitu nalezeného řešení. Autor článku řadí podle výsledků na benchmarkových úlohách algoritmus mezi nejlepší algoritmy. Jakožto první aplikaci neuronových sítí do vydání článku podává algoritmus srovnatelné výsledky s nejlepšími technikami typu tabu search, ant colonies, simulováné žíhání, genetické algorithmy a scatter search. Autor očekává vývoj dalších hybridních technik zahrnující HNA a jiné výkonné algoritmy.

Problém jsem se rozhodl řešit genetickým algoritmem.

Genetické algoritmy (GA)

V diplomové práci Paralelní Genetické Algoritmy Ing. Petra Pošíka, Ph.D. Jsem hledal inspiraci dále. Následující text je citací z této práce.

Podle Pošíka se genetické algoritmy inspirovaly v přírodě a přírodních vědách. Snaží se napodobit evoluci takovým způsobem, jakým probíhá v přírodě. Jako teoretický podklad užívají Mendelovu teorii genetiky a Darwinovu teorii přirozeného výběru. Teorie přirozeného výběru hlásá, zjednodušeně řečeno, přežití silnějšího, tzn., že rychlejší, silnější a inteligentnější jedinci mají větší šanci na přežití v dynamickém a neustále se měnícím prostředí, než jedinci pomalejší, slabší a hloupější. Tito „horší“ jedinci ve svém prostředí přežívají spíše díky tomu, že mají štěstí, než díky svým vlastnostem. Proto se reprodukce (vytváření další generace) zúčastňují také, avšak v menší míře, než jedinci „lepší“. G. Mendel zase objevil mechanismus přenosu charakterových vlastností z rodiče na potomka.

Později se ukázalo, že za jednotlivé vlastnosti jsou zodpovědné geny, které jsou uspořádány lineárně v chromozomech.
V oblasti GA se proto také používají výrazové prostředky převzaté z těchto biologických disciplín. Mluvíme o *jedincích* (nebo o *genotypech, chromozomech, strukturách* či *řetězcích*) v populaci. (Pro potřeby GA se většinou dělá určité zjednodušení: každý biologický jedinec má obyčejně více druhů chromozomů – zde se předpokládají jedinci s jediným chromozomem, proto je možné oba pojmy „sjednotit“.

Popsaný průběh GA dle Pošíka:
k = 0

Inicializace P(k)
Ohodnocení P(k)
Opakuj
    k = *k*+1
    Selekce P(k) z jedinců v P(k-1)
    Rekombinace P(k)
    Ohodnocení P(k)
Dokud není splněna terminální podmínka

Rekombinací rozumějme křížení. Selekce je výběr vhodných – přeživších jedinců do další generace.

K řešení určitého problému pomocí Genetického Algoritmu je třeba splnit následující požadavky [Michalewicz1995]:

1) Najít vhodnou reprezentaci potenciálních řešení problému (vhodně zvolit „formát“ chromozomu)
2) Najít způsob, jak vytvořit počáteční populaci chromozomů tak, aby představovaly přípustná řešení

3) Sestavit účelovou funkci, díky níž budeme schopni rozhodnout, který jedinec je „lepší“ a který „horší“

4) Zvolit nebo vytvořit vhodné genetické operátory, které ovlivňují tvorbu nových potomků

5) Vhodně nastavit různé parametry používané v GA (velikost populace, pravděpodobnosti uplatnění genetických operátorů, apod.)

Paralelní model vs. paralelní implementace

Při použití termínu „paralelní genetický algoritmus“ je třeba rozlišovat mezi případy, kdy slovo „paralelní“ označuje paralelní model GA a kdy paralelní implementaci GA.

- **Modely.** Sekvenční model GA (přesněji nazývaný globální model) má jedinou populaci a neklade žádná omezení na to, který jedinec může rekombinovat s jiným. Globální model GA se tradičně slučuje s jednoduchým GA, který bývá popisován v literatuře. V paralelním modelu GA je buď více populací (ostrovní model) nebo jedna populace rozdělená do mnoha subpopulací. Mluvíme-li ovšem o modelu, neklademe žádná omezení na skutečnou implementaci.
- **Implementace.** Sekvenční i paralelní modely GA mohou být implementovány paralelně i sekvenčně. Sekvenční implementace globálního modelu je tradiční přístup popisovaný ve většině literatury o GA. Jeden proces běžící na jednom uniprocesoru provádí všechny výpočty. V paralelní implementaci globálního modelu mohou být některé nebo všechny kroky GA (selekce, křížení, mutace, ohodnocování) prováděny současně v několika procesech běžících na paralelním počítači nebo na síti spolupracujících počítačů.

Pro účely této práce jsem definoval a následně se snažil implementovat paralalní model genetických algoritmů.

Pošík dále definuje operátor migrace. Migrace je výměna chromozomů mezi jednotlivými populacemi GA a její výhodnost spočívá v principu paralelního modelu GA, který oproti modelu sekvenčnímu nepotřebuje tak velké populace, populace jsou menší, je jich více a proto rychleji prohledávají stavový prostor řešení úlohy a v tom spočívá jádro problému – genetické algoritmy jsou heureistikami, které nemají obecně zaručeno nalezení globálního extrému dané účelové funkce, ale mají zaručeno nalezení lokálního extrému. Bez migrace můžou jednotlivé populace rychleji dokonvergovat k lokálním extrémům, ale být stále velmi vzdálené od extrému globálního. Migrace má tedy stejně jako mutace napomoci k odstranění-obejití lokálních extrémů a v některých implementacích i mutaci nahrazuje.

Migrace se dá dělit na dva typy – synchronní asynchronní migrace. Synchronní migrace probíhá v pravidelných intervalech. Její výhodou je snadná implementace. Bohužel nezohledňuje správný čas pro předávání chromozomů, takže migrace může zhoršovat efektivitu algoritmu. Další nevýhodou je potom nutnost definovat interval (počet populací) mezi jednotlivými migracemi – tento interval je jedním z parametrů GA. Synchronní migrace naráží na dva problémy – příliš častá migrace a příliš řídká migrace.

Asynchronní migrace je vyvolána určitými akcemi, typicky se populace vyvíjí až dokonvergují k nějakému lokálnímu extrému a jakmile máme takto tzv. „zdegenerované populace“, tak proběhne migrace. Pro náš problém by byla nejspíše vhodná migrace synchronní, jelikož máme minimalizovat délku rozvrhu. Při migraci se dají uplatnit stejná pravidla pro přijetí chromozomů jiným kmenem jako je uvedeno u operátoru selekce. Mám domněnku, že asynchronní migrace by vedla jen k pomalejší konvergenci a nezkvalitnila by řešení.

Definice fitness Funkce

Fitness funkcí jednotlivého chromozomu je délka trvání rozvrhu, který tento chromozom reprezentuje. Pokud chromozom reprezentuje nevalidní rozvrh, bylo by nutné definovat opravnou funkci, která by upravila hodnotu fitness funkce daného nevalidního rozvrhu. Z podstaty úlohy je tato hodnota je „inverzní“ vůči své definici, tzn čím nižší, tím lepší (fitness funkci používáme při selekci a migraci a u obou dvou operátorů potřebujeme fitness funkci pouze pro seřazení tasků – viz definice operátoru selekce). Pokud bychom nechtěli jen pořadí tasků, musela by být fitness hodnota definována jako například rozdíl maximální délky trvání rozvrhu (v daném čase se provádí vždy jen jeden task) a od této hodnoty odečíst naši fitness funkci. Další možností by byl podíl $1/f_n$, ale bylo by nutné prozkoumat experimentálně tyto hodnoty.

Rozvrh z našeho modelu extrahujeme tak, že pro danou entici uspořádaných tasků vkládá postupně tasky za sebe, s tím, že každý task musí začínat nejdříve jako jeho předchůdce v n-tici, ale je také nutné nastavit jeho scheduling time na konec posledního z jeho předchůdců, co se týče AON grafu.

Selekce

Selekce je výběr jedinců, kteří přežijí do nové generace.

Ruletová selekce – jediným ukazatelem míry přežití daného jedince je jeho fitness hodnota. Tzn, pokud bychom měli jedince s fitness hodnotou f_i , pak pravděpodobnost přežití tohoto jedince bude $p_i = f_i / \sum f_n$ pro všechna přípustná n. Takováto selekce má nevýhodu v nutnosti velké populace, což pravděpodobně není dobré pro paralelní genetický model.

Dalším typem selekce je pořadová selekce, kdy pravděpodobnost přežití nezávisí na fitness hodnotě daného jedince, ale na pořadí této fitness hodnoty v seřazeném seznamu chromozomů podle fitness hodnoty. Pokud tedy má každý task číslo pořadí o_i , pak se pravděpodobnost přežití daného

jedince vypočítá $p_i = o_i / \sum o_n$ pro všechna přípustná *n.*

Jelikož zadání úlohy je minimalizační, pak budou tasky seřazeny vzestupně – od tasku s nejmenší fitness hodnotou.

Další vhodný mechanismus je elitismus, kdy se nevybere celá máme starou generaci g_i a chceme vytvořit novou generaci g_{i+1}, ale snažíme se ponechat nejlepší jedince v generaci, proto necháme *n* jedinců s nejlepší – v našem případě nejnižší fitness hodnotou a doplníme podle náhodně vybraných chromozomů ze zbytku podle naší selekční metody.

Definice Operátorů GA pro RCPS

Definice Křížení

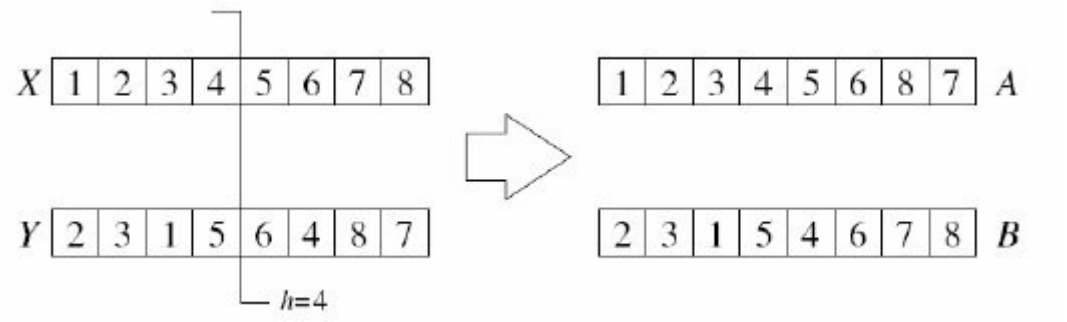
Křížení je genetický operátor, který je zodpovědný za vygenerování souboru nových chromozomů z většinou 2 (v článcích byla zmínka o křížení 3) chromozomů. Toto křížení se dělí na jednobodové, dvoubodové, rovnoměrné. Jednobodové křížení dvou chromozomů je takové, které nalezne pozici v chromozomech (oba chromozomy mají stejnou délku) a rozdělí chromozomy na dvě části – první část daného chromozomu je od počátku až do oddělovače a druhá od oddělovače až do konce chromozomu.

První část prvního chromozomu se vloží do prvního potomka a spojí se s druhou částí druhého potomka, druhý potomek vznikne spojením první části druhého potomka s druhou částí prvního potomka. Dvoubodové a vícebodové křížení by fungovalo analogicky s větším počtem oddělovačů.

Rovnoměrné křížení je takové, kdy každý prvek chromozomu – alela má 50 % šanci se objevit v prvním potomkovi a stejnou pravdepodobnost vložení do potomka druhého. Tyto typy křížení se používají pro křížení chromozomů složených z libovolných alel bez jakéhokoliv dalšího omezení.

Bohužel tyto typy křížení nejsou úplně vhodné pro tuto reprezentaci úlohy. Chromozom je totiž sekvence unikátních prvků a pokud bychom měli 2 chromozomy a oddělovač pro jednobodové křížení, nemáme zaručeno, že první část prvního chromozomu je permutací tasků v první části druhého chromozomu. Proto je nutné upravit tento operátor následovně – první část prvního chromozomu se vloží do prvního potomka a doplní se tasky z druhého potomka, které nejsou již v části první v pořadí, jak jsou uložené v potomkovi druhém. Tento postup je následně aplikován na potomka druhého.

Za tuto definici křížení děkuji Jiřímu Hrazdilovi, z jehož seminární práce jsem čerpal inspiraci. Z jednoduchého pozorování odhalíme zajímavou vlastnost uvedeného křížení – pokud jsou oba rodiče validní co se týče unikátnosti tasků, ale i precendenčních závislostí, potom jsou i jejich potomci také validní a tudíž v takovém případě není nutné definovat opravnou funkci fitness funkce.



Obr 4. Křížení – převzáno z Hrazdil

Definice operátoru mutace

Mutace je náhodná změna chromozomu, která jak již bylo zmíněno zabraňuje konvergenci genetického algoritmu k lokálnímu extrému. Často je mutace definovaná jako změna hodnoty nějaké alely chromozomu, nicméně tato transformace chromozomu dává vzniknout chromozomu složenému z neunikátních hodnot.

Proto je dobrým nápadem místo náhodné změny hodnoty udělat náhodnou transpozici dvou alel v chromozomu např doprava. K tomu je nutné definovat počet míst, o které se každý prvek může v chromozomu posunout. Prvek se může posouvat za každý prvek, který není jeho potomkem. Zjištění této max shift hodnoty se dá implementovat průchodem polem a nalezení indexu prvního prvku, který následuje v poli za prvek a zároveň je v seznamu následníků (precedence).

Následně vygenerujeme náhodně index prvku, který můžeme posunout a hodnotu, o kterou tento prvek posuneme.

Inicializace

Problém je reprezentován AON grafem precedence jednotlivých tasků, který je acyklický. Nezávisle na délce trvání můžeme tudíž vygenerovat validní rozvrh pomocí BFS, ale existuje obecnější algoritmus vygenerování rozvrhu popsaný následujícím pseudokódem:

1. *$V = T_1$ // T_1 je první – dummy task*
2. *S = Empty*
3. *Opakuj dokud není splněna ukončující podmínka*
4. *přidej všechny následníky V do množiny L pro zpracování*
5. *přidej V na konec chromozomu*
6. *pokud je množina L prázdná skonči cyklus*
7. $V = q \in L(q), \ \forall n \in P(q)$, n už je naschedulovaný

// každý task, který je v AON grafu před q už je obsažen v //chromozomu

8. *S je inicializovaný rozvrh*

Závěr

V rámci této práce jsem nenalezl paralelní algoritmus pro nalezení optimálního rozvrhu RCPSP, ale navrhl jsem a naimplementoval zárodky paralelního genetického algoritmu pro minimalizující řešení RCPSP. Důvod pro toto je hlavně ten, že jsem neměl zkušenosti, znalosti z oblasti RCPSP, které byly nutné k paralelizaci – viz poznámka Oberhubera ohledně implementace paralelního algoritmu.

Reference:

Sprecher97 : Scheduling Resource-Constrained Projects Competition at Modest Memory Requirements [1997]

Selcuk Colak et al : Resource Constrained Project Scheduling: A Hybrid Neural Approach

Matthew Bartschi Wall : A Genetic Algorithm for Resource-Constrained Scheduling [1996]

Tvrdík[2009] : Paralelní algoritmy a systémy

Oberhuber[2010] : Paralelní algoritmy a architektury

Christian Artigues : The Resource-Constrained Project Scheduling Problem

Jiří Hrazdil [2008]: Genetické algoritmy

Petr Pošík : Paralelní genetické algoritmy