České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Paralelizace GRASP heuristiky na GPU

Praha, 2014 **Autor: Jakub Lukeš**

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jakub L u k e š

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Paralelizace GRASP heuristiky na GPU

Pokyny pro vypracování:

1. Seznamte se s algoritmem GRASP a s úlohou nasazování zdrojů elektrické energie, kterou má algoritmus řešit.
2. Navrhněte a diskutujte možnosti paralelizace algoritmu na grafické kartě GeForce GTX Titan.
3. Implementujte vybraným způsobem algoritmus pomocí CUDA nebo OpenCL.
4. Proveďte testy závislosti doby výpočtu na dimenzi úlohy (počtu zdrojů). Pokud to bude možné, porovnejte dobu výpočtu a kvalitu řešení s jinými způsoby řešení úlohy.

Seznam odborné literatury:

[1] Ana Viana, J. Pinho de Sousa, Manuel A. Matos: Fast solutions for UC problems by a new metaheuristic approach, Electric Power Systems Research, Volume 78, Issue 8, August 2008, Pages 1385-1395, ISSN 0378-7796, http://dx.doi.org/10.1016/j.epsr.2008.01.002.
[2] Ana Viana, J. Pinho de Sousa, Manuel A. Matos: Using GRASP to Solve the Unit Commitment Problem, Annals of Operations Research, Volume 120, 2003, Pages 117-132.

Vedoucí bakalářské práce: Ing. Michal Dvořák

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2014

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne _____ _____

Podpis autora práce

Poděkování

Chci poděkovat celé kanceláři Katedry řídící techniky sídlící v budově G v místnosti 202 na Karlově náměstí za vděčné připomínky, rady a materiály pro vypracování této bakalářské práce. Zvláště pak vedoucímu Ing. Michalu Dvořákovi a Ing. Janu Zábojníkovi za jejich neutuchající zájem o plody mé práce.

Dále děkuji svým rodičům a ostatním příbuzným za materiální a finanční podporu a příjemnou atmosféru pro psaní této bakalářské práce.

Abstrakt

Cílem této práce je paralelizovat metaheuristiku GRASP na grafické kartě GTX TITAN za pomoci jazyka CUDA C. Algoritmus slouží k řešení úlohy nasazování zdrojů elektrické energie. Jedná se o velice náročnou úlohu, kterou řeší společnosti zabývající se distribucí a přenosem elektrické energie. Zvolený problém byl úspěšně vyřešen a jsou prezentovány výsledky porovnávající cenu řešení s jinou literaturou. Dále je porovnána doba výpočtu algoritmu na procesoru a na grafické kartě. Výsledkem této práce je algoritmus fungující na grafické kartě i na procesoru. Pro nízký počet řešení je CPU rychlejší než GPU. Nalezená cena řešení se od referenčního liší na testovaném příkladě pouze o 0.01%.

Abstract

The goal of this bachelor thesis is the implementation of GRASP metaheuristic using the GPU GTX TITAN. The CUDA C programming language is used. The algorithm solves the Unit commitment problem which is in general very hard to solve and it is usualy solved by companies distributing and transmiting electrical power. The chosen problem was successfully solved and its results are compared to the results from the literature. The comparation of the GPU and the CPU implementations computing times are presented. The result of this bachelor thesis is the implentation of GRASP metaheuristic on the CPU and the GPU. For the lower size instances the CPU implementation is faster than the GPU implementation. The solution cost found by my GRASP implementation is about 0.01% worse than the best value found in literature.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Tématem této bakalářské práce je paralelizace GRASP heuristiky na GPU. K tomu, aby bylo vysvětleno, co přesně je GRASP a jakým způsobem pracuje, je vhodné se nejdříve zmínit o úloze, kterou řeší.

Úloha optimalizuje nasazování zdrojů elektrické energie a zjednodušeně lze říci, že její úkol spočívá v dodání požadovaného výkonu do elektrické sítě za co nejnižší cenu. K tomuto úkolu jsou dány k dispozici zdroje elektrické energie. Každý s odlišnými cenami produkce a různými omezeními na ni. Detailní popis úlohy je uveden v kapitole 2.

Algoritmus řešící tento problém musí rozhodnout, který zdroj bude v danou hodinu vyrábět výkon. Není obtížné nalézt nejlevnější zdroj, problém nastává v momentě, kdy je zdroj zapnut, protože zapnutý zdroj musí produkovat alespoň minimální výkon po určitou dobu. Může se tedy stát, že se v následující hodině sníží požadavky na výkon v síti a tento zdroj již nebude potřeba. Jeho provoz tedy zbytečně zvyšuje celkovou cenu řešení. Proto cesta, vybírat vždy nejlevnější zdroj, nemusí vést k nejlevnějšímu řešení. Úloha, ze své podstaty, obsahuje velké množství proměnných, které mohou nabývat hodnot 0 nebo 1. Například zadání, kde je úkolem rozvrhnout výrobu 10-ti zdrojů pro časový horizont 24 hodin, má celkem 240 proměnných. Existuje tedy $2^{240} \approx 1.77 \cdot 10^{72}$ možností, jak lze úlohu řešit. Ne každé z těchto možností je přípustným řešením úlohy. Optimálních řešení může být i více, není ovšem snadné je v tomto množství nalézt. Uvedené dva problémy (výběr zdroje, který bude zapnut, a velké množství proměnných) způsobují, že tuto úlohu nelze řešit jednoduchým způsobem, a proto je zajímavou oblastí studia.

Pro řešení popsané úlohy byl v této bakalářské práci vybrán algoritmus GRASP, který se skládá ze dvou částí. První část nalezne za pomoci řízené náhody libovolné přípustné řešení. Druhá část postupně náhodně vypíná zdroje v určitých hodinách a vyhodnocuje jejich dopad na kvalitu řešení. V některých případech vypnutí nevadí, například pokud

byl zdroj v danou hodinu zapnut zbytečně. V ostatních případech způsobí vypnutí zdroje např. nedodání požadovaného výkonu do sítě. Tento výkon je tedy nutné vyrobit za pomoci jiných zdrojů. Výběr takových zdrojů a to, jak přesně se dané řešení změní, obstarávají speciální postupy, tzv. opravy. Kompletní popis oprav lze nalézt v kapitole 3.

V kapitole 4 jsou popsána úskalí implementace algoritmu GRASP v jazyce CUDA C. Jsou tam také uvedeny změny, které byly provedeny a kterými se algoritmus GRASP liší od článku [3]. Zjednodušeně se dá říci, že se jedná o totožný algoritmus, pouze pokud byly v článku nejasné formulace a nebo nebyly vysvětleny kroky, pak byla navrhnuta vlastní metodika řešení. S ohledem na přehlednost nebude neustále uváděno, že daná formulace byla převzata z článku [3], pouze pokud by daná sekce nepocházela z článku, pak toto bude zmíněno.

V kapitole 5 jsou prezentovány výsledky při použití algoritmu GRASP na vzorovém příkladu. Tento příklad představuje benchmark v mnoha článcích zabývající se úlohou optimalizace zdrojů elektrické energie a bude také použit pro GRASP. Jako vzorové řešení bylo převzato řešení z článku [4]. Toto řešení bylo přepočteno a byly zjištěny nesrovnalosti v jeho ceně. Tyto problémy jsou v dané kapitole diskutovány.

Na úplný závěr je v příloze A zmíněna Lambda iterace, která se v algoritmu používá pro optimální rozvržení výkonu mezi zapnuté zdroje.

Kapitola 2

Popis problému

V této kapitole jsou definovány všechny pojmy, které jsou potřebné pro řešení úlohy. Předpokládejme, že je zadán určitý plánovací horizont. V každé hodině plánovacího horizontu je zadán požadovaný výkon a požadované rezervy. Požadovaný výkon je nutné vyrobit. Rezervy jsou výkon, který musí být možné vyrobit, aniž by došlo k zapnutí dalšího zdroje.

K dispozici je dán určitý počet zdrojů, pomocí kterých lze požadavky splnit. Každý tento zdroj má několik parametrů, které ho charakterizují, konkrétně:

- minimální a maximální výkon, který je schopen dodat do sítě,
- minimální dobu, po kterou musí zůstat po zapnutí v chodu (v textu označena jako minimální doba zapnutí)
- minimální dobu, po kterou musí po odstavení zůstat odstaven (v textu označena jako minimální doba vypnutí),
- počáteční stav, zda zdroj před plánovacím horizontem vyráběl výkon nebo ne a jak dlouho tento stav trval,
- palivové náklady v závislosti na dodaném výkonu,
- cenu za zapnutí, pokud v hodině přecházející neprodukoval výkon.

Je zde tedy řešena optimalizační úloha, kde úkolem je minimalizovat kriteriální funkci (sekce 2.1), která závisí na rozvržení požadovaného výkonu v dané hodině plánovacím horizontu mezi jednotlivé zdroje. Podstatné je, ve které hodině a kolik výkonu každý zdroj produkuje.

2.1 Kriteriální funkce

Kriteriální funkce počítá celkovou cenu řešení a úkolem je ji v této úloze minimalizovat. Zde je definována jako součet palivových nákladů(sekce 2.1) a cen za zapnutí zdrojů (sekce 2.1) v dané hodině. Přesně vyjadřuje cenu za vyrobení elektrické energie. Tedy se v každé hodině plánovacího horizontu hledají hodnoty výkonu jednotlivých zdrojů, při kterých je hodnota kriteriální funkce minimální.

Palivové náklady PN jsou v této úloze reprezentované kvadratickou funkcí. Reprezentují cenu výroby výkonu P_i daného zdroje i.

$$PN_i\left(P_i\right) = a_i P_i^2 + b_i P_i + c_i \cdot u, \tag{2.1}$$

kde

- PN_i je funkce palivových nákladů i-tého zdroje,
- P_i je produkovaný výkon, který je vždy nezáporný,
- a_i je parametr měřený v \$/MW^{2}h,
- b_i je parametr měřený v \$/MWh,
- c_i je parametr měřený v \$/h,
- u_i je proměnná, která je rovna 1 pokud zdroj i běží, 0 v ostatních případech.

Cena za zapnutí zdroje ZZ je reprezentovaná konstantami, které závisejí na stavu daného zdroje v předchozí hodině. Jsou zde použity dvě konstanty, jedna pro tzv. "teplý start", která je použita, pokud byl zdroj i vypnut po dobu menší nebo rovnou určité hodnotě, a druhá pro tzv. "studený start", pokud by byl zdroj i vypnut po dobu delší než určitá hodna. Cena za zapnutí zdroje, který v hodině $t-1$ produkoval výkon, je nulová, protože se nejedná o zapnutí.

$$ZZ(i,t) = \begin{cases} \text{cena horkého startu}, & \text{doba vypnutí} \leq \text{počet h. do studeného startu} \\ \text{cena studeného startu}, & \text{doba vypnutí} > \text{počet h. do studeného startu} \\ 0, & \text{byl zapnutý v hodině } t-1 \end{cases}$$

2.2 Omezující podmínky

Jedná se o požadavky, které komplikují hledání optimálního řešení a definují, které vlastnosti musí mít. V tomto textu budeme uvažovat následující:

- Požadovaný výkon v síti - součet produkce výkonu jednotlivých zdrojů musí být roven této hodnotě.
- Požadované rezervy - výkon, který zdroje nevyrábějí, ale musí být možnost ho nějakým způsobem vyrobit, aniž by musel být zapnut další zdroj.
- Minimální doba vypnutí a minimální doba zapnutí - pokud byl zdroj v daném kroku vypnut, musí zůstat vypnut minimálně po dobu T^{off} a pokud byl zdroj v daném kroku zapnut, musí zůstat zapnut minimálně po dobu T^{on}.

Řešení, které splňuje všechny uvedené omezující podmínky se označuje jako přípustné.

2.3 Přechodný bod

Jedná se o takovou hodinu v plánovacím horizontu, kdy zdroj mění svůj stav z vypnutý na zapnutý, nebo zapnutý na vypnutý. Přechodný bod je zvlášť důležitý, pokud má být řešení nějakým způsobem upraveno, protože jeho vznik je úzce spojen se splněním požadovaného výkonu nebo požadované rezervy. Je rozlišováno, zda se hledá přechodný bod směrem doleva, směrem do minulosti, nebo doprava, směrem do budoucnosti.

Hledání přechodného bodu se provádí postupně z hodiny t. Nadále bude předpokládáno, že zdroj v hodině t je zapnut, protože přechodné body se v této bakalářské práci hledají pouze pro zdroje zapnuté v hodině t. Zdroj může mít obecně mnoho levých a pravým přechodných bodů, proto je vždy levý a pravý přechodný bod vztažen ke konkrétní hodině t. Existuje nejvýše jeden levý (pravý) přechodný bod pro konkrétní hodinu t.

Levý přechodný bod LPB je hodina v plánovacím horizontu, ve které zdroj najel a produkoval výkon až do hodiny t včetně. Tento bod může existovat, ale také nemusí. Počáteční stav daného zdroje je brán v úvahu. Situace na začátku plánovacího horizontu je trochu specifická, přibližme si ji tedy. Předpokládejme, že zdroj produkuje výkon od první hodiny až do hodiny t:

- Pokud zdroj před plánovacím horizontem neprodukoval výkon, pak zdroj má v první hodině levý přechodný bod při hledání z hodiny t.

- Pokud zdroj před plánovacím horizontem produkoval výkon, pak zdroj nemá levý přechodný bod při hledání z hodiny t.

Pravý přechodný bod PPB je hodina v plánovacím horizontu, kdy je zdroj zapnut, ale v hodině následující vypnut. Tato logika odporuje definici přechodného bodu (sekce 2.3), ale umožňuje stručnější zápis. Dle definice by byl přechodný bod až v následující hodině. Algoritmus se ale snaží v přechodných bodech, příslušící konkrétní hodině t, zdroje vypínat. V hodině, kdy zdroj je vypnut, ho nelze vypnut, proto byla jeho definice upravena. Díky této úpravě může mít zdroj levý a pravý přechodný bod ve stejnou hodinu a lze vždy mluvit o vypnutí v přechodném bodě. Co se týká konce plánovacího horizontu, tam je situace trošku specifičtější, a je vhodné se nad ní zamyslet:

- Pokud zdroj produkoval výkon v hodině t a produkoval ho nepřetržitě až do konce plánovacího horizontu, včetně poslední hodiny, pak hodina t nemá žádný pravý přechodný bod.

Pokud bude v tomto textu hovořeno pouze o přechodném bodě, pak se tím myslí libovolný, ať už pravý nebo levý, přechodný bod získaný postupně z hodiny t. Je to z důvodu toho, aby se stále nemusela vypisovat tato fráze. Tabulky 2.1 a 2.2 předvádí výpočet přechodných bodů.

Tabulka 2.1: Vysvětlení problematiky přechodných bodů. Výpočet levého přechodného bodu LPB (sekce 2.3) a pravého přechodného bodu PPB v závislosti na tom, kdy se daný zdroj zapne. Předpokládáme, že hledáme LPB a PPB ze **třetí** hodiny, která je zvýrazněna tučně.

		Poč.	Hodiny					LPB	PPB
		stav	1	2	**3**	4	5		
Zdroje	1	-5	0	1	**1**	1	0	2	4
	2	5	1	1	**1**	1	0	-1	4
	3	2	0	0	**1**	1	1	3	-1
	4	-2	1	1	**1**	1	1	1	-1
	5	-1	0	0	**1**	0	0	3	3
	6	-5	1	0	**1**	0	1	3	3

Tabulka 2.2: Vysvětlení problematiky přechodných bodů. Výpočet levého přechodného bodu LPB (sekce 2.3) a pravého přechodného bodu PPB v závislosti na tom, kdy se daný zdroj zapne. Předpokládáme, že hledáme LPB a PPB ze **čtvrté** hodiny, která je zvýrazněna tučně. Znak X značí, že pro daný zdroj přechodné body z hodiny t neurčujeme.

		Poč.	Hodiny					LPB	PPB
		stav	1	2	3	**4**	5		
Zdroje	1	-5	0	1	1	**1**	0	2	4
	2	5	1	1	1	**1**	0	-1	4
	3	2	0	0	1	**1**	1	3	-1
	4	-2	1	1	1	**1**	1	1	-1
	5	-1	0	0	1	**0**	0	X	X
	6	-5	1	0	1	**0**	1	X	X

Kapitola 3

GRASP

Při aplikaci klasických metod (např. dynamické programování nebo metoda větví a mezí) na střední nebo větší úlohy, se ukazuje, že tyto metody nejsou schopny dodat přípustné řešení v dostatečně krátkém čase. Proto se výzkum zaobíral myšlenkou najít nějaké přípustné řešení, ne nutně optimální, pro reálné problémy. Bylo použito několik heuristických přístupů vycházejících z klasických metod, např. metoda větví a mezí, případně metody používající prioritní seznamy a Lagrangeovu relaxaci, které se staly základem pro optimalizaci v průmyslu.

Po objevení metaheuristiky a evolučních algoritmů v moderní optimalizaci, se objevily další metody např. simulované žíhání, tabu prohledávání, GRASP nebo genetické algoritmy. Tyto nové metody byly použity k řešení problému nasazení zdrojů elektrické energie a obecně vykazovaly lepší výsledky než tradiční přístup. Bohužel jejich úspěšné využití v praxi brání neochota používat metody, jejichž výkon silně závisí na správném nastavení parametrů. Proces nastavování není systematický a pokud se provede špatně, může vést k výsledkům nízké kvality. [3].

GRASP (z anglického "Greedy Randomised Adaptive Search Procedure") je iterační algoritmus přistupující k úloze nasazování zdrojů elektrické energie využitím náhody. Vliv náhody řídí parametr α, jehož správná hodnota se liší příklad od příkladu. Jak bylo řečeno výše, nesprávné nastavení parametru může vést k výsledkům velmi vzdálených od optimálního a proto má GRASP dvě části. První část je konstrukční, kde je vytvořeno přípustné řešení. Druhá část je prohledávací, kde je řešení upravováno ve snaze nahradit drahé zdroje levnějšími, tedy snížit vliv špatného nastavení α na konečné řešení. Tato dvojice konstrukce-prohledávání se opakuje po pevně daný počet kroků, zde označeno jako počet řešení.

V úvodu bylo naznačeno, že vybírat vždy nejlevnější zdroj nemusí vést k optimálnímu

řešení. V některých případech ho nelze postupně skoro ani získat, protože kroky k němu vedoucí jsou často nelogické, např. v polovině plánovacího horizontu zapnout druhý nejdražší zdroj. Logika těchto kroků je zřejmá až teprve v momentě, kdy je spočtena cena řešení a je nižší než cena ostatních. GRASP tyto, z počátku nelogické, kroky nechává právě na náhodě a na parametru α.

Předtím, než bude vysvětlen přesný vliv parametru, je vhodné nadefinovat hladovou funkci.

Hladová funkce HF ohodnotí příspěvek daného zdroje ke kriteriální funkci(sekce 2.1), pokud by tento zdroj byl zvolen. Zjišťuje aktuální cenu jednoho MW daného zdroje. Díky ceně za zapnutí zdroje ZZ (sekce 2.1) bere v úvahu dopad předchozích rozhodnutí.

$$HF\left(i,t\right) = \frac{PN_i\left(P^i_{\max}\right) + ZZ\left(i,t\right)}{P^i_{\max}} \tag{3.1}$$

kde funkce $PN_i(P_i)$ je funkce palivových nákladů (sekce 2.1), $ZZ\left(i,t\right)$ je cena za zapnutí (sekce 2.1) a $P^i_{\max}$ je maximální povolený výkon i-tého zdroje.

Hlavní faktor pro výběr daného zdroje do omezeného seznamu zdrojů je jeho hodnota hladové funkce v dané hodině. Vliv parametru α je definován tak, aby nemohl(y) být zvolen(y) nejdražší zdroj(e), pokud to není specificky požadováno jeho nastavením na hodnotu 1.

$$HF_{\min}\left(t\right) \leq HF\left(i,t\right) \leq HF_{\min}\left(t\right) + \alpha\left[HF_{\max}\left(t\right) - HF_{\min}\left(t\right)\right] \tag{3.2}$$

kde význam jednotlivých symbolů je následující

- $\alpha \in [0,1]$ je parametr GRASPu,
- $HF\left(i,t\right)$ je výsledek hladové funkce pro i-tý zdroj v t-té hodině,
- $HF_{\min}\left(t\right) = min(\{HF\left(i,t\right)|i \in \{1,2,\cdots,n\}\})$,
- $HF_{\max}\left(t\right) = max(\{HF\left(i,t\right)|i \in \{1,2,\cdots,n\}\})$.

Následující příklad ilustruje vliv parametru α na vybírání dalšího zdroje. Předpokládejme, že se nacházíme ve 4-té hodině plánovacího horizontu a je třeba vybrat další zdroj, který bude zapnut, kvůli splnění podmínky požadovaného výkonu. Tabulka 3.1 ukazuje tuto situaci. Nejprve jsou zapnuty všechny zdroje, které kvůli minimální době běhu musí produkovat výkon i ve čtvrté hodině. V tomto případě jsou to např. zdroje

1 a 2, tabulka 3.2. Následně je spočtena hodnota hladové funkce pro každý zbylý zdroj (tabulka 3.4). Pro první případ předpokládejme $\alpha = 0.5$. Meze pro výběr zdrojů jsou

$$10 \leq HF\,(i,t) \leq 10 + \alpha \cdot [25 - 10] = 10 + 0.5 \cdot 15 = 17.5$$

tedy zdroje, které lze vybrat jsou 4 a 5. Tabulka 3.3 ukazuje situaci po vybrání dvou zdrojů a bylo rozhodnuto, že toto řešení je již přípustné. Pro demonstraci ukažme ten samý příklad pro $\alpha = 1$. Získáváme meze

$$10 \leq HF\,(i,t) \leq 10 + \alpha \cdot [25 - 10] = 10 + 1 \cdot 15 = 25$$

tedy lze vybrat libovolný zdroj. Náhodně byl vybrán zdroj zdroj 4 a 7, tabulka 3.5 zobrazuje toto řešení. V případě, že je vybrán zdroj č.3, pak nastává problém. Jeho hodnota hladové funkce je záporná, tedy zdroj měl zůstat vypnutý. Aby tato podmínka nebyla porušena, tak zdroj je zapnut od posledního momentu vypnutí. Situace je znázorněna v tabulce 3.6.

Z rovnice 3.2 plyne důležitá vlastnost parametru α. Pro $\alpha = 0$ je vždy zapnut nejlevnější zdroj a tato situace může vést k situaci, kde nemusí být možné nalézt optimální řešení z výše uvedených důvodů. Pro $\alpha = 1$ může být vybrán libovolný zdroj, např. nejdražší. Tato volba nejspíš také uzavře možnost nalézt optimální řešení a pokud ne, tak ho bude možná velmi těžké získat. Nejspíše bude muset být nahrazeno velké množství drahých zdrojů levnějšími. Optimální hodnota parametru je někde v intervalu $[0, 1]$ a musí se postupně hledat.

GRASP na hledání optimálního řešení používá tedy dvojici konstrukce-prohledávání. V klasickém případě jsou tyto dvojice spouštěny postupně, zde je využito paralelních výpočtu na grafické kartě GeForce GTX Titan. Všechny výpočty dvojic probíhají současně.

Poprvé byl tento algoritmus představen v literatuře [1], následně vylepšen a byly přidány pravidla pro vypínání zdrojů, které produkují výkon celý plánovací horizont(literatura [2]). V této práci zejména využito poznatků a postupů ve zdroji [3].

Tabulka 3.1: Ukázka vlivu parametru α na výběr zdroje. Situace ve čtvrté hodině. Otazníky značí neznámé hodnoty.

		Hodiny			
		1	2	3	4
Zdroje	1	1	1	1	?
	2	1	1	1	?
	3	1	0	0	?
	4	0	1	1	?
	5	0	0	0	?
	6	0	0	0	?
	7	0	0	0	?

Tabulka 3.2: Ukázka vlivu parametru α na výběr zdroje. Situace ve čtvrté hodině. Zapnutí zdrojů, které musely běžet.

		Hodiny			
		1	2	3	4
Zdroje	1	1	1	1	1
	2	1	1	1	1
	3	1	0	0	?
	4	0	1	1	?
	5	0	0	0	?
	6	0	0	0	?
	7	0	0	0	?

Tabulka 3.3: Řešení získané při volbě $\alpha = 0.5$. Předpokládáme, že v dané hodině je řešení přípustné po zapnutí vybraných zdrojů 4 a 5. Zbylé zdroje jsou nastaveny na 0.

		Hodiny			
		1	2	3	4
Zdroje	1	1	1	1	1
	2	1	1	1	1
	3	1	0	0	0
	4	0	1	1	1
	5	0	0	0	1
	6	0	0	0	0
	7	0	0	0	0

Tabulka 3.4: Hodnota hladové funkce HF (sekce 3). Pro zdroj, který musí běžet je 0. Pokud zdroj musí zůstat vypnutý, pak je jeho hladová funkce vynásobena -1. Použije se pouze v případě, že by ostatní zdroje nestačily na pokrytí požadavků a nebo pokud parametr α dosáhl hodnoty 1.

		HF
Zdroje	1	0
	2	0
	3	-5
	4	10
	5	15
	6	20
	7	25

Tabulka 3.5: Řešení získané při volbě $\alpha = 1$. Situace při náhodné volbě zdrojů, které měly kladnou hodnotu hladové funkce. Předpokládáme, že v dané hodině jsou splněny omezující podmínky po zapnutí vybraných zdrojů 4 a 7. Zbylé jsou nastaveny na 0.

		Hodiny			
		1	2	3	4
Zdroje	1	1	1	1	1
	2	1	1	1	1
	3	1	0	0	0
	4	0	1	1	1
	5	0	0	0	0
	6	0	0	0	0
	7	0	0	0	1

Tabulka 3.6: Řešení získané při volbě $\alpha = 1$. Situace při náhodné volbě třetího zdroje, který měl zápornou hodnotu hladové funkce. Předpokládáme, že v dané hodině jsou splněny omezující podmínky po zapnutí vybraných zdrojů 3 a 5. Zbylé jsou nastaveny na 0.

		Hodiny			
		1	2	3	4
Zdroje	1	1	1	1	1
	2	1	1	1	1
	3	1	1	1	1
	4	0	1	1	0
	5	0	0	0	1
	6	0	0	0	0
	7	0	0	0	0

3.1 Konstrukční fáze

Konstrukční fáze je zde od toho, aby nalezla nějaké přípustné řešení. Řešení je vytvářeno postupně v každé hodině plánovacího horizontu. Nelze říci, že by algoritmus o něčem rozhodoval, pouze dává seznam vhodných kandidátů, ze kterého náhoda jednoho vybere. Tímto je zaručen zisk různých řešení, pokud se algoritmus někdy v budoucnu dostane do té samé situace.

Seznam vhodných kandidátů je silně ovlivněn parametrem α a je to jediný moment, kdy se tento parametr projevuje. Dodržuje se rovnice 3.2 a pokaždé, když náhoda vybere zdroj ze seznamu, tak se hodnota parametru o určitou hodnotu zvýší až do maximální hodnoty 1. Při sestavování nového seznamu v následující hodině je α nastavena na výchozí hodnotu.

To zda se zdroj dostane do seznamu vhodných kandidátů, záleží jednak na uvedené rovnici 3.2 a jednak na jeho minimální době běhu a minimální době vypnutí. Pokud zdroj vytváří výkon po dobu kratší, než je jeho minimální doba běhu, pak je automaticky v další hodině zapnut, do seznamu se nedostane. Pokud zdroj neprodukuje výkon po dobu kratší, než je jeho minimální doba vypnutí, pak do seznamu se dostane až teprve v momentě, kdy byly zapnuty všechny zdroje, jejichž zapnutí toto omezení neporuší a nebo hodnota parametru α je rovna 1.

Samozřejmě má smysl vytvářet tento seznam jenom do doby, dokud dané řešení není přípustné z hlediska splnění požadovaného výkonu. To nastává tehdy, pokud maximální teoretický výkon zapnutých zdrojů v dané hodině je alespoň roven součtu požadovaného výkonu a rezerv v dané hodině. V tento moment lze mluvit o teoretickém výkonu, protože až následná Lambda iterace, příloha A, rozhodne, kolik výkonu bude každý zdroj produkovat.

3.1.1 Průběh konstrukční fáze

V každé hodině plánovacího horizontu probíhá algoritmus následovně:

1) Jsou zapnuty všechny zdroje, které musí být zapnuty, kvůli minimální době běhu.

2) Je spočtena hladová funkce 3 pro každý zdroj, který v danou hodinu není zapnutý.

3) Je sečten požadovaný výkon s rezervami pro danou hodinu.

4) Je získán maximální teoretický výkon, který jsou schopny vyprodukovat v danou hodinu zapnuté zdroje.

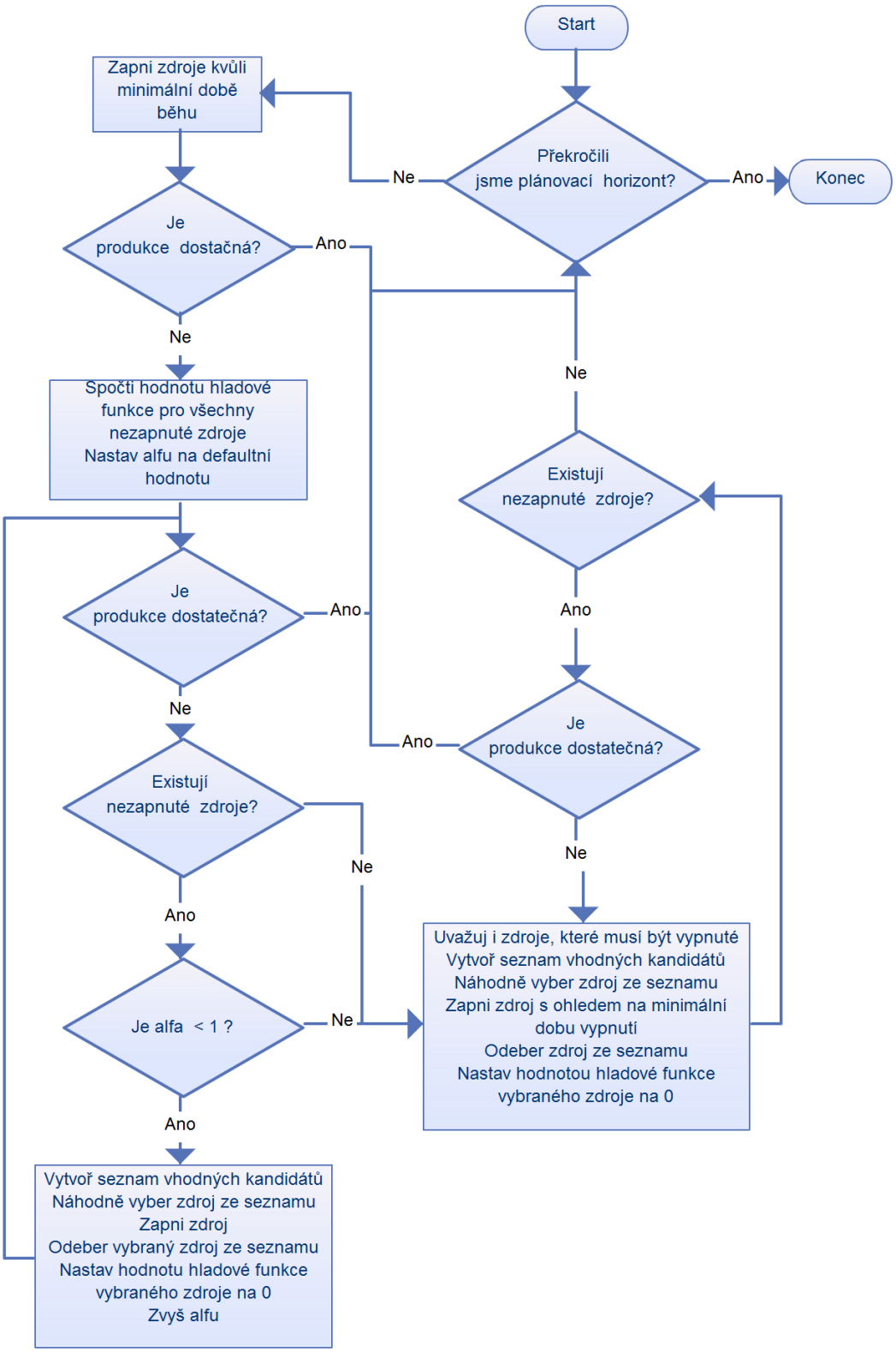
5) Je spočten chybějící výkon jako rozdíl mezi požadovaným a teoretickým.

6) Dokud není teoreticky vyroben chybějící výkon, a zároveň existují zdroje, které lze zapnout, a zároveň $\alpha < 1$:

   - Je vytvořen seznam vhodných kandidátů v závislosti na parametru α, které uvažuje jenom zdroje, které mají nezápornou hodnotu hladové funkce.
   - Je náhodně vybrán zdroj ze seznamu vhodných kandidátů, který je následně zapnut, jeho hodnota hladové funkce nastavena na 0 a seznam je zrušen.
   - Parametr α je inkrementován.

7) Dokud není teoreticky vyroben chybějící výkon, a zároveň existují zdroje, které lze zapnout, i za cenu toho, že by musely být zapnuty od posledního momentu vypnutí:

   - Je vytvořen seznam vhodných kandidátů v závislosti na parametru α, který bere absolutní hodnotu hladové funkce všech zdrojů. Tedy i zdroje se zápornou hodnotou hladové funkce, což jsou zdroje, které je nutné zapnout od poslední hodiny vypnutí, protože jejich zapnutí způsobí porušení minimální doby vypnutí.
   - Je náhodně vybrán zdroj ze seznamu vhodných kandidátů, který je následně zapnut, jeho hodnota hladové funkce nastavena na 0 a seznam je zrušen.



Obrázek 3.1: Schéma průběhu konstrukční části algoritmu GRASP.

3.2 Prohledávací fáze

Cílem této části je vylepšit (zlevnit) řešení, které jí poskytne konstrukční fáze. Při jeho vytváření je používáno velké množství rozhodnutí, které byly ovlivněny náhodou. Některá rozhodnutí otevřela cestu k optimálním řešení, jiná ji naopak zavřela. Není možné rozlišit, které je které, proto je toto řešení upravováno opět za pomoci náhody po určitý počet iterací.

V dané iteraci je nejdříve náhodně vybrána hodina z plánovacího horizontu. Ze všech zdrojů, které ve vybrané hodině produkují výkon jsou do seznamu zdrojů vybrány ty zdroje, které produkují v přechodném bodě (sekce 2.3) na minimálním výkonu.

Pokud žádné takové zdroje neexistují, pak se vytvoří seznam zdrojů, které v přechodném bodě neprodukují maximální výkon. V případě, že neexistují ani takové zdroje, pak se vytvoří seznam zdrojů, které běží celý plánovací horizont. Jejich problematika bude řešena ke konci kapitoly 3.5, protože je rozsáhlejší a tedy ji prozatím budeme ignorovat. Pokud i seznam zdrojů běžící celý plánovací horizont je prázdný, pak se iterace se opakuje. na

Úkolem je snížit cenu řešení, tedy nemá smysl vypínat zdroje mimo přechodné body. Kdyby byly zdroje vypínány mimo přechodné body, tak by vznikaly nové nájezdy a to by způsobilo velké změny v řešení. Toto rozhodně není úkolem této části.

V momentě, kdy je zdroj v dané hodině vypnut, mohou nastat dvě situace. S přípustností řešení se nic nestane, zdroj tedy běžel zbytečně. Druhá situace je zajímavější. Nutně je porušena jedna z podmínek přípustnosti řešení a je vhodné analyzovat která. V tomto textu se předpokládají celkem tři. Případná implementace nových podmínek by nezpůsobila velké problémy. Změnila by se jenom určitá část algoritmu, zbytek zůstane nezměněn. Toto dodatečné rozšíření patří mezi výhody algoritmu GRASP.

Pro každou podmínku je definována speciální oprava, která se aplikuje, pokud je tato podmínka porušena. Pokud je porušena více jak jedna podmínka, pak je aplikovaná obecná oprava, neboť obnovit přípustnost řešení je v tento moment obtížnější. Podmínky aplikace jednotlivých oprav jsou shrnuty v tabulce 3.7.

Pokud nedošlo při aplikaci opravy k nějakému problém, např. vypne se nejlevnější zdroj, který není čím nahradit, pak se řešení se přijme, i kdyby cena tohoto řešení byla vyšší než původního. Předpokládá se, že cena řešení může začít klesat až po několika iteracích, protože nahrazujeme drahé zdroje levnějšími.

3.2.1 Průběh prohledávací fáze

Celý algoritmus tedy probíhá následovně:

1) Je vytvořena záloha daného řešení.

2) Je náhodně vybrána hodina z plánovacího horizontu.

3) Je vytvořen seznam zdrojů, které v přechodném bodě (sekce 2.3) produkují minimální výkon. Pokud žádné takové zdroje neexistují, pak se vytvoří seznam zdrojů, které neprodukují na maximálním výkonu. Pokud je opět prázdný, iterace skončila.

4) Je náhodně vybrán zdroj ze seznamu zdrojů.

5) Je náhodně vybrán levý nebo pravý přechodný bod.

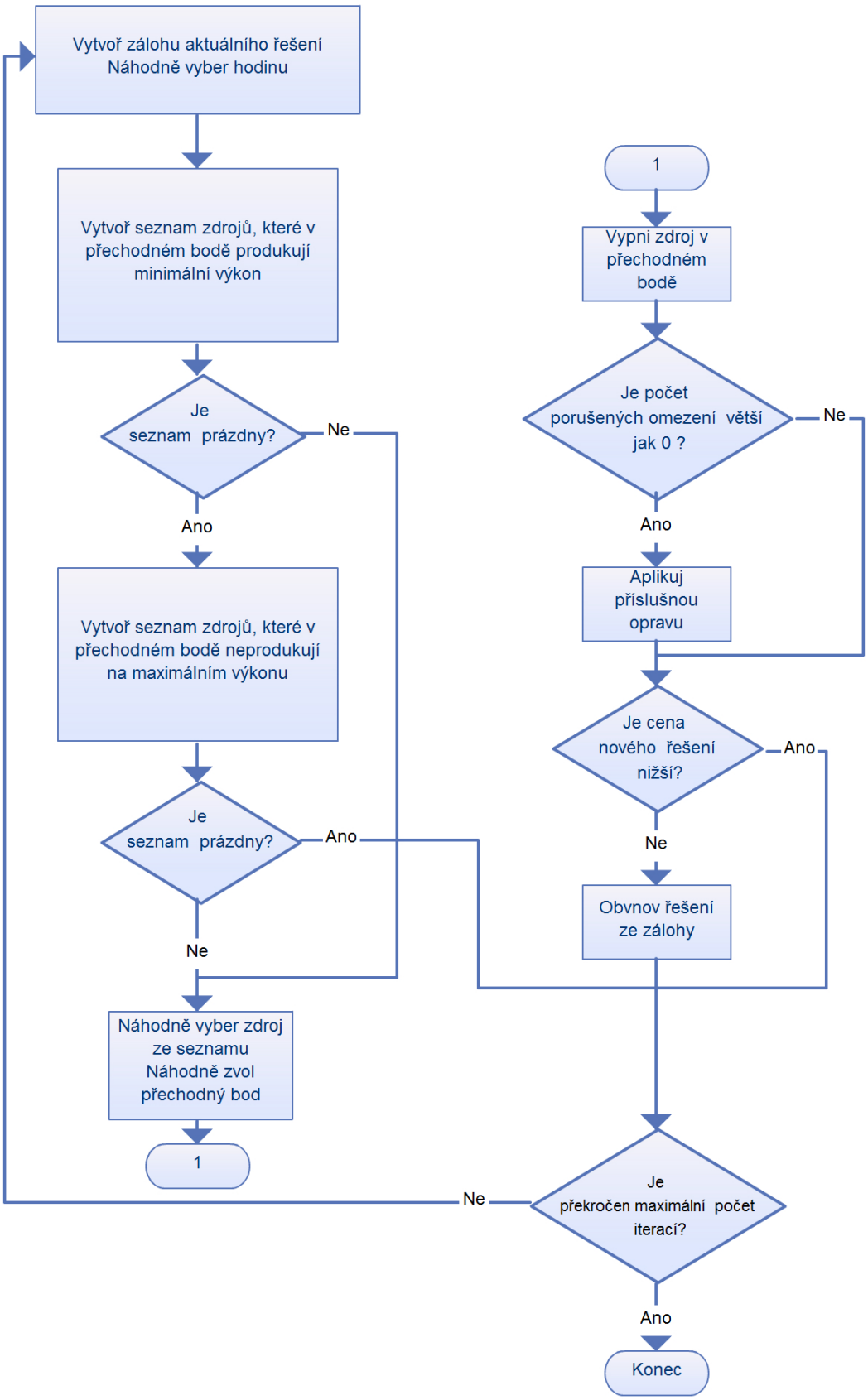
6) Zdroj je vypnut ve vybraném přechodném bodě.

7) Zkontroluje se porušení omezení a případně se aplikují opravy dle tabulky 3.7.

8) Pokud byla oprava neúspěšná, pak je toto řešení zahozeno a je obnoven stav před vypnutím. V opačném případě se toto řešení bere jako výchozí pro další iterace.

Kvůli přehlednosti algoritmu nebylo uvedeno, kdy přesně se aplikují pravidla pro vypínání zdrojů běžící celý plánovací horizont(sekce 3.5). Konkrétní situace, kdy se tyto podmínky aplikují jsou:

- Seznam zdrojů prázdný. Pak se náhodně vybere zdroj a řeší se zda, pro něj platí podmínky níže.
- Je vybrán zdroj ze seznamu zdrojů, které splňuje podmínky.



Obrázek 3.2: Schéma průběhu prohledávací části algoritmu GRASP.

Tabulka 3.7: Přehled použití oprav v závislosti, která podmínka přípustnosti řešení je splněna. Pro zjednodušení se dá říci, že pokud je porušena více než jedna podmínka, pak je provedena obecná oprava.

Splnění podmínky				
	Výkon	Rezervy	Doba běhu	Aplikovaný typ opravy
	Ano	Ano	Ano	Žádná
	Ano	Ano	Ne	Oprava minimální doby běhu
	Ano	Ne	Ano	Oprava požadovaných rezerv
	Ano	Ne	Ne	Obecná oprava
	Ne	Ano	Ano	Oprava požadovaného výkonu
	Ne	Ano	Ne	Obecná oprava
	Ne	Ne	Ano	Obecná oprava
	Ne	Ne	Ne	Obecná oprava

3.3 Zapínání vypnutého zdroje

Zapnout zdroj, který v dané hodině neprodukoval žádný výkon není jednoduché. Situaci zvláště komplikuje minimální doba zapnutí a minimální doba vypnutí. Zde je prezentován způsob jak tuto situaci řešit. Protože z žádném ve zdrojích nebyl popsán, byl mnou vymyšlen.

Předpokládejme, že je nutné zapnout zdroj v hodině t a zjistit, jak moc do minulosti případně do budoucnosti se musí daný zdroj zapnout také, aby byly splněny požadavky na minimální dobu zapnutí a minimální dobu vypnutí. Zapnout zde budeme rozumět ve smyslu produkovat alespoň minimální výkon.

1) Předpokládejme, že hodina t je první hodina plánovacího horizontu. Pak nelze zapínat zdroj směrem do minulosti. Tabulka 3.8 ukazuje, jak jsou jednotlivé zdroje řešeny. U dané situace je v závorce uveden zdroj z této tabulky.

   a) Zdroj před plánovacím horizontem vyráběl výkon po dobu T.

      - Pokud je zdroj v druhé hodině zapnut, pak je vše v pořádku a zdroj stačí zapnout pouze v první hodině (zdroj 1).
      - Pokud je zdroj v druhé hodině vypnut, pak záleží jak dlouho.
        - Pokud je tato hodnota menší než minimální doba vypnutí, zdroj tedy byl vypnut po minimální dobu vypnutí, pak je nutné zdroj zapnout od druhé hodiny až do doby dalšího zapnutí (zdroj 2).

  - Pokud je tato hodnota větší než minimální doba vypnutí, pak stačí zdroj zapnout jenom v první hodině (zdroj 3).

b) Zdroj před plánovacím horizontem nevyráběl výkon po dobu T.

- Pokud T je menší než minimální doba vypnutí daného zdroje, pak zdroj nelze zapnout. Výpočet se zastaví, řešení se zamítne a obnoví předchozí ze zálohy (zdroj 4).
- Pokud T je větší nebo rovno než minimální doba vypnutí daného zdroje, pak to vypadá, že zdroj lze zapnout. Ještě je třeba ale zkontrolovat stav ve druhé hodině.
  - Pokud je zdroj v druhé hodině zapnut, pak je vše v pořádku a zdroj stačí zapnout pouze v první hodině (zdroj 5).
  - Pokud je zdroj v druhé hodině vypnut, pak záleží jak dlouho.
    * Pokud je tato hodnota menší než minimální doba vypnutí + minimální doba zapnutí, zdroj tedy byl vypnut po minimální dobu vypnutí, pak je nutné zdroj také zapnout od druhé hodiny až do doby dalšího zapnutí (zdroj 6).
    * Pokud je tato hodnota větší než minimální doba zapnutí + minimální doba vypnutí, pak stačí zdroj zapnout po minimální dobu zapnutí (zdroj 7). Pokud ne, zapneme zdroj po minimální dobu zapnutí + minimální dobu vypnutí (zdroj 8).

Tabulka 3.8: Tabulka ukazuje jednotlivé situace, které mohou nastat při zapínání zdroje na počátku plánovacího horizontu. Tabulka hned pod ní ukazuje, zda v dané situaci lze zapnout a pokud ano, jak. Předpokládáme, že minimální doba běhu zdroje je rovna 2 hodinám a minimální doba vypnutí zdroje je 2. Otazník značí nepodstatnou hodnotu.

			Hodiny						
		Poč. stav	**1**	2	3	4	5	...	n
Zdroj	1	2	**0**	1	?	?	?	...	?
	2	2	**0**	0	1	1	?	...	?
	3	2	**0**	0	0	1	?	...	?
	4	-1	**0**	1	?	?	?	...	?
	5	-2	**0**	1	1	1	?	...	?
	6	-2	**0**	0	1	1	?	...	?
	7	-2	**0**	0	0	0	1	...	?
	8	-2	**0**	0	0	1	1	...	?

			Hodiny						
		Poč. stav	**1**	2	3	4	5	...	n
Zdroj	1	2	**1**	1	?	?	?	...	?
	2	2	**1**	1	1	1	?	...	?
	3	2	**1**	0	0	1	?	...	?
	4	-1	**0**	1	?	?	?	...	?
	5	-2	**1**	1	1	1	?	...	?
	6	-2	**1**	1	1	1	?	...	?
	7	-2	**1**	1	0	0	1	...	?
	8	-2	**1**	1	1	1	1	...	?

2) Předpokládejme, že hodina t je poslední hodina plánovacího horizontu. Pak nelze zapínat zdroj směrem do budoucnosti. Tabulka 3.9 ukazuje, jak jsou jednotlivé zdroje řešeny. U dané situace je v závorce uveden zdroj z této tabulky.

   - Pokud je zdroj v předposlední hodině zapnut, pak je vše v pořádku a zdroj stačí zapnout pouze v poslední hodině (zdroj 1).
   - Pokud je zdroj v předposlední hodině vypnut, pak záleží jak dlouho.
     - Pokud je tato hodnota menší než minimální doba vypnutí, zdroj tedy byl vypnut po minimální dobu vypnutí, pak je nutné zdroj zapnout od poslední hodiny až do doby předchozího vypnutí (zdroj 2).
     - Pokud je tato hodnota větší nebo rovna minimální době vypnutí, pak stačí zdroj zapnout pouze v poslední hodině (zdroj 3).

Tabulka 3.9: Tabulka ukazuje jednotlivé situace, které mohou nastat při zapínání zdroje na konci plánovacího horizontu. Tabulka hned pod ní ukazuje, zda v dané situaci lze zapnout a pokud ano, jak. Předpokládáme, že minimální doba běhu zdroje je rovna 2 hodinám a minimální doba vypnutí zdroje je 2. Otazník značí nepodstatnou hodnotu.

		Hodiny						
		1	2	...	7	8	9	**10**
Zdroj	1	1	1	...	?	?	1	**0**
	2	1	1	...	?	1	0	**0**
	3	1	0	...	1	0	0	**0**

		Hodiny						
		1	2	...	7	8	9	**10**
Zdroj	1	1	1	...	?	?	1	**1**
	2	1	1	...	?	1	1	**1**
	3	1	0	...	1	0	0	**1**

3) Předpokládejme, že hodina t je hodina, která není ani první ani poslední hodinou plánovacího horizontu. Tabulka 3.10 ukazuje, jak jsou jednotlivé zdroje řešeny. U dané situace je v závorce uveden zdroj z této tabulky. Pro tabulku bude preferovaný směr doprava, do budoucnosti.

   - Pokud je zdroj v hodině $t-1$ a $t+1$ zapnut, pak není nutné provádět další úpravy (zdroj 1).
   - Pokud je zdroj v hodině $t-1$ zapnut, ale v hodině $t+1$ vypnut, pak je nutné zjistit, jak dlouho je vypnut.
     - Pokud je tato hodnota menší než minimální doba vypnutí, pak je nutné zdroj zapnout od hodiny $t+1$ hodiny až do dalšího zapnutí(zdroj 2).
     - Pokud je tato hodnota větší nebo rovna minimální době vypnutí, pak stačí zdroj zapnout jenom v hodině t (zdroj 3).
   - Pokud je zdroj v hodině $t-1$ vypnut, ale v hodině $t+1$ zapnut, pak je nutné zjistit, jak dlouho je vypnut.
     - Pokud je tato hodnota menší než minimální doba vypnutí, pak je nutné zdroj zapnout od hodiny $t-1$ hodiny až do dalšího zapnutí (zdroj 4).
     - Pokud je tato hodnota větší nebo rovna minimální době vypnutí, pak stačí zdroj zapnout jenom v hodině t (zdroj 5).
   - Pokud je zdroj v hodině $t-1$ a $t+1$ vypnut, pak je třeba se ho pokusit zapnout ve směru oprav. Tímto se myslí, že pokud byl původní zdroj vypnut v levém přechodném bodě, pak se pokusíme všechny zdroje zapínat směrem do budoucnosti, je-li to možné. Pokud byl zdroj vypnut v pravém přechodném bodě, je preferovaný směr zapínání zdrojů směrem do minulosti. V tento moment je jedno, kterým směrem zapínáme. Označme preferovaný směr A a jeho dobu trvání vypnutí A_t. Druhý směr je B a jeho doba trvání vypnutí B_t.
     - Nejprve se zkontroluje, zda B_t je větší nebo rovna minimální době vypnutí.
     - Pokud ano, pak lze zapnout pouze směrem A. Zkontroluje se, zda A_t je větší než minimální doba zapnutí -1 + minimální doba vypnutí.
       - Pokud ano, lze zapnout jenom ve směru A a to po dobu minimální doba zapnutí -1 (zdroj 6).
       - Pokud ne, je nutné zdroj zapnout ve směru A na dobu A_t (zdroj 7).
     - Pokud ne, pak nelze zapnout jenom směrem A. Je ale možné, že půjde zapnout jenom ve směru B.

* Pokud je A_t větší nebo rovna minimální době vypnutí, pak zapneme ve směru B (zdroj 8).
* Pokud je A_t menší než minimální doba vypnutí tak jsme v situaci, kdy nejsme schopni zdroj legálně zapnout ani směrem A ani směrem B, protože hodnoty A_t a B_t jsou ostře menší než minimální doba vypnutí. Musíme zdroj zapnout, protože jeho nezapnutím, by se daná iterace prohlásila zbytečně za neplatnou. Proto zapneme zdroj ve směru A po dobu A_t a ve směru B po dobu B_t (zdroj 9).

Tabulka 3.10: Tabulka ukazuje jednotlivé situace, které mohou nastat při zapínání zdroje uprostřed plánovacího horizontu. Tabulka hned pod ní ukazuje, zda v dané situaci lze zdroj zapnout a pokud ano, jak. Předpokládáme, že minimální doba běhu zdroje je rovna 2 hodinám a minimální doba vypnutí zdroje je 2. Otazník značí nepodstatnou hodnotu.

		Hodiny											
		1	...	9	10	11	**12**	13	14	15	16	...	n
Zdroj	1	1	...	?	?	1	**0**	1	?	?	?	...	1
	2	1	...	?	?	1	**0**	0	1	?	?	...	1
	3	0	...	?	?	1	**0**	0	0	?	?	...	0
	4	1	...	?	1	0	**0**	1	1	?	?	...	1
	5	0	...	?	0	0	**0**	1	1	?	?	...	0
	6	1	...	1	0	0	**0**	0	0	0	1	...	1
	7	1	...	1	0	0	**0**	0	0	1	?	...	1
	8	1	...	?	1	0	**0**	0	0	1	?	...	1
	9	1	...	?	1	0	**0**	0	1	?	?	...	1

		Hodiny											
		1	...	9	10	11	**12**	13	14	15	16	...	n
Zdroj	1	1	...	?	1	1	**1**	1	?	?	?	...	1
	2	1	...	?	1	1	**1**	1	1	?	?	...	1
	3	0	...	?	1	1	**1**	0	0	?	?	...	0
	4	1	...	?	1	1	**1**	1	1	?	?	...	1
	5	0	...	?	0	0	**1**	1	1	?	?	...	0
	6	1	...	1	0	0	**1**	1	0	0	1	...	1
	7	1	...	1	0	0	**1**	1	0	1	?	...	1
	8	1	...	?	1	1	**1**	0	0	1	0	...	1
	9	1	...	?	1	1	**1**	1	1	?	?	...	1

3.4 Opravy prohledávací fáze

Problematika oprav je natolik rozsáhlá, že si zaslouží vlastní sekci. Už jenom kvůli tomu, že hledání globálního optima stojí na tom, jak jsou provedeny. Jak bylo řečeno, opravy se provádí na základě porušení podmínek pro přípustné řešení. Jsou zde definovány celkem čtyři opravy, které se aplikují vždy podle počtu a typu porušených podmínek, jak uvádí tabulka 3.7.

- Oprava požadovaného výkonu.
- Oprava požadovaných rezerv.
- Oprava minimální doby běhu zdroje.
- Obecná oprava.

3.4.1 Oprava požadovaného výkonu

Tato oprava je použita vždy, pokud při vypnutí zdroje není v danou hodinu splněna podmínka na požadovaný výkon a na minimální dobu zapnutí zdroje. Mohou nastat dvě situace.

- V dané hodině jsou rezervy větší nebo rovny součtu požadovaných rezerv s výkonem, které vyráběl vypnutý zdroj. Je tedy možné vyrobit chybějící výkon na již zapnutých zdrojích.
- V dané hodině jsou rezervy menší než součet požadovaných rezerv a s výkonem, které vyráběl vypnutý zdroj. Je nutné najít nový zdroj, který výkon vyrobí.

Pro opravu je vytvořen seznam zdrojů, s ohledem na předchozí body, které jsou levnější než vypnutý zdroj (viz sekce 3.4). V tento moment je nutné zadefinovat, co se přesně myslí slovy levnější zdroj. A pro tuto definici potřebujeme další definici, konkrétně marginální cenu.

Marginální cena $\lambda_i(P_i)$ je směrnice tečny funkce palivových nákladů (sekce 2.1) pro danou hodnotu výkonu P_i:

$$\lambda_i(P_i) = 2a_iP_i + b_i \tag{3.3}$$

Řekneme, že zdroj A je **levnější** než zdroj B, pokud

$$\lambda_A(P_{A,max}) \leq \lambda_B(P_{B,min}). \tag{3.4}$$

Řekneme, že zdroj A je **pravděpodobně levnější** než zdroj B, pokud

$$\lambda_A \left(P_{A,min}\right) \leq \lambda_B \left(P_{B,min}\right). \tag{3.5}$$

Vytvoří se seznam levnějších zdrojů ve smyslu definice 3.4. Pokud tyto zdroje nestačí na vyrobení chybějícího výkonu pak jsou do seznamu přidány i zdroje, které jsou pravděpodobně levnější (definice 3.5).

Ze seznamu jsou postupně náhodně vybírány zdroje. Pokud vybraný zdroj již nějaký výkon vyráběl, je toto zohledněno. Výkon zdroje nastaven na vhodnou hodnotu:

- Pokud je potřeba vyrobit výkon, který je větší než maximální výkon zdroje, pak zdroj v dané hodině vyrábí maximální výkon.
- Pokud je potřeba vyrobit výkon který je menší než minimální výkon zdroje, pak zdroj v dané hodině vyrábí minimální výkon.
- V ostatních případech zdroj vyrábí v dané hodině přesně chybějící výkon.

Pokud zdroj nevyráběl žádný výkon, pak tu vzniká problematika minimální doby zapnutí. Není možné se ji elegantně obejít jako v případě přechodný bodů (sekce 2.3). Proto se zde bude postupvat dle sekce 3.3. Algoritmus opravy požadovaných rezerv je zobrazen na obrázku 3.3.

3.4.2 Oprava požadovaných rezerv

Tato oprava je použita vždy, pokud při vypnutí zdroje je v danou hodinu splněna podmínka na požadovaný výkon a na minimální dobu zapnutí zdroje, ale nejsou splněny požadované rezervy, tedy vypnutý zdroj byl zdrojem rezervním.

Rezervní zdroj je zdroj, jehož odpojením dochází pouze k nesplnění požadovaných rezerv. Rezervní zdroje jsou většinou dražší než ostatní zapnuté zdroje a neprodukují na maximálním možném výkonu.

Z výše uvedených důvodu proto bude vypnutý rezervní zdroj nahrazen pravděpodobněji levnějším rezervním zdrojem (viz definice 3.5), případně zdroji.

Je vytvořen seznam pravděpodobněji levnějších zdrojů, ale jenom těch, které v danou hodinu neprodukují žádný výkon a zároveň jejich minimální doba běhu je rovna jedné

hodině. Pokud žádné takové zdroje nejsou k dispozici, hodnotí se oprava jako neúspěšná. Existence druhé podmínky je odůvodněna zabráněním velkými změnami v řešení.

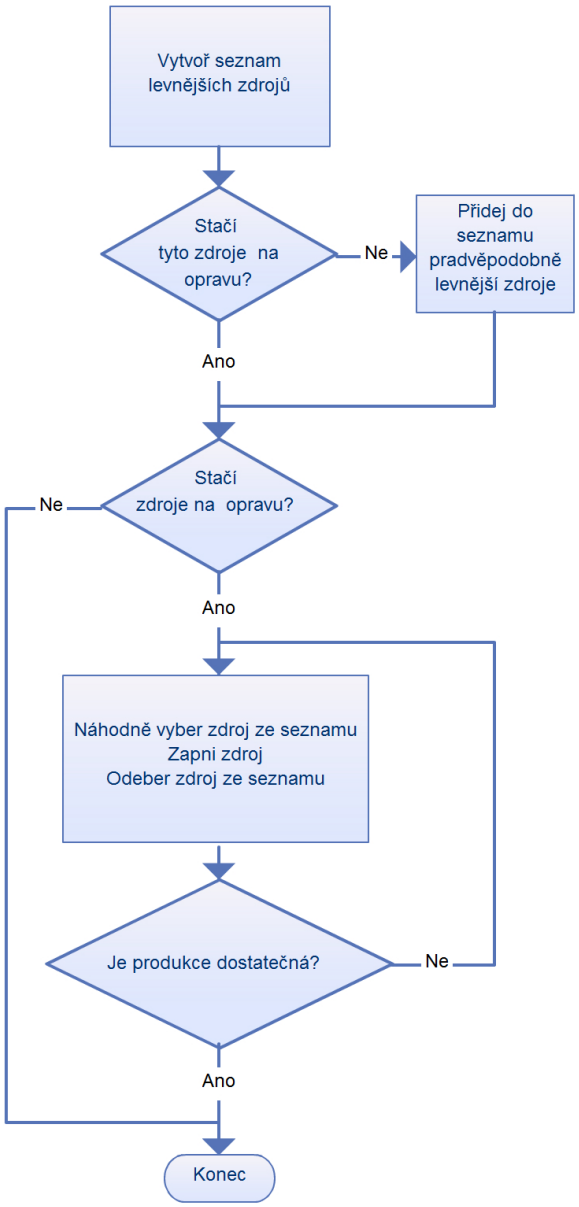
Kompletní algoritmus opravy požadovaných rezerv je zobrazen na obrázku 3.4.

3.4.3 Oprava minimální doby běhu

Tato oprava je použita vždy, pokud při vypnutí zdroje je v danou hodinu splněna podmínka na požadovaný výkon a na požadované rezervy, ale podmínka minimální doby běhu je porušena. Tedy zdroj byl zapnut po minimální dobu zapnutí, např. od času t do času t_1. Jsou uvažovány tři typy oprav:

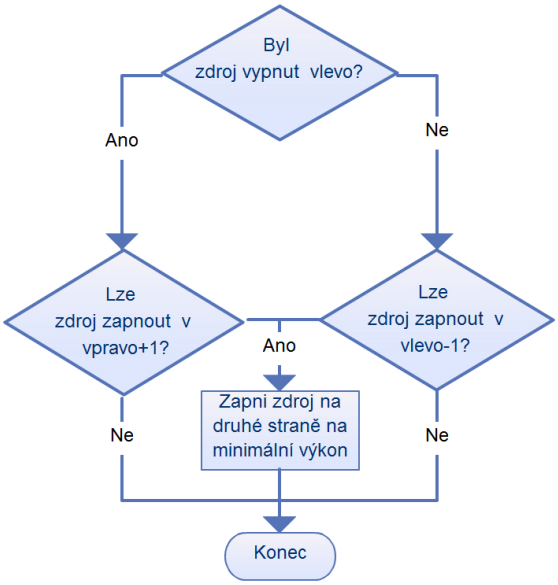
- Přesuneme produkci zdroje do intervalu $t+1$ až t_1+1, pokud měl být zdroj vypnut v čase t.
- Přesuneme produkci zdroje do intervalu $t-1$ až t_1-1, pokud měl být zdroj vypnut v čase t_1.
- Zdroj bude vypnut kompletně v intervalu t až t_1 a dle způsobených problémů budou v každé hodině zváženy další opravy.

Může se zdát, že by výroba byla přesunuta celá, ve skutečnosti se zdroj v požadovanou hodinu vypne a zapne na minimální výkon na druhé straně. Tímto způsobem je zabráněno, aby se porušila podmínka minimální doby běhu. Problém nastává v momentě, kdy přesun způsobí porušení minimální doby vypnutí. Pak se postupně vypíná zdroj po celou dobu běhu od t do t_1 a řeší se problémy s tím související. Kompletní algoritmus opravy minimální doby běhu je zobrazen na obrázku 3.5.



Obrázek 3.3: Oprava požadovaného výkonu. Průběh opravy krok po kroku.

Obrázek 3.4: Oprava požadovaných rezerv. Průběh opravy krok po kroku.



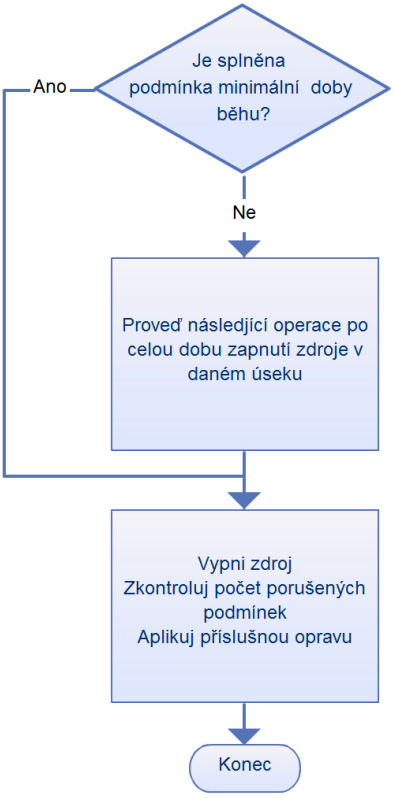
Obrázek 3.5: Oprava minimální doby běhu. Průběh opravy krok po kroku.

3.4.4 Obecná oprava

Tato oprava je použita vždy, pokud vypnutí zdroje v danou hodinu způsobí více jak dvě nesplněné podmínky. Ani v jednom ze zdrojů není popsáno co pod tím autor [3] myslel, proto byla oprava definována jako posloupnost jednodušších pohybů:

- Pokud je splněna minimální doba zapnutí, pak se aplikuje oprava požadovaný výkon s tím rozdílem, že se kontroluje, zda jsou splněny i požadovaného rezervy. Tento proces probíhá velmi podobě, jako kdyby rezervy nebyly brány v úvahu.
- Pokud není splněna minimální doba zapnutí, pak zdroj je postupně vypínán od svého přechodného bodu ke druhému a postupně jsou řešeny problémy tím vznikající.

Kompletní algoritmus obecné opravy je zobrazen na obrázku 3.6.



Obrázek 3.6: Obecná oprava nastupující v případě, že je porušena více jak jedna podmínka. Průběh opravy krok po kroku.

3.5 Pravidla pro vypínání zdrojů běžících celý plánovací horizont

Jeden z důvodů implementace těchto pravidel byl, že konstrukční fáze může vybrat nevhodný zdroj a držet ho zapnutý celý plánovací horizont. Tedy je nutné ho umět nějakým způsobem odstranit nebo se o to alespoň pokusit. Pro zdroje běžící celý plánovací horizont v našem případě platí jedna z uvedených dvou podmínek:

1) Vybraný zdroj nemá ani levý ani pravý přechodný bod.

2) Vybraný zdroj má levý přechodný bod v první hodině plánovacího horizontu a nemá pravý přechodný bod.

Nyní víme, že daný zdroj skutečně produkuje výkon celý plánovací horizont. Otázkou je, kdy a kde ho vypnout. Využijeme, že máme již náhodně vybranou hodinu t z prohledávací části algoritmu. Stačí zjistit, zda je splněna nějaká podmínka ze seznam vypínacích podmínek (tabulka 3.11), který je ve tvaru "pokud $Stav(\cdots)$ nebo $Stav(\cdots)$ pak $Vypni(\cdots)$". $Stav\,(a, b, c, d)$ reprezentuje různé stavy, které jsou studovány:

- $a \in \{ON, OFF\}$ - zda zdroj je v celém plánovacím horizontu pouze zapnu nebo pouze vypnut,
- $b \in \{Vlevo, Vpravo\}$ - směr prohledávání / vypínání,
- $c \in \{ON, OFF\}$ - počáteční stav zdroje,
- d - specifický čas nebo časový interval.

$Vypni\,(t_1, t_2)$ je interval, po který se daný zdroj vypne. $UT\,(i)$ a $DT\,(i)$ je minimální doba běhu a minimální doba vypnutí, $per\_ini\,(i)$ je délka stavu před plánovacím horizontem.

Tabulka 3.11: Pravidla pro vypínání zdrojů běžící celý plánovací horizont. Detailní popis lze nalézt v sekci 3.5.

I

$t_1 = max\left(UT\left(i\right) - per\_ini\left(i\right), 0\right)$
$Stav\left(ON, Vlevo, ON, t \geq DT\left(i\right) + t_1\right)$
$Stav\left(ON, Vlevo, OFF, t \geq UT\left(i\right) + DT\left(i\right)\right)$
$Vypni\left(t - DT\left(i\right) + 1, t\right)$

II

$t_1 = max\left(UT\left(i\right) - per\_ini\left(i\right), 0\right)$
$Stav\left(ON, Vlevo, ON, t < DT\left(i\right) + t_1\right)$
$Vypni\left(t - t_1, t - t_1 + DT\left(i\right) - 1\right)$

III

$Stav\left(ON, Vlevo, OFF, t = 0\right)$
$Stav\left(ON, Vpravo, OFF, t = 0\right)$
$Stav\left(ON, Vpravo, ON, t > 0 \text{ a } t \geq UT\left(i\right) - per\_ini\left(i\right)\right)$
$Stav\left(ON, Vpravo, OFF, t \geq UT\left(i\right)\right)$
$Vypni\left(t, t + DT\left(i\right) - 1\right)$

IV

$t_1 = max\left(UT\left(i\right), t - DT\left(i\right) + 1\right)$
$Stav\left(ON, Vlevo, OFF, t > 0 \text{ a } t < UT\left(i\right) + DT\left(i\right)\right)$
$t_1 = max\left(UT\left(i\right) - per\_ini\left(i\right), 0\right)$
$Stav\left(ON, Vpravo, ON, t < t_1\right)$
$Vypni\left(t_1, t_1 + DT\left(i\right) - 1\right)$

V

$t_1 = max\left(UT\left(i\right) - per\_ini\left(i\right), 0\right)$
$Stav\left(ON, Vpravo, OFF, t \neq 0 \text{ a } t < UT\left(i\right)\right)$
$Vypni\left(UT\left(i\right), UT\left(i\right) + DT\left(i\right) - 1\right)$

Kapitola 4

Implementace

Metodika nebyla mnou vymyšlena, ale program byl mnou naprogramován. Velká část metodiky byly převzata z článku [3]. V některých ohledech bylo algoritmus nutné změnit, ať už kvůli nejasnostem v průběhu, nebo kvůli paralelizaci. Vybraný jazyk pro naprogramování byl jazyk CUDA C, který vytvořila společnost NVIDIA. Dokumentaci k tomuto jazyku lze nalézt na webové stránce [5]. Jedná se o rozšířený programovací jazyk C, který umožňuje paralelní programování nejenom na grafické kartě GTX Titan.

Pro vytvoření a psaní dokumentu byl využit program Microsoft Visual Studio 2012 Professional [8]. Pro vygenerování dokumentace byl použit program [9]. Pro generování komentářů ke kódu byl použit program GhostDoc [10].

4.1 CUDA C

Programovací jazyk CUDA C se příliš neliší od programovacího jazyku C. Samozřejmostí je ukazatelová aritmetiku a vlastní správa paměti. Toto platí zejména pro GPU paměť. Důležitou vlastností tohoto jazyka jsou identifikátory pro rozlišení, zda daný kód se vyhodnocuje na procesoru a nebo na grafické kartě. Nelze načítat data z grafické karty při průběhu kódu na procesoru a stejně tak nemůže grafická karta využívat hodnot, které si nepřekopírovala do své paměti. Proto tu jsou jisté omezení plynoucí na maximální velikost úlohy, kterou lze řešit. Více informací na toto téma lze nalézt v kapitole 5.

Z výše uvedených důvodů je vhodné rozlišovat, kde je pole již jeho názvem. Proto pole alokované na cpu mají předponu "H_"a pole na grafické kartě "D_". Následující označení jsou pro metody a v některých případech tak lze značit i funkce. Musí být označeno před návratovým datovým typem (i neurčitým), protože tím se kompilátoru řekne, kde se daná

metoda bude nacházet.

4.1.1 __device__

Označení je pro metody a funkce, které jsou vykonávány na grafické kartě. Tato část kódu bude nakopírována na grafickou kartu, aby tam mohla být vykonávána během spuštění programu.

4.1.2 __host__

Označení je pro metody a funkce, které budou vykonávány na procesoru. Není nutné toto označovat všude, ale pro přehlednost je to užitečné.

4.1.3 __global__

Označuje pouze metody, které budou vykonány paralelně. Nemůže označovat funkce, protože by nebylo jasné, kterou hodnotu má funkce vrátit. U této metody lze specifikovat, kolik jader a kolik vláken má být vyčleněno na výpočet daného problému za pomoci syntaxe <<< počet_jader,počet_vláken >>>. Ideální počet je 32 nebo 64 vláken na jádro. Zde v práci je využita první možnost.

4.1.4 Shrnutí

- Z kódu, který byl označen __host__:
  - Lze volat kód označený __host__ ,
  - Nelze volat kód označený __device__,
  - Lze vytvářet spouštět jádra na grafické kartě pro metody označené __global__.
- Z kódu, který byl označen __device_
  - Nelze volat kód označený __host__ ,
  - Lze volat kód označený __device__,
  - Pokud je k dispozici grafická karta verze 3.5 a vyšší, pak lze vytvářet spouštět jádra na grafické kartě pro metody označené __global__. Toto nazýváme dynamický paralelismus.

- Z kódu, který byl označen __global__
  - Nelze volat kód označený __host__ ,
  - Lze volat kód označený __device__,
  - Pokud je k dispozici grafická karta verze 3.5 a vyšší, pak lze vytvářet spouštět jádra na grafické kartě pro metody označené __global__. Toto nazýváme dynamický paralelismus.

4.2 Dualita kódu

Pokud je daná metoda / funkce volána jak na grafické kartě tak na procesoru, tak ji zle označit oběma příznaky __host__ a __device__. Program sám pozná, kde je tato metoda prováděna. Toto označujeme jako dualitu kódu. Zde jí bylo využito při porovnání rychlosti výpočtu GRASP algoritmu na procesoru a grafické kartě. Dále byla užitečná při testování a odstraňování chyb, protože šlo sledovat lokální proměnné na procesoru. Lokální proměnné se na grafické kartě zobrazují obtížněji a bez speciálních nástrojů to není ani možné.

U duálního kódu vzniká nebezpečí, že grafická karta se pokusí využít hodnot z procesoru. V takových případech program zahlásí chybu a je násilně ukončen. Dualita kódu není příliš obvyklá, protože se grafické kartě svěřují většinou výpočty, které lze rozdělit na menší části.

4.3 Informace o polích

Úloha nabízí dvě možnosti přístupu. Pole struktur a nebo struktura polí. Druhá možnost byla zvolena, protože často pracuji přímo s jedním polem a ne se všemi parametry daného zdroje. Některá pole jsou dle zadání vícerozměrná. Např. pole hodnot, kolik bude daný zdroj vyrábět v každé hodině časového horizontu je jednodimenzionální. Pokud budeme řešit všechny zdroje naráz, tak máme dvoudimenzionální. Spojením všech řešení získáváme tři dimenzionální pole, kde dimenze jsou postupně řešení, zdroj, hodina. A toto pole je alokováno jako jednodimenzionální pole o délce součinu všech tří délek.

Vznikl zde problém přístupu k jednotlivým prvkům. Proto byla navržena celá třída funkcí a metod, které pracují s více dimenzionálním polem a převádí ho na jedno

dimenzionální a naopak.

Nechť je dáno pole o jednotlivých rozměrech x_l, y_l a z_l, které jsou kladné. Pak lze vždy ke každé trojici [x][y][z] jednoznačně přiřadit [pozici] v jedno dimenzionálním poli a naopak.

4.4 Zisk náhodných čísel

Generování náhodných čísel na CPU a GPU se malinko liší. Na CPU je to jednoduché. Stačí zavolat funkci *rand()*, která je v souboru *stdlib.h*. Tuto funkci stačí inicializovat za pomoci aktuálního času.

Na GPU je to podstatný problém. Jeden z problému je její inicializace. Pokud si všechna vlákna inicializují náhodný generátor ve stejný moment, pak dostanou stejný náhodný generátor. Tomu je nutné zamezit. Proto jako vstupní parametru při tvorbě náhodného generátoru použijeme nejenom čas, ale i pořadí vlákna. Tím je tento problém vyřešen. Díky [6] není nutné řešit přesun těchto generátorů náhodných čísel a stačí mít pouze jejich pole.

4.5 Omezení plynoucí z grafické karty

Úloha byla testována na grafické kartě GeFroce GTX TITAN (verze ovladačů 6.0, CUDA výpočetní verze 3.5, program lze spustit i s výpočetní verzí 2.0). Celková paměť, která je k dispozici, má velikost přibližně 4Gb. Protože GRASP algoritmus potřebuje v každé iteraci prohledávací fáze zálohu aktuálního řešení, pak je efektivní nejvýše polovina paměti. V kapitole 5 je uvedeno podrobnější testování tohoto omezení, konkrétně sekce maximální úloha 5.4. Dle získaných výsledků lze usuzovat, že určitě není možné alokovat více jak 10^9 *float*, ale je možné alokovat 10^8 *float*. Přesné omezení se nachází přibližně někde v tomto intervalu. Jako pomocný výpočet při testování slouží, že součin počet řešení $\cdot$ počet zdrojů $\cdot$ počet hodin nesmí přesáhnout číslo 10^8.

Kapitola 5

Výsledky

Algoritmus je již v tento moment vysvětlen. Je tedy vhodné ho otestovat na úloze nasazování zdrojů elektrické energie. Nejprve je porovnán z hlediska kvality získaného řešení, dále je předvedeno hledání správného parametru α, zobrazena rychlost výpočtu jak na procesoru tak na grafické kartě a nakonec jsou hledány omezení na maximální velikost problému.

5.1 Zadání

Úkolem je rozvrhnout určitý počet zdrojů elektrické energie pro plánovací horizont určité délky hodin. V tabulce 5.1 lze vidět parametry jednotlivých zdrojů, které budou uvažovány. V tabulce 5.2 jsou vidět hodnoty požadovaného napětí a rezerv pro celý plánovací horizont. V případě, že bude potřeba více než deset zdrojů, pak jsou tyto zdroje postupně duplikovány. Plánovací horizont se kopíruje stejným způsobem.

Legenda k tabulce 5.1

- $P_{\max}$(MW) je maximální výkon, který je schopen zdroj vyrobit.
- $P_{\min}$(MW) je minimální výkon, který je schopen zdroj vyrobit.
- a je parametr palivových nákladů měřený v \$/MW^{2}h.
- b je parametr palivových nákladů měřený v \$/MWh.
- c je parametr palivových nákladů měřený v \$/h.

- Min. doba běhu je počet hodin, které musí být zdroj zapnut, než může být opětovně vypnut.
- Min. doba vypnutí je počet hodin, které musí být zdroj vypnut, než může být opětovně zapnut.
- Cena horkého startu je cena zapnutí zdroje, která se použije, pokud je zdroj vypnut kratší dobu než je počet hodin studeného startu.
- Cena studeného startu je cena zapnutí zdroje, která se použije, pokud je zdroj vypnut delší dobu než je počet hodin studeného startu.
- Hodin do studeného startu je počet hodin, po kterých se se jedná už o studený start.
- Počáteční stav je stav zdroje, těsně před plánovacím horizontem. Kladné číslo říká, že zdroj byl zapnut, a záporné, že byl vypnut, daný počet hodin.

Tabulka 5.2: Požadovaný výkon a rezervy

Hodina	Výkon (MW)	Rezervy (MW)	Hodina	Výkon (MW)	Rezervy (MW)
1	700	70	13	1400	140
2	750	75	14	1300	130
3	850	85	15	1200	120
4	950	95	16	1050	105
5	1000	100	17	1000	100
6	1100	110	18	1100	110
7	1150	115	19	1200	120
8	1200	120	20	1400	140
9	1300	130	21	1300	130
10	1400	140	22	1100	110
11	1450	145	23	900	90
12	1500	150	24	800	80

Tabulka 5.1: Parametry jednotlivých zdrojů. Vysvětlení jednotlivých řádků lze nalézt v zadání 5.1.

	Zdroj 1	Zdroj 2	Zdroj 3	Zdroj 4	Zdroj 5
$P_{\max}$(MW)	455	455	130	130	162
$P_{\min}$(MW)	150	150	20	20	25
a ($\$MW^2h$)	0,00048	0,00031	0,00200	0,00211	0,00398
b ($/MWh)	16,19	17,26	16,6	16,5	19,7
c ($/h)	1000	970	700	680	450
Min. doba běhu (h)	8	8	5	5	6
Min. doba vypnutí (h)	8	8	5	5	6
Cena horkého startu ($)	4500	5000	550	560	900
Cena studeného startu ($)	9000	10000	1100	1120	1800
Hodin do studeného startu (h)	5	5	4	4	4
Počáteční stav (h)	+8	+8	-5	-5	-6
	Zdroj 6	Zdroj 7	Zdroj 8	Zdroj 9	Zdroj 10
$P_{\max}$(MW)	80	85	55	55	55
$P_{\min}$(MW)	20	25	10	10	10
a ($/MW^{2}h)	0,00712	0,00079	0,00413	0,00222	0,00173
b ($/MWh)	22,26	27,74	25,92	27,27	27,79
c ($/h)	370	480	660	665	670
Min. doba běhu (h)	3	3	1	1	1
Min. doba vypnutí (h)	3	3	1	1	1
Cena teplého startu ($)	170	260	30	30	30
Cena studeného startu ($)	340	520	60	60	60
Hodin do studeného startu (h)	2	2	0	0	0
Počáteční stav (h)	-3	-3	-1	-1	-1

5.2 Porovnání výsledků

Porovnání výsledků proběhlo spouštěním programu na GPU s parametry uvedenými v tabulce 5.15. Algoritmus našel řešení (tabulka 5.5) za 565 871$. Cena řešení autorů je 565 825$ [3] a je tedy o 46$ $\approx$ 0.01% levnější. Pro toto řešení autoři nevypsali kompletní rozvržení výroby, tedy ho nelze brát jako referenční. Za referenční řešení lze považovat řešení zobrazení v tabulce 5.6. Cena tohoto řešení je 561 170$ [4]. Toto rozvržení bylo

vloženo do programu a spočtena jeho cena na 565 853$, jejíž změna je vysvětlena o pár řádek níže. Cena vzorového řešení je nižší o 18$ $\approx$ 0.01%, než cena řešení získaného algoritmem.

Přehled je zobrazen v tabulce 5.4 Nalezené řešení se od vzorového liší spuštěním devátého zdroje místo osmého ve 20-té hodině. V tabulce je tato skutečnost ztučněna. Zde je seznam změn, které způsobují, že zde uvedená cena se liší od uvedené ve zdroji [4]:

- V první hodině se vyrobí pouze 699MW místo požadovaných 700MW a tedy cena je o 13$ levnější. Argumentují to použitím přesnosti $\epsilon = 1.0\%$.
- Ve třetí hodině je nájezd zdroje č.5. Počáteční stav zdroje je -6 a následovně nemění tento stav po dvě hodiny v plánovacím horizontu. Celkem 8 hodin drží vypnutý stav. Tato hodnota je dvakrát větší než doba studeného startu (4 hodiny). Správná cena je 1800$, místo 900$.
- V páté hodině je zapnut zdroj č.4. Počáteční stav zdroje je -5 a následně nemění tento stav po čtyři hodiny v plánovacím horizontu. Celkem 9 hodin drží vypnutý stav. Tato hodnota je více než dvakrát větší než doba studeného startu (4 hodiny). Správná cena je 1120$, místo 560$.
- V 20-té hodině jsou současně zapnuty zdroje 6, 7 a 8, které byly vypnuté (postupně) 5 , 5 a 6 hodin, tedy všechny splňují podmínku pro studený start. Správná cena je 920$, místo 490$.
- V páté hodině se podařilo Lambda iteraci, příloha A,lépe rozvrhnout výrobu mezi zdroje a cena je nižší o 24$.

Tabulka 5.3: Nastavení vložené do programu pro získání uvedeného řešení v tabulce 5.5.

Počet řešení	4096
Počet zdrojů	10
Počet hodin	24
Počet iteraci prohledávací fáze	1000
Povolená chyba Lambda iterace (MW)	0.10
Výchozí α	0.20
Inkrementace α	0.10

Tabulka 5.4: Porovnání jednotlivých nalezených cen řešení. Bohužel u referenčního řešení není uvedena doba výpočtu.

Řešení	Cena ($)	Čas (s)
Nalezené	565 871	1,30
Refereční	565 853	X
Autoři	565 825	8,65

Tabulka 5.5: Nejlepší řešení nalezené programem. Jeho cena je 565 871$. Rez. jsou rezervy, PN jsou palivové náklady (sekce 2.1) a ZZ je cena za start zdroje (sekce 2.1). Rozdíl od referenčního je v tom, že program zapnul devátý zdroj místo osmého v 20-té hodině. Tučně je označeno, co se liší od referenčního řešení (tabulka 5.6).

Hodina	Vyrobene napeti (MW)										Rez. ($)	PN ($)	ZZ ($)
1	455	245	0	0	0	0	0	0	0	0	210	13685	0
2	455	295	0	0	0	0	0	0	0	0	160	14555	0
3	455	370	0	0	25	0	0	0	0	0	222	16811	1800
4	455	455	0	0	40	0	0	0	0	0	122	18599	0
5	455	390	0	130	25	0	0	0	0	0	202	20021	1120
6	455	360	130	130	25	0	0	0	0	0	232	22389	1100
7	455	410	130	130	25	0	0	0	0	0	182	23262	0
8	455	455	130	130	30	0	0	0	0	0	132	24151	0
9	455	455	130	130	85	20	25	0	0	0	197	27252	860
10	455	455	130	130	162	33	25	10	0	0	152	30058	60
11	455	455	130	130	162	73	25	10	10	0	157	31917	60
12	455	455	130	130	162	80	25	43	10	10	162	33892	60
13	455	455	130	130	162	33	25	10	0	0	152	30058	0
14	455	455	130	130	85	20	25	0	0	0	197	27252	0
15	455	455	130	130	30	0	0	0	0	0	132	24151	0
16	455	310	130	130	25	0	0	0	0	0	282	21515	0
17	455	260	130	130	25	0	0	0	0	0	332	20643	0
18	455	360	130	130	25	0	0	0	0	0	232	22389	0
19	455	455	130	130	30	0	0	0	0	0	132	24151	0
20	455	455	130	130	162	33	25	0	**10**	0	152	30077	920
21	455	455	130	130	85	20	25	0	0	0	197	27252	0
22	455	455	0	0	145	20	25	0	0	0	137	22737	0
23	455	425	0	0	0	20	0	0	0	0	90	17647	0
24	455	345	0	0	0	0	0	0	0	0	110	15428	0

Tabulka 5.6: Opravené řešení z článku [4], kde je uvedena cena 561 170$. Po vložení do programu bylo rozvrženo a spočtena nová cena 565 853$. Rez. jsou rezervy, PN jsou palivové náklady ze sekce 2.1 a ZZ je cena za start zdroje ze sekce 2.1. Tučně je označeno, co se liší od referenčního řešení (tabulka 5.5).

Hodina	Vyrobene napeti (MW)										Rez. ($)	PN ($)	ZZ ($)
1	455	245	0	0	0	0	0	0	0	0	210	13685	0
2	455	295	0	0	0	0	0	0	0	0	160	14555	0
3	455	370	0	0	25	0	0	0	0	0	222	16811	1800
4	455	455	0	0	40	0	0	0	0	0	122	18599	0
5	455	390	0	130	25	0	0	0	0	0	202	20021	1120
6	455	360	130	130	25	0	0	0	0	0	232	22389	1100
7	455	410	130	130	25	0	0	0	0	0	182	23262	0
8	455	455	130	130	30	0	0	0	0	0	132	24151	0
9	455	455	130	130	85	20	25	0	0	0	197	27252	860
10	455	455	130	130	162	33	25	10	0	0	152	30058	60
11	455	455	130	130	162	73	25	10	10	0	157	31917	60
12	455	455	130	130	162	80	25	43	10	10	162	33892	60
13	455	455	130	130	162	33	25	10	0	0	152	30058	0
14	455	455	130	130	85	20	25	0	0	0	197	27252	0
15	455	455	130	130	30	0	0	0	0	0	132	24151	0
16	455	310	130	130	25	0	0	0	0	0	282	21515	0
17	455	260	130	130	25	0	0	0	0	0	332	20643	0
18	455	360	130	130	25	0	0	0	0	0	232	22389	0
19	455	455	130	130	30	0	0	0	0	0	132	24151	0
20	455	455	130	130	162	33	25	**10**	0	0	152	30058	920
21	455	455	130	130	85	20	25	0	0	0	197	27252	0
22	455	455	0	0	145	20	25	0	0	0	137	22737	0
23	455	425	0	0	0	20	0	0	0	0	90	17647	0
24	455	345	0	0	0	0	0	0	0	0	110	15428	0

5.3 Vliv α a počet iterací prohledávací fáze.

Jak bylo zmíněno, GRASP využívá pro ovlivňování výběru dalšího zdroje parametr α a tedy jeho volba velmi silně ovlivní konečný výsledek. V tabulkách 5.7, 5.8 a 5.9 lze vidět, jak výsledné řešení ovlivňuje parametr α a jeho inkrementace. Nelze jednoznačně říci, zda má cena tendenci se zvyšovat s rostoucím počtem iterací prohledávací fáze. Hodnota pro $\alpha = 0.50$ a inkrementaci $\alpha = 0.00$ se sice zvyšuje, ale hodnota pro $\alpha = 0.25$ a její inkrementaci 0.75 se naopak snižuje. V tabulce kurzívou.

Pohledem do tabulek lze 5.10, 5.11 a 5.12 lze vidět informace o prodloužení výpočtu. Pokud se počet iterací zvětší z 10 na 100, pak se doba výpočtu prodlouží alespoň o 0.09 a nejvýše o 0.12 vteřiny. Pokud se počet iterací zvýší ze 10 na 1000, pak se doba výpočtu prodlouží alespoň o 0.95 a nejvýše o 1.30 vteřiny.

Tabulky 5.7, 5.10, 5.7, 5.11, 5.8 a 5.12 vždy po dvojici sdílejí stejná data, které lze také porovnávat. Je zde vidět, jak dopadne řešení, které v každém kroku vybere vždy nejlevnější zdroj. Je to řešení, které má $\alpha = 0.00$ a její inkrementaci 0.00. Dolní odhad se v konstrukční fázi počítá z dat zdrojů, které lze v dané hodině zapnout. Pokud je vybrán nejlevnější zdroj, pak je vyhozen a díky nulové inkrementaci se situace opakuje v další hodině. Ani 1000 iterací prohledávací fáze cenu řešení nezměnilo. Toto řešení je o 0.98% dražší než referenční.

Tabulka 5.7: Porovnání vlivu α a její inkrementace na konečnou cenu řešení. α inkrementaci udává, o kolik se zvedne α v případě vybrání zdroje. Nejlevnější řešení je označeno tučně. Kurzívou je řešení, které s počtem iterací roste cena. Počet iterací prohledávací fáze byl roven 10.

Počet řešení		α inkrementace				
10		0.00	0.25	0.50	0.75	1.00
Výchozí α	0.00	571386	569928	579261	587988	595074
	0.25	567634	576677	587988	*598668*	598668
	0.50	**566532**	587988	596561	596561	596561
	0.75	576259	602324	602324	602324	602324
	1.00	604158	604158	604158	604158	604628

Tabulka 5.8: Porovnání vlivu α a její inkrementace na konečnou cenu řešení. α inkrementaci udává, o kolik se zvedne α v případě vybrání zdroje. Nejlevnější řešení je označeno tučně. Počet iterací prohledávací fáze byl roven 100.

Počet řešení		α inkrementace				
100		0.00	0.25	0.50	0.75	1.00
Výchozí α	0.00	571386	570686	581426	588219	596925
	0.25	568048	574858	593427	*599335*	599335
	0.50	**566155**	589777	599495	599495	599495
	0.75	574763	596718	596718	596718	596718
	1.00	608853	608853	608853	608853	608853

Tabulka 5.9: Porovnání vlivu α a její inkrementace na konečnou cenu řešení. α inkrementaci udává, o kolik se zvedne α v případě vybrání zdroje. Nejlevnější řešení je označeno tučně. Počet iterací prohledávací fáze byl roven 1000.

Počet řešení		α inkrementace				
1000		0.00	0.25	0.50	0.75	1.00
Výchozí α	0.00	571386	569942	581418	583362	594479
	0.25	567596	577218	589759	*599390*	596815
	0.50	**566878**	590380	600056	599114	595477
	0.75	574867	602070	604702	598389	599431
	1.00	603583	603749	607554	608695	607030

Tabulka 5.10: Porovnání vlivu α a její inkrementace na konečnou cenu řešení. α inkrementaci udává, o kolik se zvedne α v případě vybrání zdroje. Nejrychlejší řešení je označeno tučně. Počet iterací prohledávací fáze byl roven 10.

Počet řešení		α inkrementace				
10		0.00	0.25	0.50	0.75	1.00
Výchozí α	0.00	**0,05**	0,06	0,06	0,07	0,07
	0.25	0,05	0,06	0,07	0,07	0,07
	0.50	0,06	0,07	0,07	0,07	0,07
	0.75	0,06	0,07	0,07	0,07	0,07
	1.00	0,07	0,07	0,07	0,07	0,07

Tabulka 5.11: Porovnání vlivu α a její inkrementace na konečnou cenu řešení. α inkrementaci udává, o kolik se zvedne α v případě vybrání zdroje. Nejrychlejší řešení je označeno tučně. Počet iterací prohledávací fáze byl roven 100.

Počet řešení		α inkrementace				
100		0.00	0.25	0.50	0.75	1.00
Výchozí α	0.00	**0,14**	0,17	0,18	0,18	0,19
	0.25	0,15	0,17	0,18	0,19	0,19
	0.50	0,16	0,18	0,18	0,18	0,18
	0.75	0,18	0,19	0,19	0,19	0,19
	1.00	0,19	0,19	0,19	0,19	0,19

Tabulka 5.12: Porovnání vlivu α a její inkrementace na konečnou cenu řešení. α inkrementaci udává, o kolik se zvedne α v případě vybrání zdroje. Nejrychlejší řešení je označeno tučně. Počet iterací prohledávací fáze byl roven 1000.

Počet řešení		α inkrementace				
1000		0.00	0.25	0.50	0.75	1.00
Výchozí α	0.00	**1,00**	1,22	1,30	1,34	1,36
	0.25	1,13	1,30	1,35	1,36	1,36
	0.50	1,22	1,35	1,36	1,35	1,35
	0.75	1,29	1,36	1,36	1,36	1,35
	1.00	1,36	1,36	1,36	1,37	1,36

5.4 Maximální úloha

Důležitá otázka je, jak velké problémy lze řešit na grafické kartě. Postupně byly testovány alokace všech proměnných v závislosti na třech nejdůležitějších parametrech. Konkrétně na počtu požadovaných řešení, počtu zdrojů a délce plánovacího horizontu. Výpočetní kapacity byly testovány následovně. Vytvořil se objekt GRASP a byl přesunut na grafickou kartu. Při vytvoření objektu dojde k alokaci paměti na CPU. Testováno při nastavení kompilace na 64 bitový systém. Výsledky lze nalézt v tabulkách 5.13 a 5.14.

Číslo 2 znamená, že bylo možné alokovat, 1 že pouze na CPU a 0, že se vůbec nepodařilo. Ze zmíněných tabulek je vidět, že lze např. provádět výpočet pro 10000 řešení, 10 zdrojů a 1000 hodin, a nebo 100 řešení, 100 zdrojů a 10000 hodin. Jedna ze špatných vlastností algoritmu je, že v každém momentu musí mít zálohu řešení, tedy efektivní výpočet lze provádět pouze na polovině paměti grafické karty.

Tabulka 5.13: Přehled, jak velkou úlohu lze alokovat na CPU a GPU. Číslo 2 znamená, že úloha byla úspěšně alokovaná na CPU i GPU, číslo 1 znamená, že úloha byla úspěšně alokovaná na CPU, ale na GPU nikoliv a číslo 0, že úlohu nelze spustit ani na CPU. Počet řešení pro kterou byla úloha alokována je 100.

Počet řešení		Počet hodin				
100		1	10	100	1000	10000
Počet zdrojů	1	2	2	2	2	2
	10	2	2	2	2	2
	100	2	2	2	2	2
	1000	2	2	2	2	0
	10000	2	2	2	0	0

Tabulka 5.14: Přehled, jak velkou úlohu lze alokovat na CPU a GPU. Číslo 2 znamená, že úloha byla úspěšně alokovaná na CPU i GPU, číslo 1 znamená, že úloha byla úspěšně alokovaná na CPU, ale na GPU nikoliv a číslo 0, že úlohu nelze spustit ani na CPU. Počet řešení pro kterou byla úloha alokována je právě 10 000.

Počet řešení		Počet hodin				
10000		1	10	100	1000	10000
Počet zdrojů	1	2	2	2	2	0
	10	2	2	2	2	0
	100	2	2	2	0	0
	1000	2	2	0	0	0
	10000	0	0	0	0	0

5.5 Vliv počtu řešení

V následující sekci je shrnut vliv počtu řešení na dobu výpočtu jak na grafické kartě tak na procesoru. Jednotlivé počty řešení jsou 32, 320 a 3200. Počty zdrojů a délka plánovacího horizontu je uvedena v tabulce vždy pro CPU i GPU verzi programu. Tedy z tabulek 5.16, 5.18 a 5.20 lze vidět, že pokud je při výpočtu na CPU počet požadovaných řešení zvětšen 10 krát (z 32 řešení na 320 řešení), pak i výpočetní čas je zvednut o přibližně 10 krát (navýšení doby výpočtu o 834.95%) . Naproti tomu u GPU je situace v tabulkách 5.17, 5.19 a 5.21 skoro stejná, výpočet se při zvětšení problému, také 10 krát, prodlouží přibližně o 0.01 (1.08%) vteřiny. Při navýšení z 320 řešení na 3200 se výpočetní čas CPU opět prodlouží přibližně 10 krát (828.97%) a u GPU přibližně o $0,09$ $(9,46\%)$ vteřiny.

V tabulkách 5.22, 5.23, 5.24 a 5.25 je uvedená situace pro dva počty požadovaných řešení, konkrétně pro 125 a 3125. Lze vidět, jak se mění doba výpočtu, když se počet zdrojů a nebo délka plánovacího horizontu zvětší 5 krát. Hodnota -1 znamená, že úlohu se nepodařilo alokovat. Z těchto byl vybrán úsek a zobrazen na obrázcích 5.1, 5.2, 5.3 a 5.4. Je vidět, že pokud jsou požadovány větší počty řešení, pak je vhodné zvolit GPU.

Tabulka 5.15: Nastavení pro získání uvedených hodnot v sekci 5.5

Počet iteraci prohledávací fáze	1000
Povolená chyba Lambda iterace (MW)	0.10
Výchozí α	0.20
Inkrementace α	0.10

Tabulka 5.16: Vliv počtu zdrojů a hodin na dobu výpočtu. CPU, 32 řešení.

Počet řešení		Počet hodin		
32		24	48	72
P. zdrojů	10	0,05	0,07	0,08
	50	0,09	0,14	0,19
	90	0,13	0,21	0,29

Tabulka 5.17: Vliv počtu zdrojů a hodin na dobu výpočtu. GPU, 32 řešení.

Počet řešení		Počet hodin		
32		24	48	72
P. zdrojů	10	0,99	1,55	2,11
	50	3,64	6,04	8,34
	90	6,07	9,91	13,75

Tabulka 5.18: Vliv počtu zdrojů a hodin na dobu výpočtu. CPU, 320 řešení.

Počet řešení		Počet hodin		
320		24	48	72
P. zdrojů	10	0,47	0,64	0,80
	50	0,90	1,37	1,86
	90	1,29	2,07	2,94

Tabulka 5.19: Vliv počtu zdrojů a hodin na dobu výpočtu. GPU, 320 řešení.

Počet řešení		Počet hodin		
320		24	48	72
P. zdrojů	10	1,00	1,56	2,16
	50	3,77	6,21	8,58
	90	6,24	10,18	14,22

Tabulka 5.20: Vliv počtu zdrojů a hodin na dobu výpočtu. CPU, 3200 řešení.

Počet řešení		Počet hodin		
3200		24	48	72
P. zdrojů	10	4,37	6,27	7,94
	50	8,31	13,42	18,52
	90	12,09	20,54	29,18

Tabulka 5.21: Vliv počtu zdrojů a hodin na dobu výpočtu. GPU, 3200 řešení.

Počet řešení		Počet hodin		
3200		24	48	72
P. zdrojů	10	1,09	1,70	2,31
	50	4,12	6,67	9,25
	90	6,84	11,12	15,59

Tabulka 5.22: Vliv počtu zdrojů a hodin na dobu výpočtu. Hodnota −1 znamená, že úlohu se nepodařilo alokovat. CPU, 125 řešení.

Počet řešení	Počet hodin			
125	24	120	600	3000
10	1,25	1,47	3,03	14,31
50	1,41	1,13	6,84	88,78
250	2,13	4,22	33,97	450,97
1250	6,28	23,91	185,50	2869,34

Tabulka 5.23: Vliv počtu zdrojů a hodin na dobu výpočtu. Hodnota −1 znamená, že úlohu se nepodařilo alokovat. GPU, 125 řešení.

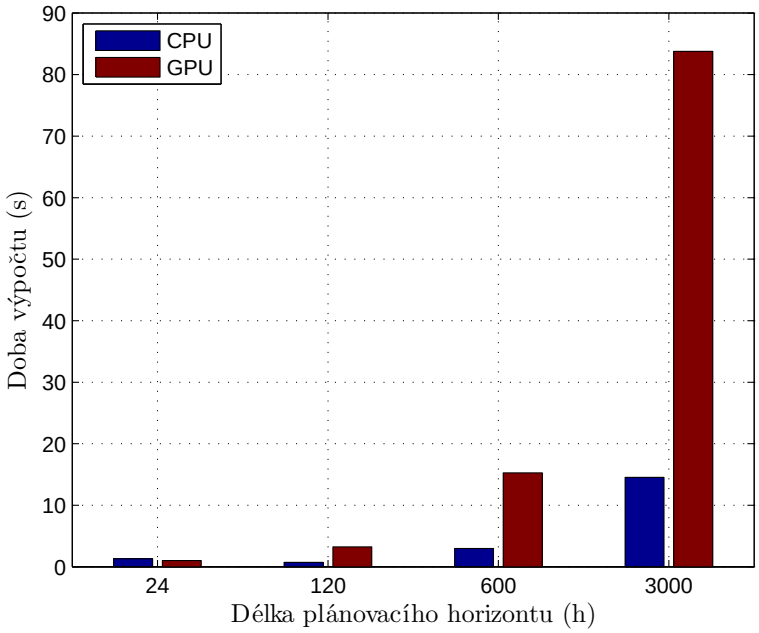
Počet řešení	Počet hodin			
125	24	120	600	3000
10	1,00	3,25	15,29	83,79
50	3,65	13,35	68,92	-1,00
250	15,20	56,68	-1,00	-1,00
1250	69,85	-1,00	-1,00	-1,00

Tabulka 5.24: Vliv počtu zdrojů a hodin na dobu výpočtu. Hodnota −1 znamená, že úlohu se nepodařilo alokovat. CPU, 3200 řešení.

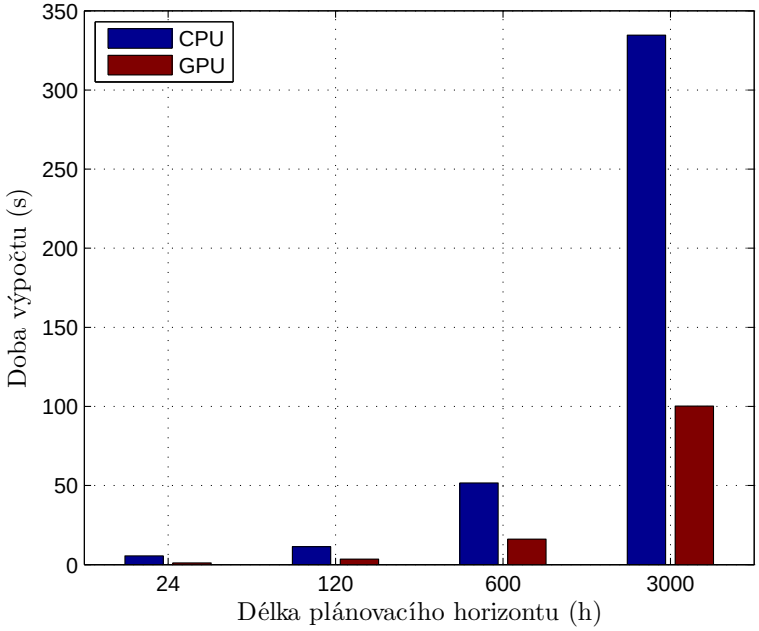
Počet řešení	Počet hodin			
3125	24	120	600	3000
10	5,50	11,31	51,63	333,00
50	7,94	28,06	170,81	2211,97
250	26,28	105,59	848,47	-1,00
1250	114,75	531,25	-1,00	-1,00

Tabulka 5.25: Vliv počtu zdrojů a hodin na dobu výpočtu. Hodnota −1 znamená, že úlohu se nepodařilo alokovat. GPU, 3200 řešení.

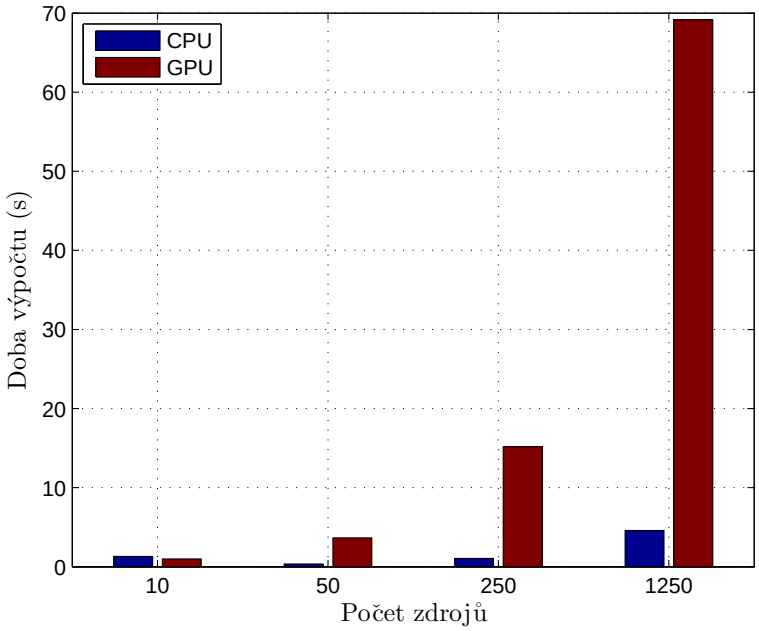
Počet řešení	Počet hodin			
3125	24	120	600	3000
10	1,09	3,52	16,10	99,66
50	4,09	14,49	83,62	-1,00
250	17,00	71,73	-1,00	-1,00
1250	81,27	-1,00	-1,00	-1,00



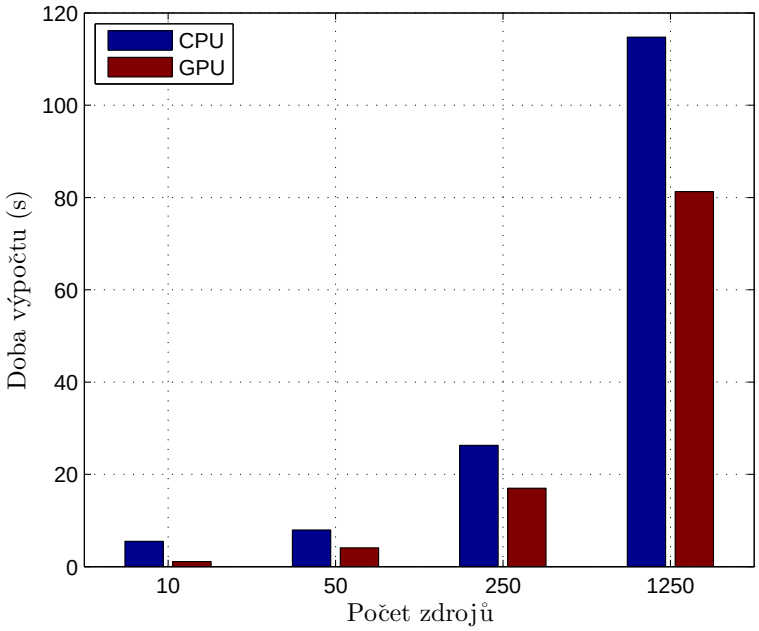
Obrázek 5.1: Porovnání doby výpočtu stejné úlohy na CPU a GPU.
125 řešení, 10 zdrojů.



Obrázek 5.2: Porovnání doby výpočtu stejné úlohy na CPU a GPU.
3125 řešení, 10 zdrojů.



Obrázek 5.3: Porovnání doby výpočtu stejné úlohy na CPU a GPU. 125 řešení, 24 hodin.



Obrázek 5.4: Porovnání doby výpočtu stejné úlohy na CPU a GPU. 3125 řešení, 24 hodin.

Kapitola 6

Závěr

V rámci této bakalářské práce je možné se seznámit s algoritmem GRASP a s úlohou nasazování zdrojů elektrické energie, kterou algoritmus řeší. Byla naprogramována upravená verze algoritmu v programovacím jazyce CUDA C od společnosti NVIDIA. Při programování byl brán ohled na dualitu kódu, tedy aby mohla být ta samá část kódu vypočtena na procesoru nebo na grafické kartě. Tato dualita, ač v konečném důsledku komplikovala a prodlužovala programování, zjednodušila výsledné testování.

Pokud je algoritmus spuštěn na grafické kartě, pak je využito paralelních výpočtů. Ukazuje se, že paralelizace byla úspěšná, ačkoliv je implementována velice jednoduchým způsobem. Počáteční obavy, že výpočetní výkon jednotlivých jader na grafické kartě nebude dostačující, se ve výsledku ukázaly jako neopodstatněné. Pro reálné nasazení paralelizovaného algoritmu je vhodné vyřešit problémy týkající se nemožnosti aplikovat dynamický paralelismus, kdy vlákna si spouštějí další vlákna, v duálním kódu. Zrychlení díky dynamickému paralelismu by se projevilo nejspíše až u problémů z reálného světa, které jsou mnohem rozměrnější, protože spouštění vláken trvá určitou dobu.

Při porovnávání rychlosti výpočtu na procesoru a na grafické kartě se ukazuje, že doba výpočtu se nejpodstatněji prodlužuje při zvětšování počtu požadovaných řešení. Pokud je na CPU počítáno 10 krát více řešení, pak se prodlužuje čas výpočtu 10 krát (o 834.95%). Naproti GPU se čas výpočtu prodlužuje přibližně o 0.01 vteřiny (1.08%). Pro 320 řešení je vhodné použít CPU a pro 3200 již GPU.

Cena přepočteného referenčního řešení je 565 853$, neboť ve zdrojovém článku byla cena nesprávně spočtena. Prezentovaná cena řešení autory této metodiky je 565 825$. Cena nalezená mou implementací nejlevnějšího řešení je 565 871$ a tato cena je vzdálena od optimální hodnoty o 0.01%. Z hodnoty je vidět, že cena je velmi blízko řešení

uvedenému v nejnovější vědecké literatuře. Toto řešení je navíc nalezeno v lepším čase, o $84,97\%$ rychleji.

Průběh práce komplikovaly zejména nejasné formulace týkající se přechodných bodů v zadané literatuře. Autoři se např. pokoušeli vypínat zdroj v místech, kde je vypnutý. U oprav nebylo jasné, zda mohou zapínat i zdroj, který byl před momentem vypnut. Algoritmus obecné opravy nebyl nikde uveden, autoři se odkazují na dokument ve kterém o dané definici nejsou informace. V rámci této práce byly vyřešeny všechny výše zmíněné problémy a jejich řešení implementováno.

Literatura

[1] VIANA, Ana, Jorge Pinho DE SOUSA a Manuel MATOS. A NEW META-HEURISTIC APPROACH TO THE UNIT COMMITMENT PROBLEM. 2002. vyd. 14th PSCC: Sevilla, 2002. Dostupné z: http://pscc.ee.ethz.ch/uploads/tx_ethpublications/s05p05.pdf

[2] VIANA, Ana a Jorge Pinho DE SOUSA. Using GRASP to Solve the Unit Commitment Problem. Manufactured in The Netherlands: Kluwer Academic Publishers, 2003.

[3] VIANA, Ana, J. Pinho DE SOUSA a Manuel A. MATOS. Fast solutions for UC problems by a new metaheuristic approach. Porto: Campus da FEUP, 2008.

[4] DUTTA, Saptarshi a Dilip DATTA. A Binary-Real-Coded Differential Evolution for Unit Commitment Problem: A Preliminary Study. Multi-disciplinary Trends in Artificial Intelligence [online]. 2011, č. 7080, s. 406 [cit. 2014-05-22]. DOI: 10.1007/978-3-642-25725-4_36. Dostupné z: http://link.springer.com/10.1007/978-3-642-25725-4_36

[5] NVIDIA. NVIDIA Developer Zone [online]. v6.0. 2014, 13.2. [cit. 2014-05-22]. Dostupné z: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

[6] NVIDIA. CuRAND [online]. v6.0. 2014, 13.2. [cit. 2014-05-22]. Dostupné z: http://docs.nvidia.com/cuda/curand/#axzz32TjpJtOt

[7] UDACITY. Introduction to Parallel Programming. UDACITY. Youtube [online]. 2012 [cit. 2014-05-22]. Dostupné z: http://youtu.be/zb49vDrOxgA

[8] MICROSOFT. Visual Studio: Domovská stránka [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://www.visualstudio.com/cs-cz/visual-studio-homepage-vs.aspx

[9] Doxygen [online]. 2012 [cit. 2014-05-22]. Doxygen Download Page. Dostupné z: `http://www.stack.nl/~dimitri/doxygen/download.html#latestsrc`

[10] SUBMAIN. GhostDoc: Simplify your XML Comments [online]. 2014 [cit. 2014-05-22]. Dostupné z: `http://submain.com/products/ghostdoc.aspx`

[11] Economic Dispatch of Generated Power Using Modified Lambda - Iteration Method. [online]. [cit. 2014-02-06]. Dostupné z: `http://www.iosrjournals.org/iosr-jeee/Papers/Vol7-issue1/H0714954.pdf?id=5802`

[12] OVERBYE, Tom a Ross BALDICK. EE369 POWER SYSTEM ANALYSIS: Lecture 16 - Economic Dispatch. In: [online]. [cit. 2014-05-02]. Dostupné z: `http://users.ece.utexas.edu/~baldick/classes/369/Lecture_16.ppt`

[13] HAMHALTER, Jan a Jaroslav TIŠER. Diferenciální počet funkcí více proměnných: Vázané extrémy. Vyd. 2. Praha: Česká technika - nakladatelství ČVUT, 2005c1997, s. 116-124. ISBN 80-01-03356-2.

Příloha A

Lambda iterace

Lambda iterace je v programu využívána na rozdělení výroby výkonu v danou hodinu. Minimalizuje celkovou cenu daného řešení vhodným rozvržením výroby elektrické energie mezi jednotlivé zdroje, které v dané iteraci vybrala konstrukční fáze. Rozvržení, v programu, probíhá s ohledem na minimální a maximální výkon všech zdrojů.

Pro první přiblížení byla použita metoda, která je odvozená v literatuře [11]. Článek obsahuje nejasnosti a překlepy, kvůli kterým nebylo možné metodu zprovoznit a publikované výsledky zopakovat. Proto byla vybraná metoda modifikována za pomoci materiálu z přednášek [12]. Bohužel nová metoda není nikde odvozena, proto je vhodné ukázat, že dokáže problém vyřešit.

Nejprve je zde ukázáno, jak lze problém analyticky řešit, sekce A.1. Pak se dojde k závěru, že analytické řešení, není schopné zohledňovat omezení plynoucí z jednotlivých zdrojů, konkrétně jejich minimální a maximální výkon. Poté je přiblíženo, jak tato omezení zahrnout do algoritmu A.2. Nakonec je zmíněna implementace A.3 a vliv maximální povolené chyby ϵ na dobu výpočtu, tabulka A.1.

A.1 Odvození problému

Nechť je zadáno $n \in \mathbb{N}$ zdrojů. Nechť $F_i(P_i)$ je i-tá ($i \in \{0, 1, \cdots, n-1\}$) funkce ceny paliva i-tého zdroje v závislosti na jejím aktuálním výkonu P_i (dle 2.1), tedy každý zdroj má vlastní koeficienty $a_i, b_i, c_i \in \mathbb{R}$, minimální výkon $P_{i,min}$ a maximální výkon $P_{i,max}$ a platí, že

$$P_{i,min} \leq P_i \leq P_{i,max}.$$

Dále je zadána vazební podmínka

$$G\left(P_0, P_1, \cdots, P_{n-1}\right) : \sum_{i=0}^{n-1} P_i - P_d = 0 \tag{A.1}$$

kde $P_d \in \mathbb{R}^+ \setminus \{0\}$ je požadovaný výkon v síti. Úkolem je minimalizovat funkci FC A.2.

$$FC\left(P_0, P_1, \cdots, P_{n-1}\right) = \sum_{i=0}^{n-1} F_i(P_i) \tag{A.2}$$

Množina G, zadaná rovnicí A.1, je omezená a uzavřená a funkce FC A.2 je spojitá, tedy FC nabývá svého minima i maxima vzhledem k G. Pro nalezení extrému je použita metoda Lagrangeových multiplikátorů [13], pro kterou je sestavena Lagrangeova funkce L A.3.

$$L\left(P_0, P_1, \cdots, P_{n-1}\right) = FC\left(P_0, P_1, \cdots, P_{n-1}\right) - \lambda \cdot G\left(P_0, P_1, \cdots, P_{n-1}\right) \tag{A.3}$$

Podmínky pro stacionární body funkce L společně s vazebnou podmínkou G jsou tedy

$$\begin{aligned}
\tfrac{\partial L}{\partial P_0} &= 0 && & 2 \cdot a_0 \cdot P_0 + b_0 - \lambda &= 0 \\
\tfrac{\partial L}{\partial P_1} &= 0 && \text{tj.} & 2 \cdot a_1 \cdot P_1 + b_1 - \lambda &= 0 \\
&\vdots && & &\vdots \\
\tfrac{\partial L}{\partial P_{n-1}} &= 0 && & 2 \cdot a_{n-1} \cdot P_{n-1} + b_{n-1} - \lambda &= 0 \\
G\left(P_0, P_1, \cdots, P_{n-1}\right) &= 0 && & \sum_{i=0}^{n-1} P_i - P_d &= 0
\end{aligned}$$

Vyjádřením z prvních n rovnic je získána závislost výkonu i-té zdroje na parametru λ.

$$P_i(\lambda) = \frac{\lambda - b_i}{2 \cdot a_i} \tag{A.4}$$

Dosazením do vazební podmínky G A.1

$$\frac{\lambda - b_1}{2 \cdot a_1} + \frac{\lambda - b_2}{2 \cdot a_2} + ... \frac{\lambda - b_{n-1}}{2 \cdot a_{n-1}} - P_d = 0$$

a následným vyjádřením parametru λ je získána jeho optimální hodnota A.5. Nyní lze dopočítat optimální hodnotu A.4 pro každý zdroj.

$$\lambda = \frac{\sum_{i=0}^{n-1} \frac{b_i}{a_i} + 2 \cdot P_d}{\sum_{i=0}^{n-1} \frac{1}{a_i}} \tag{A.5}$$

Jelikož tato metoda musí být schopna pracovat obecně, lze tento výraz upravit za pomoci dat z [3]. Pro zjednodušení a ukázání problému předpokládejme, že $b_i \approx K \cdot a_i$ a $a_i \approx \frac{1}{L}$, pro $K, L > 0$. Tedy

$$\sum_{i=0}^{n-1} \frac{b_i}{a_i} \approx \sum_{i=0}^{n-1} K = K \cdot n$$

$$\sum_{i=0}^{n-1} \frac{1}{a_i} \approx \sum_{i=0}^{n-1} L = L \cdot n$$

$$\lambda \approx \frac{K \cdot n + 2 \cdot P_d}{L \cdot n} = \frac{K}{L} + \frac{2 \cdot P_d}{n \cdot L}$$

Pomocí tohoto výrazu mohou být vyjádřeny jednotlivé výkony

$$P_i(\lambda) = \frac{\lambda - b_j}{2 \cdot a_j} = \frac{L}{2} \cdot \lambda - \frac{K}{2} = \frac{L}{2} \cdot \frac{K}{L} + \frac{L}{2} \cdot \frac{2 \cdot P_d}{n \cdot L} - \frac{K}{2} = \frac{P_d}{n}. \tag{A.6}$$

Za uvedených předpokladů je optimální $P_i = \frac{P_d}{n}$.

Nechť $P_d = 1000$, $n = 10$, $P_{3,min} = 150$. Optimum je díky vztahu A.6 získáno pro $P_i = 100$. V tento moment nemá řešení smysl, protože nebyl dodržen minimální výkon třetího zdroje. Nejjednodušší způsob, jak zohledňovat minimální a maximální výkony jednotlivých zdrojů je, zahrnout je do výpočtu iteračně, tedy pro každé λ je testováno, kolik skutečně výkonu všechny zdroje vyprodukují. Tento postup je popsán v A.2.

A.2 Výsledný průběh Lambda iterace

Transformace A.4 je nadále upravena a jsou do ní zahrnuty omezení na výkon zdroje.

$$P_{i,\mathrm{om}}(\lambda) = \begin{cases} P_{i,min} & , P_i(\lambda) < P_{i,min} \\ P_{i,max} & , P_i(\lambda) > P_{i,max} \\ P_i(\lambda) & , \text{jinak} \end{cases}$$

Tato transformace má tu výhodu, že parametr λ definuje výkony jednotlivých zdrojů současně. Tedy stačí najít takové λ, že

$$G\left(P_{0,\mathrm{om}}(\lambda), P_{1,\mathrm{om}}(\lambda), \cdots, P_{n-1,\mathrm{om}}(\lambda)\right) = P_d$$

podobně jako se za pomoci metody půlení intervalu hledají kořeny polynomu, budeme zde hledat hodnotu λ. Potřebujeme horní λ_h a dolní odhad λ_d.

$$\lambda_h \geq 2 \cdot a_i \cdot P_{i,max} + b_i \tag{A.7}$$

$$\lambda_d \leq 2 \cdot a_i \cdot P_{i,min} + b_i \tag{A.8}$$

stvořme účelovou funkci

$$D(\lambda) = G(P_{0,\mathrm{om}}(\lambda), P_{1,\mathrm{om}}(\lambda), \cdots, P_{n-1,\mathrm{om}}(\lambda)) - P_d \tag{A.9}$$

λ_h splňuje, že $D(\lambda_h) \geq 0$ a λ_d splňuje, že $D(\lambda_d) \leq 0$ a funkce $D(\lambda)$ má pouze jeden průnik s osou x. Za pomoci metody půlení intervalu lze nalézt takové λ, že $D(\lambda) = 0$.

Tabulka A.1: Dopad maximální dovolené chyby ϵ na dobu výpočtu Lambdy iterace.

ϵ (MW)	t (ms)
1,000	37,50
0,100	45,30
0,010	54,70
0,005	56,50
0,004	48,84

A.3 Implementace Lambda iterace

Implementace byla provedena s ohledem na splnění omezujících podmínek 2.2. Pokud je lambda iterace ukončena a horní a dolní odhad lambdy není stejný, a tato situace může nastat díky parametru ϵ, pak je jako λ označen její horní odhad. Tím je zaručeno, že omezující podmínky jsou splněny. Nemůže se stát, že by se Lambda iteraci, aplikovanou po konstrukční fázi, nepodařilo rozvrhnout výkony mezi zdroje.

Vhodná počáteční hodnota λ_h je spočtena z postupným dosazováním do předpisu A.7 a nalezením největší hodnoty. Nelze říci, že by nejvyšší hodnota byla získána pro nejvyšší dovolené $P_{i,\max}$, neboť tato hodnota silně závisí na koeficientech a_i a b_i.

Vhodná počáteční hodnota λ_d je nastavena 0, protože určitě splňuje rovnici A.8. Na jednu stranu to způsobí několik iterací navíc, na stranu druhou není nutného žádného výpočtu, neboť tato hodnota nezávisí na datech.

Epsilon ϵ zde reprezentuje maximální dovolenou chybu, které lambda iterace může dosáhnout,

$$D(\lambda) \geq -\epsilon$$

Pokud nedojde ke změně λ_h ani λ_d po dobu 10-ti iterací, je jako optimální hodnota označena λ_h a lepší nejde dosáhnout. Tímto je zabráněno nekonečnému cyklu. Tabulka A.1 ukazuje, jak se mění závislost doby výpočtu na zvoleném ϵ.

Příloha B

Obsah přiloženého CD

1. Text této bakalářské práce.
2. Projekt Microstoft Visual Studia 2010 s jednoduchou metodou předvádějící program.
3. Vygenerovaná dokumentace k projektu.
4. Popis, jak projekt zprovoznit.