Automatizované řešení úloh s omezeními

Martin Kot

Katedra informatiky, FEI,
Vysoká škola báňská – Technická universita Ostrava
17. listopadu 15, Ostrava-Poruba 708 33
Česká republika

18. září 2013

Přednášející

Garant předmětu:

Jméno: prof. RNDr. Petr Jančar, CSc.

E-mail: petr.jancar@vsb.cz

Místnost: A1046

Přednášející a cvičící:

Jméno: Ing. Martin Kot, Ph.D.

E-mail: martin.kot@vsb.cz

Místnost: A1024

WWW stránky

Webové stránky k předmětu naleznete na adrese:

http://www.cs.vsb.cz/kot/aruo

Na těchto stránkách najdete:

- Informace o předmětu
- Slidy z přednášek
- Aktuální informace

Požadavky

Klasifikovaný zápočet (100 bodů):

- Projekt (60 bodů)
- Zápočtová písemka (40 bodů)

Pro získání zápočtu je potřeba získat alespoň 31 bodů z projektu a alespoň 20 bodů ze zápočtové písemky.

Výukové materiály

Hlavními výukovými texty jsou:

- Slajdy
- Rina Dechter
  Constraint Processing,
  Morgan Kaufmann Publishers, 2003.
- Slajdy autorky knihy

Úvod

Úvod

- Úlohy s omezeními řešíme v běžném životě
- Obvykle je řešíme intuitivně, bez počítače
- Např. hospodaření s penězi, sestavování jídelníčku, ...
- S rostoucí složitostí úloh se obracíme na počítače
- V obecnosti je většina úloh s omezeními "výpočetně nezvládnutelná" (úlohy bývají NP-těžké)
- Nemůžeme počítat s efektivními algoritmy řešícími úlohy v plném rozsahu
- Můžeme navrhnout řešení efektivní pro většinu instancí

Základní koncept

- **Proměnné** (variables) - mohou nabývat různé hodnoty
- **Doména, definiční obor** (domain) - množina možných hodnot pro danou proměnnou
- **Omezení** (constraints) - pravidla kladoucí podmínky na možné ohodnocení proměnných a jejich kombinací
- Často je více možných modelů problému (zasedací pořádek hostů na svatbě - hosté jako proměnné nebo židle jako proměnné)
- **Síť omezení** (constraint network), **problém s omezeními** (constraint problém) - model zahrnující proměnné, jejich domény a omezení
- Pozn. pojem síť omezení vychází z historie, kdy se výzkum zaměřoval na typy problémů, kde závislosti byly popsatelné jednoduchými grafy
- **Řešení** (solution) - přiřazení jedné hodnoty z domény pro každou proměnnou tak, aby nebyla porušena žádná omezení
- Řešení může být jedno nebo více, ale taky žádné
- **Splnitelný** (satisfiable), **konzistentní** (consistent) - má řešení
- **Nesplnitelný** (unsatisfiable), **nekonzistentní** (inconsistent) - nemá řešení

Typické úlohy

U problémů s omezeními nás obvykle zajímají tyto úlohy:

- určit, zda existuje řešení (splňující všechna omezení)
- najít jedno řešení
- zjistit jestli přiřazení několika hodnot proměnných může být rozšířeno na řešení
- najít optimální řešení vzhledem k cenové funkci (cost function)

Všechny tyto úlohy souhrně označujeme jako constraint satisfaction problems (CSP)

Příklad – n dam na šachovnici

- Úkolem je umístit n dam na šachovnici $n \times n$ tak, aby se vzájemně neohrožovaly
- Model např.:
  - proměnné $x_1, \ldots, x_n$ pro sloupce
  - doména D_i každé proměnné je $D_i = \{1, \ldots, n\}$ (číslo řádku)
  - omezení pro každý pár sloupců zakazují sdílení řádku nebo diagonály

Příklad - křížovky

- Úkolem je ze slovníku přiřadit slova buňkám horziontálně a vertikálně
- Každé obsaditelné políčko je proměnná
- Doménou každé proměnné je abeceda
- Omezení jsou dána slovníkem
- Slidy Riny Dechter - 1. kapitola, slide 3

Příklad - barvení map

- Úkolem je obarvit všechny státy politické mapy 4 barvami tak, aby sousední státy (s nenulovou délkou hranice) neměly stejnou barvu
- Od roku 1976 dokázáno, že to vždy jde (Appel, Haken)
- Abstrakce grafem - vrchol pro každou zemi, hrana při společné hranici
- Vrcholy jsou proměnné, domény obsahují barvy, omezením je zákaz stejné barvy pro sousední vrcholy
- Zobecněním je (vrcholové) barvení grafu k barvami
- Mnoho praktických problémů se dá převést na barvení grafu (např. přidělování radiových frekvencí)
- Slidy Riny Dechter - 1. kapitola, slide 4

Příklad - konfigurace, návrh

- Celá řada problémů hledající optimální návrh nebo konfiguraci při splnění nějakých podmínek
- Např. návrh automobilů, konfigurace počítačů, rozložení místností na patře budovy, ...
- Slidy Riny Dechter - 1. kapitola, slidy 5-7

Příklad - Huffmanovo-Clowesovo ohodnocení scény

- Jeden z prvních problémů s omezeními (definován 1975)
- Cílem je rozpoznání 3D objektů ve scéně prostřednictvím interpretace čar v 2D kresbě
- Úsečkám přiřazujeme ohodnocení ($+$ konvexní, $-$ konkávní, $<$ okrajové) u krajních bodů
- Je jen omezený počet možných přiřazení více úsečkám se společným krajním bodem
- Omezení jsou dána tím, že úsečka musí mít stejné ohodnocení na obou stranách
- Slidy Riny Dechter - 1. kapitola, slidy 8-9

Příklad - plánování

- Plánování (scheduling) zahrnuje mnoho různých problémů
- Příkladem je např. plánování výroby v dílně - omezený počet strojů a lidí, snaha vyrobit co nejvíce, stroje a lidé by neměli zahálet
- Patří sem i tvorba rozvrhů - učitelé, studenti, místnosti, předměty, ...
- Často jsou úlohy voleny jako proměnné a domény obsahují časy (začátku vykonávání úkolu)

Inference

- **Inference** - snaha vytvořit jednodušší problém díky jeho přeformulování
- Někdy inferenční metody najdou řešení nebo zjistí nekonzistentnost
- **Lokální konzistentnost** (local consistency) - díváme se na podčásti problému a požadujeme, aby tam nebyly sporné části
- Např. při $x_1, \ldots, x_n$, $D_i = \{1, \ldots, 10\}$ a omezení, že x_1 je ostře větší, než ostatní, nepotřebujeme v D_1 nechat 1.
- Algoritmy pro lokální konzistentnost se také nazývají algoritmy **šíření omezení** (constraint propagation)
- Většina algoritmů šíření omezení je polynomiální a převádějí síť na jinou, ekvivalentní ale explicitnější, odvozením nových omezení a jejich přidáním k problému

Inference

- **Hranová konzistentnost** (arc-consistency) - každé možné přiřazení hodnoty proměnné (z její domény) má možné přiřazení z domény jiné vybrané proměnné
- **Konzistentnost cesty** (path-consistency) - každé přiřazení hodnoty 2 proměnným (z jejich domén) je rozšiřitelné na třetí proměnnou
- **i-konzistentnost** (i-consistency) - každé přiřazení hodnoty $i - 1$ proměnným (z jejich domén) je rozšiřitelné na $i$tou proměnnou
- Vynucení i-konzistentnosti je obvykle výpočetně náročné - exponenciální časově i prostorově vzhledem k i
- Často může dojít k dokázání nekonzistentnosti sítě vyprázdněním některé z domén při zajišťování např. i-konzistence

Prohledávání

- Nejčastěji se používá backtracking
- Prochází se prostor řešení do hloubky
- Každý krok je přiřazení hodnoty jedné proměnné z její domény
- Když není možné (konzistentně) přiřadit žádnou hodnotu, nastává tzv. **dead-end**
- Dead-end se řeší backtrackingem - vrátíme se k předchozí proměnné a nastavíme ji jinou hodnotu

Možná vylepšení backtrackingu

- Vylepšení dělíme na ta pro dopředný pohyb (look-ahead schemes) a pro zpětný pohyb (look-back schemes)
- Dopředné:
  - Snaha přiřazovat hodnoty proměnným tak, aby byla co největší šance na úspěch nebo rychlé zjištění nekonzistentnosti sítě
  - Nejprve přiřazujeme proměnným, kterých se týká hodně omezení
  - Při výběru hodnoty pro proměnnou vybereme nejméně omezenou hodnotu
- Zpětné:
  - Řeší např. jak daleko se vrátit (backjumping)
  - Analýzují se příčiny vzniklého dead-endu
  - Zaznamenávají se příčiny dead-endu ve formě nových omezení, aby nenastal ze stejného důvodu znovu (constraint learning, no-good recording)

Stochastické lokální hledání

- Algoritmus vylepšuje instanciaci změnou hodnot proměnných tak, aby maximalizoval počet splněných omezení
- Neúplné algoritmy - můžou se zaseknout na lokálním minimu cenové funkce, nedokáží nekonzistentnost sítě
- Spolu s heuristikami se ukazují užitečné v praxi pro velké a těžké problémy

Sítě omezení

Síť omezení

- **Síť omezení** (constraint network) se skládá množiny proměnných $X = \{x_1, \ldots, x_n\}$ s vlastními doménami $D = \{D_1, \ldots, D_n\}$ (obsahujícími možné hodnoty pro každou proměnnou $D_i = \{v_1, \ldots, v_k\}$) a množiny omezení $C = \{C_1, \ldots, C_t\}$. Formálně tedy jde o trojici $\mathcal{R} = (X, D, C)$.
- Omezení C_i je relace R_i definovaná na podmnožině proměnných $S_i, S_i \subseteq X$.
- S_i se nazývá **rozsah** (scope) relace R_i
- Relace určují vzájemné povolené přiřazení hodnot proměnným
- Pro $S_i = \{x_{i_1}, \ldots, x_{i_r}\}$ je $R_i \subseteq D_{i_1} \times \ldots \times D_{i_r}$
- Omezení tedy můžeme chápat jako dvojici $C_i = (S_i, R_i)$
- Pozn. - Při jasném rozsahu používáme jen R_i nebo rozsah uvedeme v indexu R_{S_i}, často taky bez závorek R_{xyz}
- Množina rozsahů $S = \{S_1, \ldots, S_t\}$ - **schéma sítě** (network scheme)

Síť omezení

- Předpokládá se jen jedno omezení nad každým $S_i \in S$ (při relačních omezeních)
- Arita omezení = kardinalita rozsahu
- Binární síť omezení (binary constraint network) - má pouze unární a binární omezení
- Příklad - formalizace n dam na šachovnici
  - Slidy Riny Dechter, chapter 2, slides 2-3
  - Sloupce jsou proměnné $x_1, \ldots, x_n$
  - Možné řádky jsou domény $D_i = \{1, \ldots, n\}$
  - Omezení jsou binární - vyjadřují, že dámy se nesmí ohrožovat vyjmenováním povolených dvojic řádků pro dvojice sloupců
  - Pro 4 dámy: $\mathcal{R} = (X, D, C)$, $X = \{x_1, x_2, x_3, x_4\}$, $\forall i D_i = \{1, 2, 3, 4\}$, $C_1 = R_{12}$, $C_2 = R_{13}$, $C_3 = R_{14}$, $C_4 = R_{23}$, $C_5 = R_{24}$, $C_6 = R_{34}$

Instanciace sítě

- **Instanciace** (instantiation) **podmnožiny proměnných** - celé podmnožině proměnných se přiřadí hodnoty z příslušných domén
- Formálně je instanciace množiny $\{x_{i_1}, \ldots, x_{i_k}\}$ $n$tice dvojic $((x_{i_1}, a_{i_1}), \ldots, (x_{i_k}, a_{i_k}))$, kde (x, a) reprezentuje přiřazení $a \in D_x$ proměnné x
- Pozn. Používá se taky $(x_1 = a_1, \ldots, x_i = a_i)$ nebo $\bar{a} = (a_1, \ldots, a_i)$

Splnění omezení

- Instanciace $\bar{a}$ **splňuje omezení** (S, R) právě tehdy, když je definována nad všemi proměnnými z S a prvky $\bar{a}$ asociované s S jsou v relaci R
- Příklad: uvažujme $R_{xyz} = \{(1,1,1),(1,0,1),(0,0,0)\}$. Potom
  $\bar{a} = ((x,1),(y,1),(z,1),(t,0))$ splňuje R_{xyz} ale
  $\bar{a} = ((x,1),(y,0),(z,0),(t,0))$ ne, protože $(1,0,0) \notin R_{xyz}$.

Konzistentní částečná instanciace

- Částečná instanciace je **konzistentní**, když splňuje omezení, v jejichž rozsahu není žádná neinstanciovaná proměnná
- Projekci $\bar{a}$ na podmnožinu rozsahu S_i označujeme $\bar{a}[S_i]$ nebo $\pi_{S_i}(\bar{a})$

Řešení

- **Řešení** (solution) sítě $\mathcal{R} = (X, D, C)$, kde $X = \{x_1, \ldots, x_n\}$ je instanciace všech proměnných, která splňuje všechna omezení
- **Relace řešení** $sol(\mathcal{R})$ (ozn. také ρ_X) je definována jako $sol(\mathcal{R}) = \{a = (a_1, \ldots, a_n) \mid a_i \in D_i, \forall S_i$ ve schématu R je $\bar{a}[S_i] \in R_i\}$
- Říkáme, že síť omezení **vyjadřuje** (express) nebo **reprezentuje** (represents) relaci všech svých řešení.
- Pro síť $\mathcal{R}$ nad X a $A \subseteq X$ je $sol(A)$ nebo ρ_A množina konzistentních instanciací nad A
- Příklad: 4 dámy na šachovnici (slidy Riny Dechter, chapter 2, slide 3) - Uvažujme $Y = \{x_1, x_2, x_3\}$, instanciaci $\bar{a} = ((x_1, 1), (x_2, 4), (x_3, 2))$ na obr. (a). Ta je konzistentní, protože $\bar{a}[\{x_1, x_2\}] = (1, 4)$ a $(1, 4) \in R_{12}$ atd. Ale $\bar{a}$ není součástí žádného řešení. 4 dámy na šachovnici reprezentují $\rho_{1234} = \{(2, 4, 1, 3), (3, 1, 4, 2)\}$.

Graf omezení

- Jedna z reprezentací sítě omezení je prostřednictvím **grafu omezení** ((primal) constraint graph)
- Uzly reprezentují proměnné, hrany spojují proměnné které jsou svázány některým z omezení
- Pro binární omezení - chybějící hrana znamená úplnou relaci mezi příslušnými proměnnými
- Dává smysl pro binární i jiná omezení

Hypergraf, duální graf

- **Hypergraf** je struktura $\mathcal{H} = (V, S)$, V je množina vrcholů, $S = \{S_1, \ldots, S_l\}$ množina podmnožin množiny vrcholů ($S_i \subseteq V$) nazývaných hyperhrany
- V hypergrafu omezení opět vrcholy reprezentují proměnné a hyperhrany (kreslené jako oblasti)
- Je přesnější reprezentací nebinárních omezení
- **Duální graf omezení**
  - Hyperhrana původního grafu (tedy rozsah omezení) je reprezentována vrcholem
  - Hrany spojují vrcholy s neprázdným průnikem hyperhran (tedy sdílející některou z proměnných)
  - Hrany jsou ohodnoceny sdílenou proměnnou

Duální problém

- **Duální problém** (dual problem) vychází z duálního grafu
- Omezení tvoří proměnné nazývané **c-proměnné** (c-variables)
- Domény proměnných jsou tvořeny možnými kombinacemi hodnot vynucenými odpovídajícími omezeními
- Omezení mezi c-proměnnými vynucují, že jimi sdíleným (původním) proměnným musí být přiřazené stejné hodnoty
- Jakoukoliv síť omezení takto je možno převést na binární a řešit ji technikami pro binární

Numerická omezení

- **Numerická omezení** (numeric constraints) vyjadřují omezení aritmetickými výrazy
- Příklad: Pro dámy na šachovnici můžeme dát omezení $\forall i, j : x_i \neq x_j \wedge |x_i - x_j| \neq |i - j|$
- Příklad: Můžeme uvažovat proměnné s doménami, které jsou podmnožiny přirozených čísel. Omezení ve tvaru konjunkce lineárních nerovnic $ax_i - bx_j = c$, $ax_i - bx_j < c$, $ax_i - bx_j \leq c$. Jde o speciální případ tzv. celočíselného lineárního programu.
- Příklad: SEND + MORE = MONEY z prvního cvičení

Booleovská omezení

- Když jsou proměnné dvouhodnotové, využívá se často jazyk výrokové (nebo také booleovské) logiky
- Výrokové proměnné nabývají hodnot $\{true, false\}$ nebo $\{1, 0\}$ a označujeme je $P, Q, R, \ldots$
- Literály jsou P (P je true) nebo $\neg P$ (P je false)
- Klauzule - disjunkce literálů
- Operace rezoluce mezi $\alpha \vee Q$ a $\beta \vee \neg Q$ dává $\alpha \vee \beta$
- Konjunktivní normální forma (také označována jako teorie) - množina klauzulí označující jejich konjunkci
- Model nebo řešení teorie ϕ je přiřazení pravdivostních hodnot proměnným tak, aby všechny klauzule byly pravdivé
- SAT - má formule model? (je příslušný problém konzistentní?)

Binární sítě omezení

- Základním konceptem práce s omezeními je vyvozování nových omezení
- Nová omezení můžou být zpřísněním původních nebo mohou omezovat proměnné, které dříve omezené nebyly
- Příklad: Z $x \geq y$, $y \geq z$ lze vyvodit $x \geq z$
- Odvozené omezení je redundantní vzhledem k síti - jeho odstranění neovlivní množinu řešení
- **Skládání** (composition): pro dané binární (popř. unární) omezení R_{xy}, R_{yz} složení $R_{xy}R_{yz}$ generuje binární relaci $R_{xz} = \{(a, b) \mid a \in D_x, b \in D_z, \exists c \in D_y$ tž. $(a, c) \in R_{xy}$ a $(c, b) \in R_{yz}\}$
- Skládání lze popsat také jako násobení booleovských matic

Projekční síť

- Ne každá obecná relace může být reprezentována sadou binárních omezení (relací s n proměnnými a k hodnotami v doménách je 2^{k^n}, binárních sítí omezení jen $2^{k^2 n^2}$)
- Ale můžeme takovou relaci aproximovat binárně, například **projekční sítí** (projection network)
- Projekční síť relace ρ je projekce ρ na každou dvojici jejích proměnných
- Formálně, když ρ je relace nad $X = \{x_1, \ldots, x_n\}$, její projekční síť $P(\rho)$ je definována jako $\mathcal{P} = (X, D, P)$ kde $D = \{D_i \mid 1 \leq i \leq n\}$, $D_i = \pi_i(\rho)$,$P = \{P_{ij} \mid 1 \leq i, j \leq n\}$, $P_{ij} = \pi_{x_i, x_j}(\rho)$
- Řešení sítě $P(\rho)$ vždy obsahuje ρ
- Neexistuje síť binární síť $\mathcal{R}'$ taková, že $\rho \subseteq sol(R') \subset sol(P(\rho))$
- Když relace nemůže být reprezentována projekční sítí, nemůže být reprezentována žádnou binární sítí

Minimální síť

- Pro dvě binární sítě $\mathcal{R}, \mathcal{R}'$ na stejné množině proměnných $x_1, \ldots, x_n$ říkáme, že $\mathcal{R}'$ je alespoň tak striktní (tight) jako $\mathcal{R}$ právě tehdy, když $\forall i, j : R'_{ij} \subseteq R_{ij}$
- Průnik sítí $\mathcal{R}, \mathcal{R}'$, označovaný $\mathcal{R} \cap \mathcal{R}'$ je binární síť získaná vzájemným průnikem všech vzájemně si odpovídajících omezení
- Když jsou sítě $\mathcal{R}, \mathcal{R}'$ ekvivalentní, je s nimi ekvivalentní i $\mathcal{R} \cap \mathcal{R}'$ a je minimálně tak striktní jako obě tyto sítě
- Existuje částečné uspořádání sítí podle striktnosti
- **Minimální síť** je průnikem všech ekvivalentních sítí
- Minimální síť je ekvivalentní se sítěmi, ze kterých průnikem vznikla a je alespoň tak striktní, jako každá z nich
- Formálně: Nechť $\{\mathcal{R}_1, \ldots, \mathcal{R}_l\}$ je množina všech sítí ekvivalentních s R_0 a nechť $\rho = sol(\mathcal{R}_0)$. Potom minimální síť M pro $\mathcal{R}_0$ nebo ρ je definována jako $M(R_0) = M(\rho) = \bigcap_{i=1}^{l} R_i$

Minimální síť

- Minimální síť je identická s projekční sítí množiny řešení minimální sítě. Tedy pro každou binární síť $\mathcal{R}$ tž. $\rho = sol(\mathcal{R})$ platí $M(\rho) = P(\rho)$
- Binární omezení v minimální síti označujeme M_{ij}
- Unární omezení jsou redukované domény
- Když nějaká dvojice hodnot proměnných vyhovuje omezení v minimální síti, potom se vyskytuje v alespoň jednom řešení

Šíření omezení

Šíření omezení (constraint propagation)

- Můžeme dokázat nekonzistentnost sítě
- Můžeme vyloučit některé kombinace přiřazení hodnot proměnným a tím urychlit výpočet
- Můžeme i najít řešení - nová omezení způsobí, že každé řešení se najde prostým prohledáváním do hloubky, kde nenastane nikdy dead-end
- Většinou ale řešení problému vyvozováním (inference) je výpočetně náročné, vyžaduje přidání exponenciálního množství nových omezení
- Obvykle se přidávaji jen nějaká omezení, prohledávání pak může narazit na dead-end, ale končí relativně rychle
- Algoritmy přidávající omezený počet omezení označujeme vynucení lokální konzistentnosti (local consistency-enforcing), omezené vyvozování konzistence (bounded consistency inference) nebo šíření omezení (constraint propagation)

Hranová konzistentnost

- V minimální síti platí, že přiřazení jakékoliv hodnoty z domény proměnné je rozšiřitelné na jakoukoliv další proměnnou
- Takovou vlastnost nazýváme **hranová konzistentnost** (arc-consistency)
- Může být splněna také v neminimálních sítích
- Může být vynucena na každé síti efektivním výpočtem, často nazývaným **propagace** (propagation)
- Definice: Pro danou síť $\mathcal{R} = \{X, D, C\}$ s $R_{ij} \in C$ je proměnná x_i **hranově konzistentní** (arc-consistent) vzhledem k x_j právě tehdy, když pro každou hodnotu $a_i \in D_i$ existuje hodnota $a_j \in D_j$ tž. $(a_i, a_j) \in R_{ij}$.
- Definice: Podsíť (nebo hrana) definovaná $\{x_i, x_j\}$ je hranově konzistentní, právě když x_i je hranově konzistentní vzhledem k x_j a x_j je hranově konzistentní vzhledem k x_i
- Definice: Síť omezení je hranově konzistentní, právě když všechny její hrany (podsítě velikosti 2) jsou hranově konzistentní

Revidující procedura

- Slidy R. Dechter, chapter 3, slide 7
- Převádí síť na hranově konzistentní redukcí domén proměnných v rozsahu omezení
- Aplikována na x_i, x_j vrací největší doménu D_i pro kterou x_i je hranově konzistentní vzhledem k x_j
- Složitost v nejhorším případě je $O(k^2)$, kde k je mez velikosti domén

Algoritmus ARC-CONSISTENCY-1 (AC-1)

- Slidy R. Dechter, chapter 3, slide 9
- Brute-force algoritmus zajišťující hranovou konzistentnost sítě
- Aplikuje revidující proceduru na všechny dvojice proměnných, které se vyskytují v omezeních dokud se nějaký doména mění
- Může zjistit nekonzistentnost sítě
- Složitost v nejhorším případě je $O(enk^3)$, kde n je počet proměnných, k je mez velikostí domén, e je počet omezení

Algoritmus ARC-CONSISTENCY-3 (AC-3)

- Slidy R. Dechter, chapter 3, slide 10
- Algoritmus AC-1 je možné vylepšit
- Když se změní jen pár domén, není potřeba procházet všechna omezení znovu
- Stačí udržovat frontu dvojic proměnných, které ještě zpracovány nebyly nebo v doméně jedné z nich došlo ke změně
- Algoritmus označujeme ARC-CONSISTENCY-3 (AC-3), AC-2 byl mezikrok
- Složitost je $O(ek^3)$

Algoritmus ARC-CONSISTENCY-4 (AC-4)

- Slidy R. Dechter, chapter 3, slide 12
- Algoritmus AC-3 stále není optimální (z hlediska časové složitosti)
- Pouhé ověření konzistentnosti sítě potřebuje ek^2 operací, zajištění konzistentnosti asi nemůže jít rychleji
- Algoritmus AC-4 dosahuje tuto mez
- Nevyužívá revidující proceduru, zkoumá strukturu relace omezení
- Každé hodnotě a_i z domény x_i přiřadí počet konzistentních hodnot z x_j
- Hodnota a_i je odebrána z domény, pokud nemá v některé ze sousedních proměnných odpovídající protějšek
- Složitost je $O(ek^2)$

Složitost

- Místo k^2 vzniklé z univerzální relace můžeme uvažovat t pro maximální počet dvojic v relacích
- Revidující procedura potom má $O(t)$
- AC-1: $O(nket)$
- AC-3: $O(ekt)$
- AC-4: $O(et)$
- Složitost v nejlepším případě pro AC-1 a AC-3 je ek, když už problém je hranově konzistentní. Pro AC-4 je to ek^2, protože tolik trvá už inicializace
- Při slabých omezeních v praxi často AC-1 a AC-3 jsou rychlejší než AC-4

Konzistentnost cesty

- Definice: Pro danou síť $\mathcal{R} = \{X, D, C\}$ je množina dvou proměnných $\{x_i, x_j\}$ **konzistentní po cestě** (path-consistent) vzhledem k x_k právě tehdy, když pro každé přiřazení $(\langle x_i, a_i\rangle, \langle x_j, a_j\rangle)$ existuje hodnota $a_k \in D_k$ tž. přiřazení $(\langle x_i, a_i\rangle, \langle x_k, a_k\rangle)$ a $(\langle x_k, a_k\rangle, \langle x_j, a_j\rangle)$ jsou konzistentní.
- Definice (alternativa): Pro danou síť $\mathcal{R} = \{X, D, C\}$ je binární omezení R_{ij} je konzistentní cestou vzhledem k x_k x_j právě tehdy, když pro každý pár $(a_i, a_j) \in R_{ij}$, kde a_i, a_j jsou z příslušných domén, existuje hodnota $a_k \in D_k$ tž. $(a_i, a_k) \in R_{ik}$ a $(a_k, a_j) \in R_{kj}$.
- Definice: Podsíť nad třemi proměnnými $\{x_i, x_j, x_k\}$ je konzistentní po cestách jestliže pro každou permutaci (i, j, k) je R_{ij} konzistentní po cestě vzhledem k x_k
- Definice: Síť omezení je konzistentní po cestách, právě když pro všechny R_{ij} a každé $k \neq i, j$ je R_{ij} konzistentní po cestě vzhledem k x_k

Revidující procedura REVISE-3

- Slidy R. Dechter, chapter 3, slide 17
- Bere dvojice proměnných (x, y) a jejich omezení R_{xy} a třetí proměnnou z a vrací nejslabší omezení R'_{xy} splňující konzistentnost po cestě
- Testuje se každá dvojice konzistentních hodnot R_{xy}, jestli je možné ji rozšířit na hodnotu z. Když ne, dvojici smaže.
- Složitost je $O(k^3)$ nebo $O(tk)$

Algoritmus PATH-CONSISTENCY-1 (PC-1)

- Slidy R. Dechter, chapter 3, slide 18
- Vynutí konzistentnost po cestách pro všechny podsítě velikosti 3, výsledkem je síť konzistentní po cestách
- Je analogií AC-1
- Složitost je $O(n^5k^5)$ nebo $O(n^5t^2k)$

Algoritmus PATH-CONSISTENCY-2 (PC-2)

- Slidy R. Dechter, chapter 3, slide 19
- Vylepšuje PC-1 udržováním fronty uspořádáných trojic ke zpracování
- Když je omezení R_ij změněno vymazáním některé dvojice hodnot, všechny trojice zahrnující x_i a x_j a s nimi nějakou třetí hodnotu x_k jsou znovu zpracovány
- Je analogií AC-3
- Složitost je $O(n^3k^5)$ nebo $O(n^3t^2k)$
- Také není optimální z hlediska časové složitosti, existuje optimální algoritmus PC-4 se složitostí $O(n^3k^3)$ nebo $O(n^3tk)$

i-konzistentnost

- Definice: Pro síť $\mathcal{R} = (X, D, C)$ je relace $R_S \in C$, kde $|S| = i - 1$, **i-konzistentní** (i-consistent) vzhledem k proměnné $y \notin S$ právě tehdy, když pro každé $t \in R_S$ existuje hodnota $a \in D_y$ tž. (t, a).
- Definice: Síť je **i-konzistentní** (i-consistent) když pro jakoukoliv instanciaci $i - 1$ různých proměnných existuje instanciace každé $i$té proměnné tž. získaných i hodnot dohromady plňuje všechna omezení na těchto proměnných
- Revidující procedura REVISE-i – slidy R. Dechter, chapter 3, slide 23
- Algoritmus I-CONSISTENCY – slidy R. Dechter, chapter 3, slide 23
- Algoritmus je exponenciální vzhledem k i, tedy při zajišťování globální konzistentnosti (i je počet všech proměnných v síti) exponenciální vzhledem k velikosti sítě
- Zavádí do sítě omezení na $i - 1$ proměnných, z binární sítě tak může udělat nebinární

Zobecněná hranová konzistentnost

- Jedna ze dvou možností rozšíření hranové konzistentnosi na nebinární sítě
- Definice: Pro síť $\mathcal{R} = (X, D, C)$ s relací $R_S \in C$, je proměnná x **hranově konzistentní** vzhledem k R_S právě tehdy, když pro každé $a \in D_x$ existuje n-tice $t \in R_S$ tž. $t[x] = a$
- Definice: Omezení R_S je hranově konzistentní právě tehdy, když je hranově konzistentní vzhledem ke každé proměnné ve svém rozsahu
- Zajištění konzistentnosti redukuje doménu proměnné
- Druhý přístup je **relační hranová konzistentnost**, zaznamenává odvozené omezení ve formě změnou některé z relací reprezentujících omezení

Globální omezení

- Modelování reálných aplikací ve formě problému s omezeními vyžaduje specializované algoritmy šíření omezení pro často využívaná omezení
- alldifferent
  - Požaduje se, aby všechny proměnné měly přiřazeny vzájemně různé hodnoty
  - Je možné popsat binárními omezeními
  - Aplikování hranové konzistentnosti většinou nic nepřinese, protože síť už je konzistentní (pokud nejsou jednoprvkové domény)
- Typicky dávají smysl nad libovolným rozsahem
- Další příklady: omezení součtem (jedna proměnná je součtem jiných), kumulativní omezení (kumulativní spotřeba zdrojů v čase nepřekračuje specifikovanou kapacitu)

Konzistentnost hranic

- Na velkých doménách je "drahé" zajistit hranovou konzistentnost
- Levnější je zajistit **konzistentnost hranic** (bounds-consistency)
- Idea je omezit domény intervalem a zajistit, že krajní body těchto intervalů splňují hranovou konzistentnost
- Slidy R. Dechter, chapter 3, slide 31
- Zajištění konzistentnosti hranic při omezení typu alldifferent je možné v čase $O(n \log n)$

Dopředné hledání

Search

- I při uplatnění různých algoritmů pro zajištění konzistentnosti sítě apod. často nakonec nezbyde nic jiného, než použít metodu "pokus – omyl"
- Jedná se o prohledávání prostoru možných řešení
- Často se prohledávání kombinuje s různými způsoby dedukce
- Pojem prohledávání (search) v oblasti CSP zahrnuje mnoho algoritmů, které řeši problémy "hádáním" dalšího kroku, někdy v kombinaci s nějakými heuristikami
- V CSP je prohledávání téměř výhradně spjato s variantami backtrackingu

Backtracking

- Aktuální částečné řešení se vždy rozšíří přiřazením nějakých hodnot dalším proměnným
- Začíná se s "nějak zvolenou" první proměnnou
- Po jedné se dalším proměnným přiřazují provizorní hodnoty a kontroluje se, jestli právě přiřazená hodnota je konzistentní vzhledem k dříve přiřazeným
- Dead-end - stav, kdy žádná hodnota v doméně právě přiřazované proměnné není konzistentní s dříve přiřazenými hodnotami
- Backtracking - po dead-end situaci změníme hodnotu přiřazenou bezprostředně před tou, která k dead-end situaci vedla
- Algoritmus končí nalezením požadovaného počtu řešení, nebo zjištěním, že žádné řešení neexistuje nebo že již byla nalezena všechna řešení
- Výhodou je polynomiální prostorová složitost, nevýhodou exponenciální časová složitost

Backtracking

- Hodně výzkumu se zabývalo backtrackingem
- Byla snaha zmenšit prohledávaný prostor snížením vlastního prostoru řešení nebo efektivnějšími způsoby procházení
- Velikost prostoru řešení závisí hlavně na stupni konzistentnosti sítě a pořadí proměnných při přiřazování hodnot
- Vznikly dva druhy procedur
  - spouštěné před prohledáváním - zmenšení prostoru řešení
  - spouštěné během prohledávání - rozhodnutí, které části prostoru řešení algoritmus vynechá

Stavový prostor

- **Stavový prostor** (state space) je dán 4 prvky:
  - množinou stavů S
  - množinou operátorů O, přiřazují stavy stavům
  - počátečním stavem $s_0 \in S$
  - množinou cílových stavů $S_g \subseteq S$
- Základní cíl je nalezení řešení - sekvence operátorů, které převedou počáteční stav do cílového stavu
- Stavový prostor lze efektivně reprezentovat orientovaným grafem
- **Prohledávací graf** (search graph)
  - uzly reprezentují stavy
  - orientovaná hrana z s_i do s_j znamená, že existuje operátor transformující s_i na s_j
  - koncové uzly (listy) můžou být cílové (reprezentují řešení) a necílové (reprezentují dead-end)
- Prohledávací algoritmy jsou vlastně algoritmy procházející tímto grafem

Prohledávání

- Věta: Nechť R' je striktnější síť než R, kde obě reprezentují stejnou množinu řešení. Pro jakékoliv uspořádání proměnných platí, že každá cesta vyskytující se v prohledávacím grafu vytvořeného z R' se vyskytuje také v grafu vytvořeného z R
- Nabízí se tedy možnost zvážit uplatnění vynucení konzistentnosti před prohledáváním
- Ne vždy je čas využitý na zajišťování konzistence vyvážen úsporou času při prohledávání
- Přidání nových omezení při zajištování vyšších stupňů konzistence také může značně spomalit prohledávání - po přiřazení hodnoty každé proměnné je potřeba kontrolovat splnění těchto omezení
- Pro binární omezení nikdy nebude více než $O(n)$ operací ověřujících konzistenci pro každý stav
- Při obecných omezeních můžeme mít až $O(n^{r-1})$ omezení, kde r je mez arity těchto omezení
- Backtrack-free síť pro uspořádání proměnných d je síť jejíž prohledávací graf má jen cílové listy (tedy každý list odpovídá

Backtracking

- Prochází prohledávací strom do hloubky
- Má dvě fáze:
  - Dopředná - je vybrána další proměnná a aktuální částečné řešení je rozšířeno přiřazením konzistentní hodnoty vybrané proměnné
  - Zpětná - když neexistuje konzistentní hodnota pro aktuálně přiřazovanou proměnnou, vrací se algoritmus k proměnné přiřazované v minulém kroku
- Algoritmus - slidy R. Dechter, chapter 5, slide 12

Složitost rozšíření aktuálního částečného řešení

- Uvažujme omezení uložená v tabulkách
- e je počet omezení, t je maximální počet n-tic v omezení, k maximální velikost domény, r je maximální arita omezení (tedy $t \leq k^r$))
- Omezení je možné uložit tak, aby bylo možné nalézt konkrétní n-tici v logaritmickém čase $\log t \leq r \log k \leq n \log k$
- Protože proměnná může být v rozsahu až e omezení, časová složitost v nejhorším případě procedury CONSISTENT je $O(e \log t)$ (nebo také $O(er \log k)$)
- Procedura SELECT-VALUE může volat CONSISTENT až $k$krát, čili složitost je $O(ek \log t)$ (nebo také $O(ekr \log k)$)
- V binárních sítích se aktuálně přiřazená hodnota ověřuje maximálně vůči n dříve přiřazeným proměnným, tedy CONSISTENT má složitost $O(n)$ a SELECT-VALUE má $O(nk)$
- CONSISTENT dělá i jiné operace než vyhledávání ntic omezení v tabulce. K těmto operacím je nutno přihlédnout, pokud můžou ovlivnit asymptotickou složitost

Vylepšení backtrackingu

- trashing - opakované objevování stejných nekonzistentností a částečných řešení během hledání
- Efektivní řešení problému trashing je nepravděpodobné – jde o NP-úplný problém
- Je ale možné snížit množství trashingu a tím vylepšit výkon prohledávací procedury
- Vylepšení dělíme na dvě skupiny
  - pro dopředný pohyb (look-ahead schemes)
    - Volají se při přípravě na přiřazení hodnoty další proměnné
    - Snaží se zjistit jak přiřazení ovlivní další prohledávání
    - Použitá inference může vést k výběru proměnné k přiřazení nebo k výběru hodnoty pro některou proměnnou
  - pro zpětný pohyb (look-back schemes)
    - Volají se po dead-endu
    - Rozhodují, jak daleko se vrátit analýzou příčin dead-endu (až k příčině problému) - backjumping
    - Můžou zachytit příčinu dead-endu do nového omezení - ukládání omezení (constraint recording), učení omezení (constraint learning)

Strategie pro dopředný pohyb

- Zvyšují čas potřebný pro instanciaci proměnné, ale můžou přinést rychlejší nalezení řešení
- Například můžeme zjistit, že žádná z hodnot aktuálně přiřaditelných by nebyla konzistentní s následujícími a rovnou provést backtrack
- Čím více šíření omezení se použije pro pohled dopředu, tím menší prostor bude potřeba prohledávat, ale tím více času navíc ztratíme
- Je možné při pohledu dopředu proměnným dosud neinstanciovaným díky testům konzistentnosti zmenšit domény a při jejich instanciaci již potom nebude potřeba kontrolovat konzistentnost
- Málokdy se zlepší asymptotická složitost v nejhorším případě
- Klíčové je správné vyvážení mezi režií vnesenou přidanými algoritmy a jejich efektivností
- Algoritmus GENERALIZED-LOOK-AHEAD (slidy R. Dechter, chapter 5, slide 15) - obecný framework pro všechna vylepšení dopředného pohybu, změny strategie se projeví různými procedurami SELECT-VALUE

Forward-checking

- Jedna z metod pro dopředný pohyb pro výběr hodnoty
- Propaguje se efekt výběru hodnoty na každou následující proměnnou
- Když některá z domén následujících proměnných bude prázdná, právě zvažovaná hodnota se nevybere a zkusí následující
- Využívá proceduru SELECT-VALUE-FORWARD-CHECKING (slidy R. Dechter, chapter 5, slide 18)
- Pokud uvažujeme konstantní složitost ověření jednoho omezení na jedné dvojici hodnot, má SELECT-VALUE-FORWARD-CHECKING složitost ek^2 (k je kardinalita největší domény, e je počet omezení)

Arc-Consistency Look-Ahead

- Zajišťuje plnou hranovou konzistentnost všech dosud neinstanciovaných proměnných pro každou možnou hodnotu aktuálně přiřazované proměnné
- Využívá proceduru AC-1 pro hranovou konzistentost, kde některé proměnné již mají přiřazenou hodnotu
- S využitím nejoptimálnějších algoritmů pro hranovou konzistentnost je složitost při jedné proměnné $O(ek^3)$ (k hodnot, pro každou $O(ek^3)$ konzistentnost)
- Označován také jako **real full look-ahead**
- Oblíbenou variantou je MAINTAINING-ARC-CONSISTENCY (MAC)

MAC

- Provádí plnou hranovou konzistentnost kdykoliv je zamítnutá některá hodnota z domény
- Příklad: Vybíráme hodnotu pro x z domény $\{1, 2, 3, 4\}$. Nejprve dáme $x = 1$ a aplikujeme hranovou konzistentnost. Když se někde objeví prázdná doména, $x = 1$ je zamítnuto a doména se redukuje na $\{2, 3, 4\}$. Znovu se provede hranová konzistentnost.

Full Look-Ahead, Partial Look-Ahead

- Obě dělají více práce než forward-checking a méně než plná hranová konzistentnost
- Partial Look-Ahead - složitost je $O(ek^3)$, stejná jako Arc-Consistency Look-Ahead, nevystihuje to ale realitu

LVO

- Informaci získanou šířením omezení je možné použít i k ohodnocení neodmítnutých hodnot podle šance na to, že povedou k řešení
- Look-ahead value ordering (LVO)
- Může využít forward-checking nebo jakýkoliv vyšší stupeň šíření omezení
- Nepřiřadí se rovnou první možná hodnota, zkouší pro každou a zkoumá efekt forward-checking (nebo jiné metody) na domény dosud nepřiřazených proměnných
- LVO přístupy se liší nejen metodou šíření omezení, ale také heuristickými mírami použitými k ohodnocení hodnot. Některé jsou:
  - Min-conflicts (MC) - vybírá hodnotu, která odstraňuje nejméně hodnot z domén následujících proměnných
  - Max-domain-size (MD) - vybírá hodnutu, která bude mít největší minimální velikost domény na následujících proměnných
  - Estimate solutions (ES) - počítá horní mez na počet řešení násobením velikostí domén následujících proměnných po odstraněnní hodnot nekompatibilních s aktuálně uvažovanou. První volí tu s největším výsledkem.

Look-ahead pro uspořádání proměnných

- Uspořádání proměnných má obrovský význam na velikost prohledáváného prostoru
- Empirické a teoretické studie ukazují, že některá fixní uspořádání jsou v obecnosti efektivnější a generují menší prostory
- Např. min-width a max-cardinality jsou docela efektivní

Dynamické uspořádání proměnných

- Pořadí proměnných se určuje během prohledávání
- Běžná metoda je **fail-first** - vybrat vždy proměnnou, která má potenciál nejvíce omezit search space
- Při stejných ostatních faktorech se vybírá proměnná s nejmenší doménou - bude nejméně podstromů s kořeny v jednotlivých hodnotách z domény
- DVO - dynamic variable ordering
- Např. použití SELECT-VARIABLE (slidy R. Dechter, chapter 5, slide 26) v GENERALIZED-LOOK-AHEAD po iniciazaci $i \leftarrow 1$ a po kroku $i \leftarrow i + 1$
- DVFC - dynamic variable forward checking - DVO s využitím forward checking
- DVFC se v mnoha studiích ukázalo efektivní v porovnání přínosu a dodatečné výpočetní zátěže
- Algoritmus upravuje domény následujících proměnných, aby obsahovaly jen hodnoty konzistentní s aktuálním přiřazením. Vybere se proměnná s minimální doménou. (slidy R. Dechter, chapter 5, slide

Backtracking s náhodným výběrem

- Pořadí proměnných i hodnot má velký význam
- Je těžké najít jednu heuristiku, která bude ideální pro všechny případy
- Je možné využít náhodu
- Např. při min-domain bude pro výběr proměnné více kandidátů (se stejně velkou doménou) - zvolí se náhodně
- Náhodně lze volit i hodnoty, případně náhodu využít při "remíze" při využití heuristiky pro volbu hodnoty
- Potom je běžné zkoušet po (vhodně zvoleném) limitu zastavit hledání a spustit jej znovu. Limit je vhodné postupně zvyšovat a tím se zajistí jistota správného výsledku
- V praxi se ukázalo užitečné

Zpětné hledání

Konfliktní množiny

- Backjumping je jeden ze základních způsobů omezení tendence backtrackingu opakovaně prohlednávat stejné dead-end situace
- **Vinná proměná** (culprit variable) - její instanciace je odpovědná za dead-end (žádná možná kombinace následujících proměnných nevede k řešení)
- Když dokážeme identifikovat vinnou proměnnou, je backjump k nové instanciaci této proměnné lepší než postupné procházení prostoru backtrackingem
- Identifikace vinných proměnných je založena na konfliktních množinách
- Definice: Nechť $\bar{a} = (a_{i_1}, \ldots, a_{i_k})$ je konzistentní instanciace libovolné podmnožiny proměnných a x je dosud neinstanciovaná proměnná. Pokud v doméně x neexistuje hodnota b taková, aby $(\bar{a}, x = b)$ bylo konzistentní, nazveme $\bar{a}$ **konfliktní množinou** (conflict set) proměnné x. Pokud navíc $\bar{a}$ neobsahuje žádnou podmnožinu konfliktní s x, nazveme $\bar{a}$ **minimální konfliktní množinou** proměnné x.

Konfliktní množiny

- Definice: Nechť $\bar{a}_i = (a_1, \ldots, a_i)$ je konzistentní $i$tice. Když je $\bar{a}$ v konfliktu s x_{i+1} nazveme situaci listový (leaf) dead end a proměnná x_{i+1} je listová dead-end proměnná
- Definice: Pro síť $\mathcal{R} = (X, D, C)$ každou částečnou instanciaci $\bar{a}$, která není součástí žádného řešení, nazýváme **nedobrou** (no-good).
- Každá konfliktní množina je nedobrá, ale nedobrá množina nemusí být konfliktní pro jednu proměnnou
- Definice: Nechť $\bar{a}_i = (a_1, \ldots, a_i)$ je listová dead-end situace. Řekneme, že x_j, jde $j \leq i$ is **bezpečná** (safe) vzhledem k $\bar{a}$, když částečná instanciace $\bar{a}_j = (a_1, \ldots, a_j)$ je nedobrá.

Backjumping styly

- **Gaschnig's backjumping** - skáče zpět na vinnou proměnnou jen při listové dead-end situaci.
- **Graph-based backjumping** - vytáhne z grafu omezení informaci o zbytečných krocích zpět a skáče i ve vnitřních (ne listových) dead-end situacích
- **Conflict-directed backjumping** - kombinuje skoky v listových i vnitřních dead end situacích a neomezuje se jen na informace z grafu

Gaschnig's backjumping

- Definice: Nechť $\bar{a}_i = (a_1, \ldots, a_i)$ je listový dead-end. Index viny (culprit index) vzhledem k $\bar{a}_i$ je definován jako $b = min\{j \leq i \mid \bar{a}_j$ je v konfliktu s $x_{i+1}\}$. Vinná proměnná instanciace $\bar{a}_i$ je x_b.
- Tvrzení: Když je $\bar{a}_i$ listový dead-end a x_b je vinná proměnná, tak $\bar{a}_b$ je bezpečná destinace pro backjump a $\bar{a}_j, j < b$ není.
- Určení vinné proměnné pro $\bar{a}_i$ je relativně snadné - je potřeba otestovat maximálně i podmnožin na konzistentnost s x_{i+1}
- Navíc může být určena v průběhu hledání udržováním nějakých informací při přiřazování $\bar{a}_i$

Graph-Based backjumping

- Když algoritmus skáče zpět na proměnnou x_j z listového dead-endu a x_j nemá další hodnoty k vyzkoušení, označujeme ji **vnitřní dead-end proměnnou** a $\bar{a}_{j-1}$ **vnitřní dead-end stav**
- Gaschnigův algoritmus provádí větší skoky jen z listových dead-endů, když skočí na vnitřní, tak se zachová jako standardní backtracking
- Algoritmus **graph-Based backjumping** umožňuje větší skoky i ve vnitřních dead-endech
- Informaci o možných konfliktních množinách získává pouze z grafu omezení (nezkoumá tedy typ omezení nebo třeba domény)
- Při dead-endu skáče na naposledy přiřazenou proměnnou, která je s aktuální proměnnou spojena hranou v grafu omezení
- Pokud je po skoku znovu dead-end, opět skáče na naposledy přiřazenou proměnnou, která je v grafu spojena s některou z proměnných, při kterých na dead-end narazil

Conflict-Directed Backjumping

- Spojuje dva předchozí postupy
- Pracuje podobně jako graph-based backjumping, ale nedívá se jen na graf omezení
- Pro každou proměnnou si udržuje **jumpback set**
- Definice: Při daném uspořádání proměnných nazveme omezení R dřívější než omezení Q, když nejpozdější proměnná z $scope(R) - scope(Q)$ předchází nepozdější proměnnou z $scope(Q) - scope(R)$

Conflict-Directed Backjumping

- Relace dřívějšího omezení definuje úplné uspořádání na omezeních
- Definice: Nechť pro síť $\mathcal{R} = (X, D, C)$ s uspořádáním proměnných d je $\bar{a}_i$ ntice jejíž potenciální dead-end proměnná je x_{i+1}. **Nejdřívější minimální konfliktní množina** pro $\bar{a}_i$ (nebo pro x_{i+1}), označovaná $emc(\bar{a}_i)$ může být vytvořena takto: uvažujme omezení uspořádáné podle relace dřívějšího, pro $j \in \{1, \ldots, c\}$ (počet omezení), pokud existuje $b \in D_{i+1}$ tž. R_j je porušeno přiřazením $\bar{a}_i, x_{i+1} = b$ a žádné dřívější omezení tímto porušeno není, potom do $var - emc(\bar{a}_i)$ přidáme S_j (rozsah R_j), $emc(\bar{a}_i)$ potom získáme projekcí $emc(\bar{a}_i) = a_i[var - emc(\bar{a}_i)]$
- Definice: **Jumpback set** J_{i+1} listového dead-endu x_i je jeho $var - emc(\bar{a}_i)$. Jumpback set vnitřního stavu $\bar{a}_i$ (nebo proměnné x_{i+1}) obsahuje všechny $var - emc(\bar{a}_j)$ všech relevantních dead-endů $\bar{a}_j, j \geq i$, které nastaly v současné řešení proměnné x_i

Conflict-Directed Backjumping

- Tvrzení: Pro danou ntici $\bar{a}_i$ je nejpozdější proměnná v jumpback množině J_i nejdřívější proměnnou, kam je bezpečné skočit.
- Algoritmus - viz slidy R. Dechter, chapter 6, slide 21

Učící se algoritmy

- Při construkci minimální konfliktní množiny se ujasní nedobré instanciace a použijí se k rozhodnutí skoku zpět
- Stejně tak se takové instanciace dají přidat formou nových omezení, takže je algoritmus nebude v budoucnu znovu procházet díky testování konzistentnosti
- Tím se může "prosekat" strom prohledávaného prostoru
- Techniku nazýváme **constraint recording** ne **learning**
- Příležitost k naučení nového omezení je při každém dead-endu
- Pokud $\bar{a}_i$ je konfliktní množinou pro x_{i+1}, nedává moc smysl si přimo mezi omezení zakázat $\bar{a}_i$
- Pokud $\bar{a}_i$ obsahuje podmnožiny konfliktní s x_{i+1}, může se vyplatit zaznamenat tyto podmnožiny.
- Výhodné je identifikovat co nejmenší konfliktní množiny a ty si zapamatovat
- Jedním z kandidátů je earliest minimal conflict set (emc) z konflikty řízeného backjumpingu

Graph-Based Learning

- **Graph-based learning** využívá stejné metody jako graph-based backjumping k určení nedobrých instanciací
- Informace o konfliktech vyvozuje pouze z grafu omezení
- Pro listový dead-end $\bar{a}_i$ jsou hodnoty přiřazené předkům x_{i+1} zaznamenány v uložené konfliktní množině.
- Příklad viz slidy R. Dechter, chapter 6, slide 30

Deep learning

- **Deep learning** znamená, že se zaznamenávají jen minimální konfliktní množiny
- **Shallow learning** - netrvá na tom, aby zaznamenané množiny byly minimální
- Deep learning využívá nejvíce informace k prosekání prohledávaného stromu, ale výpočet minimálních konfliktních množin je náročný (až exponenciální)

Další varianty učících se algoritmů

- Jumpback Learning - místo všech minimálních konfliktních množin si pamatuje jen jednu, většinou jumpback množinu z conflict-directed backjumping
- Bounded and Relevance-Bounded Learning - každý učící algoritmus může být doplněn o omezení na velikost konfliktních množin
- Nonsystematic Randomized Backtrack Learning - při pravděpodobnostních algoritmech (náhodný výběr hodnot) se pamatují všechny nedobré instanciace, aby po restartu nebyly znovu procházeny

SAT

SAT

- Konjunktivní normální forma (KNF) umožňuje snadný test syntaktické správnosti formule, v obecnosti je obtížné určit splnitelnost (NP-úplný problém)
- V disjunktivní normální formě (DNF) je snadné určit splnitelnost
  - Stačí, aby byl splnitelný alespoň jeden disjunkt
  - Disjunkt (konjunkce literálů) je splnitelný, pokud se v něm žádná proměnná nevyskytuje v pozitivní (nenegované) i negativní (negované) formě
  - Každou formuli je možné převést do DNF
  - Při převodu do DNF formule může narůst exponenciálně, tedy polynomiální řešení formule v DNF je exponenciální vůči původní formuli
- I některé speciální tvary formulí v KNF je možné řešit v polynomiálním čase, např.
  - Horn SAT
  - 2-SAT

Horn SAT

- Formule je Hornova, pokud může být generována gramatikou (s počátečním symbolem H)
  - $P ::= \bot \mid \top \mid p$
  - $A ::= P \mid P \wedge A$
  - $C ::= A \rightarrow P$
  - $H ::= C \mid C \wedge H$
- Příklad: $(p \wedge q \wedge s \rightarrow \bot) \wedge (q \wedge r \rightarrow p) \wedge (p \wedge s \rightarrow s)$
- Po převodu Hornovy formule do KNF je v každé klauzuli právě jeden pozitivní literál (je možno prostřednictvím tohoto faktu Hornovy formule alternativně definovat)

Řešení Horn SAT

- Vstup: Hornova formule Φ (ve tvaru konjunkce implikací)
- Algoritmus:
  - označ všechny výskyty $\top$ v Φ
  - dokud exisuje konjunkt $P_1 \wedge P_2 \wedge \ldots \P_k \rightarrow P'$, kde všechny P_i jsou označeny ale P' ne, označ P' (všechny jeho výskyty ve formuli)
  - pokud je označen nějaký výskyt $\bot$, vrať nesplnitelná, jinak vrať splnitelná
- Počet průchodů cyklem je lineární vůči počtu proměnných ve formuli, čas jednoho průchodu cyklem je také lineární vůči velikosti formule, algoritmus je tedy kvadratický
- Korektnost algoritmu - indukcí je možno ukázat, že plati: "Všechna označená P jsou true pro všechny valuace, ve kterých se Φ vyhodnotí na $\top$"

2-SAT

- Formule v KNF se 2 literály v každé klauzuli
- Každou formuli je možné polynomiálně převést do KNF se 3 literály v klauzuli, ale již neexistuje obecný polynomiální algoritmus převodu do KNF se 2 literály
- Každá klauzule instance 2-SAT vlastně odpovídá jednoduché implikaci: $x_0 \vee \neg x_3 \equiv (\neg x_0 \rightarrow \neg x_3) \equiv (x_3 \rightarrow x_0)$
- Je proto možné zadávat formule v implikační normální formě (konjunkce implikací)
- Implikační graf - vrchol pro každý literál a orientované hrany podle implikace v implikační normální formě
- Existuje mnoho algoritmů pro řešení 2-SAT, nejefektivnější z nich jsou lineární

Kromův algoritmus

- Je polynomiální
- Navržen již v roce 1967
- Předpokládejme, že ve Φ jsou dvě klauzule, jedna s x a druhá $\neg x$
- Takové klauzule zkombinujeme a vytvoříme novou (rezoluční pravidlo)
- Z $(a \vee b) \wedge (\neg b \vee \neg c)$ vznikne $a \vee \neg c$
- V implikační formě rezoluční pravidlo odpovídá tranzitivitě
- Konzistentní formule je taková, kde nemůžeme současně vytvořit $(x \vee x)$ a $(\neg x \vee \neg x)$
- Když je formule konzistentní, je možné postupně přidat pro každou proměnnou právě jednu z hodnot $(x \vee x)$ (resp. $(\neg x \rightarrow x)$) nebo $(\neg x \vee \neg x)$ (resp. $(x \rightarrow \neg x)$)tak, aby konzistentní zůstala
- Valuace splňující formuli dává true všem proměnným s klauzulí $(x \vee x)$ a false všem s klauzulí $(\neg x \vee \neg x)$

Kromův algoritmus

- Při návrhu se Krom soustředil na úplnost a správnost, nerozebíral složitost
- Dá se ukázat, že pracuje v polynomiálním čase
- Zgrupují se všechny klauzule obsahující stejnou proměnnou a aplikují se všechny možné rezoluce v $O(n^3)$, kde n je počet proměnných (pro každou proměnnou může být $O(n^2)$ klauzulí, které ji obsahují a se kterými lze potenciálně dělat rezoluci)
- $O(n)$ testů konzistentnosti, maximálně $O(n^3)$ každý, tedy $O(n^4)$ celkem
- Při důkladnějším zamyšlení se dá odvodit $O(n^2)$

Omezený backtracking

- Even, Itai, Shamir - 1976
- Obecný algoritmus pro CSP s binárními proměnnými a binárními omezeními
- Idea - přiřazovat hodnoty proměnným postupně, při tzv. choice point vybere hodnotu, při backtrackingu se nikdy nevrátí přes poslední choice point
- Na začátku není žádný choice point (CP) a všechny proměnné jsou bez přiřazené hodnoty
- Pak následují kroky
  - Pokud existuje klauzule s oběma proměnnými nastavenými tak, že je klauzule false, vrátit se zpět na nejbližší CP, odebere přiřazení od tohoto CP a změní hodnotu v tomto CP
  - Pokud existuje klauzule, kde je jedna z proměnných nastavená a klauzule může být true i false, tak druhá klauzule je nastavena tak, aby klauzule byla true
  - Pokud všechny klauzule mají garantováno true nebo mají obě proměnné nepřiřazené, vytvoří se CP a některá nepřiřazená proměnná se nastaví na true nebo false

Omezený backtracking

- Na některých vstupech může být často backtracking a dlouhý řetězec přiřazení mezi návraty, takže nemusí být lineární
- Vylepšení - od CP se dělají paralelně obě větve (pro sekvenční algoritmus prolínáním) a když první větev narazí na nový CP, druhá větev končí
- Po vylepšení je možné dokázat lineární složitost

Silně souvislé komponenty

- Aspvale, Plass, Tarjan - 1979
- Jednodušší procedura v linearním čase založená na silně souvislých komponentách (SCC - strongly connected components)
- SCC - v orientovaném grafu jde o rozklad (množina podmnožin, vzájemně mají prázdný průnik a jejich sjednocením je původní množina) množiny vrcholů tž. každý vrchol v komponentě je dosažitelný z každého dalšího vrcholu stejné komponenty orientovanou cestou
- Existuje několik efektivních algoritmů na hledání SCC v grafu v lineárním čase, většinou jsou založené na hledání do hloubky

Silně souvislé komponenty

- Provede se rozklad na SCC na implikačním grafu (kde pro $x_0 \vee \neg x_3$ je $(\neg x_0 \rightarrow \neg x_3)$ i $(x_3 \rightarrow x_0)$)
- Pokud do stejné komponenty patří x i $\neg x$, tak je formule nesplnitelná
- Autoři ukázali, že toto je nutná, ale také postačující podmínka
- Formule 2-SAT je tedy splnitelná právě tehdy, když neexistuje proměnná patřící do stejné komponenty jako její negace
- Lineární algoritmus rozhodující splnitelnost - v lineárním čase provede rozklad na SCC a poté pro každou proměnnou zkontroluje, ve které komponentě se nachází její negace

Silně souvislé komponenty

- Pro nalezení splňujícího ohodnocení
  - Vytvořit implikační graf, udělat SCC
  - Ověřit splnitelnost, pro nesplnitelné formule skončit
  - Vytvořit kondenzovaný graf - za každou klauzuli se dá jeden vrchol a hrana mezi vrcholy je tam, kde mezi komponentami vedla alespoň jedna hrana. Z vlastností SCC je výsledný kondenzovaný graf acyklický
  - Komponenty se uspořádají topologicky (některé algoritmy pro hledání SCC je rovnou nachází podle tohoto uspořádání)
  - Berou se komponenty podle pořadí, komponentě bez přiřazení je přiřazeno false. Doplňková komponenta (obsahující opačné literály stejných proměnných) tak bude true. Všechny předcházející komponenty už musejí mít přiřazeno také false. Pokud je něco true, všem následujícím komponentám se přiřadí také true.

Aplikace 2-SAT

- Bezkonfliktní umístění objektů - popisky grafu nebo mapy, návrh VLSI (very large scale integration) obvodů
- Data clustering - hledá se minimální poloměr clusterů v metrickém prostoru
- Scheduling - např přiřazení domácích a hostujících celků po nalosování rozpisu sportovních zápasů tak, aby se omezila po sobě jdoucí utkání stejného týmu na domácím hřišti (resp. na soupeřově hřišti)
- Digitální tomografie - rozpoznání tvarů z jejich průsečíků
- V řešičích SAT jako podprocedura, když po částečném přiřazení a úpravě zůstane jen instance 2-SAT
- ...

Obecný SAT

- Rozšířením algoritmu pro Hornovy formule lze získat algoritmus pro obecné formule
- Podformule se označují značkami true, false tak, že všechny označené podformule se vyhodnotí na svou značku pro každé ohodnocení, které celou formuli vyhodnotí na true.

Lineární řešič

- Formule se převedou do fragmentu $\Phi ::= p \mid (\neg\Phi) \mid (\Phi \wedge \Phi)$
- Vytvoříme k takto převedené formuli syntaktický graf, kde stejné podformule reprezentuje jeden podstrom. Jedná se o orientovaný acyklický graf (directed acyclic graph - DAG)
- Převod formule je induktivní: $T(p) = p$, $T(\neg\Phi) = \neg T(\Phi)$, $T(\Phi_1 \wedge \Phi_2) = T(\Phi_1) \wedge T(\Phi_2)$, $T(\Phi_1 \vee \Phi_2) = \neg(\neg T(\Phi_1) \wedge \neg T(\Phi_2))$, $T(\Phi_1 \rightarrow \Phi_2) = \neg(T(\Phi_1) \wedge \neg T(\Phi_2))$
- Φ je splnitelná právě tehdy, když $\neg\Phi$ je splnitelná
- Příklad syntaktického stromu a odpovídajícího DAG je na obrázku 1.12 v souboru satsolvers.pdf
- Je definovaná sada pravidel pro vynucená nové značky - viz. obrázek 1.14 v souboru satsolvers.pdf
- Jen označení grafu vynucenými značkami nestačí
- Je nutné zkontrolovat přepočítáním zdola nahoru, když se shoduje, máme svědka splnitelnosti
- Příklad dokázání splnitelnosti formule je na obrázku 1.13 v souboru satsolvers.pdf

Lineární řešič

- Tento SAT solver má lineární časovou složitost vzhledem k velikosti DAG pro $T(\Phi)$
- Transformace $\Phi \rightarrow T(\Phi)$ zvětší formuli také lineárně, tedy algoritmus je lineární vůči Φ
- Linearita algoritmu je vykoupena tím, že selže na formulích typu $\neg(\Phi_1 \wedge \Phi_2)$

Kubický řešič

- V lineárním řešiči jsme uvažovali 2 možnosti:
  - Vynucený spor - současně vynucené T i F k jednomu uzlu
  - Úplné ohodnocení všech uzlů
- Existuje třetí možnost - všechna vynucená omezení jsou konzistentní, ale ne všechny uzly jsou ohodnocené
- Příklad: $(p \vee q \vee r) \wedge (p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (\neg p \vee \neg g \vee \neg r)$
- Tato formule není splnitelná - první a poslední klauzule říkají, že alespoň jedna z proměnných p, q, r musí být true a alespoň jedna false. Zbylé klauzule říkají, že všechny tyto 3 proměnné mají stejnou hodnotu.
- Lineární řešič nenajde ani spor ani úplné ohodnocení (viz. obrázek 1.17 ze souboru satsolver.pdf)

Kubický řešič

- Když není označeno vše a není spor, vybere se uzel a přiřadí se mu dočasně T.
- Zkoumáme, zda přiřazením nevznikl spor, pokud ano, uzel dostane místo dočasného T permanentní F.
- Když spor nenastane, zkusí se totéž s dočasnou hodnotou F.
- Všechny uzly, které v obou případech dostaly stejnou hodnotu, budou mít tuto hodnotu jako permanentní.
- Stejně se testují další uzly
- Příklad běhu algoritmu je na obrázku 1.18 v souboru satsolver.pdf
- Pořád se ale může stát, že nepřiřadí všem uzlům nějakou permanentní hodnotu - když každá z voleb T,F vede k různým značením ostatních uzlů, ale ne ke sporu.
- Kubická složitost:
  - Každý test je vlastně použití lineárního řešiče
  - Musíme testovat pro každý nepřiřazený uzel
  - Každé nové permanentní přiřazení způsobí, že ostatní nepřiřazené je potřeba testovat znovu

Kubický řešič

- Pokud řešič vrátí, že je formule nesplnitelná nebo splnitelná, je to vždy správně
- Může se stát, že vrátí odpověď "Nevím"

Další přístupy řešení SATu

- Konflikty řízené učení klauzulí (conflict-driven clause learning) - moderní varianta DPLL algoritmu
- Stochastické lokální hledání - WalkSAT
- DPLL - Systematický backtracking procházející exponenciální prostor, pochází z 60. let
- Moderní řešiče SAT:
  - Conflict-driven
  - Look-ahead

Konflikty řízené řešení SAT

- Davis–Putnam–Logemann–Loveland (DPLL) algoritmus s:
  - efektivní analýzou konfliktů
  - učením klauzulí
  - nechronologickým backtrackingem (backjumpingem)
  - adaptivním větvením
  - náhodnými restarty
- Tato vylepšení se empiricky ukázala důležitá pro zvládání velkých instancí SAT

Look-ahead přístup

- Speciální posílená redukce a heuristiky
- Jsou obecně silnější na těžkých instancích

DPLL

- Vybere literál, přiřadí mu hodnotu, zjednoduší formuli a rekurzivně otestuje splnitelnost
- Když je rekurzivní volání neúspěšné, změní hodnotu a znovu zjednoduší formuli a rekurzivně volá
- Zjednodušení
  - Odstraní klauzule, které jsou true
  - Odstraní literály, které jsou false, z ostatních formulí
- Unit propagation - když klauzule obsahuje 1 literál, tak může být splněna jen jeho nastavením na true. Tedy není nutné větvení. V praxi to často vede ke kaskádě jednotek a z toho plyne velké zmenšení prohledávacího prostoru
- Pure literal elimination - když se proměnná vyskytuje ve formuli pouze v jedné polaritě, nazývá se čistá (pure). Tyto proměnné můžeme nastavit vždy tak, aby všechny klauzule, kde se vyskytují, byly true. Čili tyto klauzule lze odstranit.

DPLL

- Nesplnitelnost vyplyne, když některá klauzule bude prázdná, čili všechny proměnné byly nastaveny tak, že literály v této klauzuli nastavovali na false
- Splnitelnost formule - když přiřadí hodnoty všem proměnným a nedojde ke sporu, čili k prázdné klauzuli, nebo také, když všechny klauzule jsou splněny
- Výzkum pro vylepšení algoritmu:
  - Politika výběru literálů pro větvení
  - Nové datové struktury pro urychlení operací na formuli (hlavně operace unit propagation)
  - Varianty základního backtrackingu (backjumping, clause learning)

Stochastické algoritmy pro SAT

- Nejznámější jsou GSAT a WalkSAT
- Oba jsou určeny na formule v KNF
- Přiřadí náhodné ohodnocení
- Když je formule splněna, končí
- Když není splněna, přehodí hodnotu jedné proměnné a opakují postup
- Stochastické algoritmy jsou neúplné - u splnitelné formule najdou často řešení, ale neumí ukázat, že je formule nesplnitelná
- Po nějaké době hledaní se obvykle aplikuje restart - znovu náhodně přiřadí hodnotu všem proměnným a běží na této nové instanci
- Intervaly mezi restarty se často postupně zvětšují

WalkSAT

- Vybere některou nesplněnou klauzuli a v ní přehodí proměnnou
- Klauzule volí náhodně
- Proměnnou v klauzuli volí tak, aby minimální počet splněných klauzulí změnil na nesplněné
- S menší pravděpodobností i proměnnou volí náhodně
- Ukázal se vhodný při instancích vzniklých z automatického plánování (přístup převedení plánování na SAT se nazývá SATplan)

GSAT

- Náhodně přiřadí hodnoty proměnných
- Vybere proměnnou, která maximálně sníží počet nesplněných klauzulí a změní její hodnotu
- Problémem jsou lokální minima - v jiných problémech může nalezení lokálního minima stačit, v SATu ne
- "Krok stranou" (změna hodnoty jiné proměnné, než by určil algoritmus) značně vylepší úspěšnost
- Mezi přístupy k opuštění lokálních minim se často uplatňuje simulované žíhání

Optimalizační problémy

Optimalizační problémy

- V praxi se často rozlišují hard a soft omezení
- Analýza vede k síti omezení rozšířené o globální cenovou (neboli kriteriální) funkci
- Řešení musí splňovat všechna hard omezení a optimalizovat cenovou funkci
- Mnoho problémů z průmyslu je tohoto typu
- Většinu CSP problémů můžeme uvažovat i v optimalizační verzi - když není možné splnit všechna omezení, hledá se řešení splňující co nejvíce omezení
- Max-CSP - třída problémů vzniklá tímto způsobem
- Jedním z nejznámějších problému z Max-CSP je MaxSAT (snaha najit ohodnocení formule v KNF splňující co nejvíce klauzulí)
- Lze také omezením dát váhy a místo počtu minimalizovat sumu vah porušených omezení

Optimalizační problém

- Optimalizační problém s omezeními (COP - constraint omptimization problem) je síť omezení rozšířená o cenovou funkci
- $X = \{x_1, \ldots, x_n\}$
- $F_1, \ldots, F_l$ jsou reálné funkce definované nad $Q_1, \ldots, Q_l$, kde $Q_j \subseteq X$
- Nechť $\bar{a} = \{a_1, \ldots, a_n\}$, kde a_i je z domény x_i, nechť $\bar{a}_i$ je přiřazení prvním i proměnným. Globalní cenová funkce F je definována jako $F(\bar{a}) = \sum_{j=1}^{l} F_j(\bar{a})$, kde $F_j(\bar{a})$ znamená $F_j(\bar{a})$ aplikována na přiřazení $\bar{a}$ omezené na doménu funkce F_j
- Cenová síť (cost network) je $C = (X, D, C_h, C_s)$ kde (X, D, C_h) je síť omezení a $C_s = \{F_{Q_1}, \ldots, F_{Q_l}\}$ je množina cenových komponent nad doménami $Q_1, \ldots, Q_l$, $Q_i = \{X_{i_1}, \ldots, X_{i_l}\}$
- Cenová síť je jednou z možných reprezentací optimalizačního problému s omezeními
- Funkce v C_s jsou měkká (soft) omezení
- Optimalizační úloha je snaha najít $\bar{a^0} = \{a_1, \ldots, a_n\}$ splňující všechna omezení, tž. $F(\bar{a^0}) = max_{\bar{a}} F(\bar{a})$ (popřípadě místo maxima lze hledat i minimum)

Cenový graf

- Uzel pro každou proměnnou
- Hrany pro jakékoliv omezení (hard i soft)

Řešení COP jako série CSP

- Vždy se řeší úloha CSP s hard omezeními a s limitem cenové funkce, který se postupně zvyšuje (tedy omezení, že globalní cenová funkce je větší než limit)
- Postupně se tak zvyšuje kvalita nalezeného řešení
- Řešení je optimální v rámci limitu daného rychlostí zvyšování limitu
- Promyšlenější přístup - použití "binárního vyhledávání"
- Lze tedy použít pro optimalizaci jakékoliv techniky, které se používají (a byly probrány) pro CSP

Branch and Bound search

- Nejjednoduššní a nejnaivnější přístup - rozšířený backtracking
- Po prvním nalezeném řešení se backtracking nezastaví, ale prohledá i zbytek prostoru
- Pamatuje si aktuálně nejlepší nalezené řešení
- Může využít jakoukoliv techniku zlepšení backtrackingu na hard omezeních
- Je možné dále omezit prohledávaný prostor analýzou cost funkce
- Např. když součet funkcí nad již přiřazenými hodnotami je vyšší než dosud nejlepší (s minimální cenovou funkcí) řešení, tak ve větvi není potřeba pokračovat

Depth-first branch and bound

- Udržuje nejlepší řešení
- Počítá horní mez s využitím omezující funkce $f(\bar{a}_i)$, která nadhodnocuje nejlepší řešení, které je rozšířením $\bar{a}_i$
- Když i tento nadhodnocený odhad je menší, než dosud nejlepší nalezené řešení, není potřeba $\bar{a}_i$ dále rozšiřovat (pokračovat ve větvi výpočtu, která k němu vedla)
- Omezující funkce může být použita také jako heuristika pro výběr hodnoty, která se přiřadí aktulně zpracovávané proměnné
- Při hard omezeních využívá techniky zmíněné dříve, ale v každém uzlu prohledávacího stromu přidá kontrolu omezující funkce
- Obdobně je možné řešit minimalizaci

Omezující funkce

- Základním možným vylepšením branch-and-bound je zlepšení přesnosti omezující funkce
- Nejprve byly algoritmy vyvíjeny a zlepšovány pro třídu problémů známých jako celočíselné programování
- Mají lineární tvrdá omezení a globální cenovou funkci, proměnné mají celočíselné domény
- Některé idee se rozšiřují na obecná omezení a obecné cenové funkce

Russian Doll search

- Idea - n po sobě jdoucích branch-and-bound hledání, každé s přidanou jednou proměnnou (a relevantními omezeními)
- První podproblém zahrnuje jen $n$tou proměnnou, $i$tý posledních i proměnných
- Každý podproblém se řeší přes branch-and-bound, kde mezní funkce využívá znalostí z předchozích běhů
- Navíc dříve nalezaná optimální řešení jsou využívána jako heuristiky pro výběr hodnot k přiřazení a ke zlepšení počáteční horní (dolní) meze řešení
- V praxi se ukazuje, že i n prohledávání je efektivní, protože jednotlivá hledání jsou rychlejší díky omezení prohledávaného prostoru

Bucket elimination

- Využívá datovou strukturu bucket
- Proměnné předpokládáme uspořádány podle nějakého uspořádání
- Cenové funkce se dělí do košíků odpovídajících proměnným
- Začne od košíku poslední proměnné a dá do něj všechny funkce s touto proměnnou v doméně
- Do každého dalšího košíku dá funkce, které proměnnou onoho košíku obsahují v doméně, ale zatím nebyly nikam přiděleny
- Košíky se zpracovávají od poslední proměnné
- Zpracování košíku - sečtou se funkce v něm eliminuje se proměnná odpovídající košíku maximalizací
- Vzniklou funkci zařadíme do odpovídajícího nižšího košíku (patřícímu větší proměnné) a pokračujeme následující proměnnou (o jedna větší v uspořádání)
- Po skončení (maximalizaci funkce v košíku nejvyšší proměnné) dostane hodnotu cenové funkce optimálního řešení
- Zpětně od nejvyšší proměnné k nejnižší lze získat přiřazení hodnot všech proměnných pro optimální řešení

Složitost algoritmu bucket elimination

- Složitost je ovlivněna uspořádáním proměnných
- Čím více rozměrné funkce eliminací vznikají, tím více prostoru je potřeba pro jejich uložení (exponenciální růst prostoru)
- Arita funkcí závisíá na počtu proměnných v košíku
- S pomocí grafových technik (výpočet grafové šířky apod.) lze aritu odhadnout
- Složitost je lineární vůči počtu omezení (hard + soft), ale exponenciální vůči indukované grafové šířce

Počítání počtu řešení

- $\sharp P$ - třída problémů, které řeší nedeterministický turingův stroj a řešení je počet akceptujících výpočtů
- Obvykle problémy spojené s problémy v NP
- Některé problémy v rozhodovací verzi jsou polynomiálně řešitelné, ale počítání počtu jejich řešení je $\sharp P$-úplné - např. 2-SAT, splnitelnost formule v DNF, párování apod.
- Počítání počtu řešení lze elegantně řešit eliminací košíků
- Vstupní relaci reprezentuje 1 pro konzistentní a 0 pro nekonzistentní přiřazení
- Místo maximalizace při eliminaci košíku se používá součet, místo sčítání funkcí je násobení
- Lze také řešit prohledáváním, které projde celý prostor jakýmkoliv backtrackingem

Mini-bucket elimination

- Eliminace košíků potřebuje pro některé vstupy exponenciální prostor
- Lze se inspirovat z řešení CSP, kde místo plné konzistentnosti se dělá i-konzistentnost pro nějaké rozumně malé i
- V košíku ukládáme aproximaci prostřednictvím funkcí o menší aritě (minikošíky v rámci košíky)
- Maximalizujeme každou tuto funkci s menší aritou zvlášť a maxima sčítáme. Vzniklá funkce je aproximací původní funkce
- Maximalizovaný souet v prvním koši je potom horní mez
- Dolní mez se spočítá při druhé fázi algoritmu (dohledávání hodnot proměnných)
- Kvalita mezí závisí na stupni rozkladu do mini-košíků
- Je mnoho možností, jak rozdělovat proměnné do minikošíků, vedou k různě přesným mezím a dává proto smysl hledat vhodné heuristiky pro tento rozklad

Plánování, rozrhování

Plánování

- Základní zkoumaná úloha - je dána sekvence úloh, které mají být zpracovány na dostupných strojích.
- V nejjednodušší verzi je úloha dána časem běhu a musí na tento čas být přidělena na nějaký stroj
- V jiných variantách se uvažují další omezení specifikující, který plán zpracování úloh je povolen
- Snažíme se úlohy naplánovat co nejefektivněji, nejčastěji to znamená, aby byly za co nejkratší čas všechny hotové
- Budeme se zabývat tzv. on-line verzí, kdy algoritmus nemá k dispozici najednou celý seznam úloh
- Mnoho plánovacích úloh je NP-úplných
- Po důkazech NP-úplnosti se výzkum zaměřil na aproximační algoritmy

Definice apod.

- Počet strojů označujeme m, počet úloh n
- Každá úloha je charakterizovaná dobou běhu t a případně dalšími parametry podle typu úlohy
- Na každý stroj v každém okamžiku může být přidělena pouze jedna úloha
- Všechny plánovací úlohy směřují k minimalizaci nějaké funkce (míra výkonu)
- Výkon on-line algoritmů měříme prostřednictvím poměru kvality nalezeného řešení ku optimálnímu řešení (bereme hodnotu, že pro všechny možné instance je poměr menší nebo roven této hodnotě)
- Minimální celkový čas pro dokončení všech úloh označujeme T_{opt}

Různá paradigmata

- Plánování jeden po druhém (one by one) - úlohy jsou v seznamu, algoritmus je dostává po jedné a musí každou přiřadit některému stroji, aniž by se mohl podívat na následující položky seznamu úloh (ale může si pamatovat předchozí úlohy). Je možné úlohy plánovat na jakýkoliv časový slot, ale po naplánování to již nelze změnit.
- Neznámé časy běhu - než úloha skončí, není známa doba jejího běhu. Často je zde možnost restartu nebo preempce, algoritmus má přístup k již přiřazeným úlohám a může přiřazení měnit.
- Intervalové plánování - každá úloha má přesně daný interval, kdy může být zpracována. Úlohy, které není možné zpracovat v tomto intervalu, je možné odmítnout.

Kriteriální funkce

- Nejčastěji celková doba zpracování
- Když je možné úlohy odmítat - doba zpracování + pokuta za odmítnutí
- Součet časů, kdy jsou jednotlivé úlohy dokončené
- Součet časů, který stráví jednotlivé úlohy v systému (od jejich příchodu do dokončení)
- Součet časů, které stráví úlohy čekáním na zpracování
- Uvažují se i vážené varianty - hodnoty různých úloh ovlivňují funkci s různou váhou
- Při povolené preempci můžeme uvažovat i počet preempcí úloh

List scheduling

- Základní algoritmus, studovaný již v roce 1966
- Úlohy jsou uspořádány v seznamu (sekvenci)
- Kdykoliv je nějaký stroj volný, naplánujeme na něj první úlohu ze seznamu, která je k dispozici (tedy není ještě naplánována, aktuální čas je větší než release čas úlohy a všechny předcházející úlohy v precedenčním grafu jsou skončeny)
- Je možné jej použít v různých paradigmatech
- Výkonostní poměr tohoto algoritmu je $2 - 1/m$

Plánování jeden po druhém

- Algoritmus musí úlohu ze seznamu přiřadit na stroj bez znalosti následujících úloh a přiřazení již pak nemůže změnit
- Obvykle se neuvažují precedenční omezení a release časy
- Většinou se ani neurčuje časový slot, jen stroj
- Pro $m = 2$ a $m = 3$ je prokazatelně nejlepší list scheduling (LS)
- Pro větší počty strojů jsou i lepší algoritmy
- Hlavní problém LS - kdy stroje rovnoměrně zatíží a potom přijde dlouhá úloha (může být až skoro 2x delší plán, než by mohl být)
- Řešení - nějaká mírná disbalance - některé stroje nechává mírně zatížené aby na ně mohl naplánovat dlouhé úlohy
- Pro počet strojů nad 4 získáme poměr lepší než LS, když se úloha přidělí vždy na jeden ze dvou nejméně vytížených strojů. Pro velký počet strojů se ale také poměr blíží 2 (jako u LS)
- Aby se poměr neblížil 2 pro velký počet strojů, je nutné jich nechat nevytížených nějaký konstantní počet - navrženo několik algoritmů

Randomizované algoritmy

- Mnohem méně prozkoumány než deterministické
- Uvažujeme poměr střední doby zpracování úloh ku optimální době
- Pro dva stroje algoritmus s poměrem $4/3$, což je nejlepší možné:
  - Když to jde, naplánuje úlohu náhodně tž. střední doba délky plánu je $2/3$ součtu časů zpracování úloh
  - Když to nejde, naplánuje úlohu na méně zatížený stroj
- I pro $m = 3, 4, 5$ je znám nejlepší možný algoritmus, pro $m = 6, 7$ je lepší než deterministický, ale není dokázáno, že by byl nejlepší možný
- Vždy dává úlohu na jeden ze dvou nejméně vytížených strojů. Pro velké počty strojů se výkonostní poměr ale blíží také 2

Plánování s preempcí

- Úloha může být přiřazena více strojům, ale časové sloty musí být disjunktní
- Přiřazení musí být určeno hned, když úloha dorazí (stále uvažujeme paradigma jeden po druhém)
- Pro off-line je řešení lehké - optimální celkový čas je maximum z maximální doby běhu úlohy a sumy dob běhů všech úloh dělené m
- Deterministické i randomizované verze jsou plně vyřešeny

Plánování s odmítáním

- Za pokutu je možné úlohu odmítnout
- Každá úloha má kromě dané doby výpočtu určenou i pokutu
- Kriteriální funkce je celková doba na zpracování všech úloh plus součet pokut za odmítnuté úlohy
- Uvažujeme stroje se stejnou rychlostí a nebereme do úvahy žádná další omezení (precedenci apod.)
- Hledá se tedy rovnováha mezi zařazením úlohy a tím prodloužením doby výpočtu a pokutou za odmítnutí úlohy
- Na začátku je často výhodné úlohy s malou pokutou odmítnout, ale později může být výhodné je dodatečně zařadit a zpracovat (vytíží se třeba volný stroj a sníží se pokuta)

Plánování s odmítáním

- Deterministické algoritmy bez preempce - je dokázáno, že po efektivním rozhodnutí o odmítnutí úlohy již na přidělování úloh na stroje stačí LS alg.
- Optimální řešení je známo pro dva stroje a pro velký počet strojů
- Pro dva stroje:
  - úloha je odmítnuta, když $t \geq mp$ (p je pokuta)
  - úloha je odmítnuta, když $t \geq \phi(P + p)$ (P je celková dosud zaplacená pokuta, ϕ je zlatý řez)
  - akceptované úlohy naplánuje prostřednictvím LS
- Randomizované - varianty deterministických, náhodně se volí prahy pro odmítnutí úlohy

Různé rychlosti strojů

- Uvažují se 3 varianty:
  - Related - stroje mají různé rychlosti, ale stejné pro všechny úlohy
  - Unrelated - vektor rychlostí strojů se liší pro různé úlohy
  - Restricted - úloha má určeno, na kterých strojích může být zpracována
- Doubling strategy - konstantní poměr pro related verzi:
  - Odhadneme celkový čas každou úlohu dáme na nejpomalejší stroj tak, aby nebyl odhad překročen
  - Není-li naplánování možné, zdvojnásobíme odhad
- Poměr jde vylepšit sofistikovanějšími technikami zvětšování odhadu

Shop scheduling

- Úloha (job) je složena z podúloh (task)
- Uvažují se 3 varianty:
  - Open shop - podúlohy můžou být zpracovány v libovolném pořadí, ale ne dvě současně na různých strojích
  - Flow shop - pořadí podúloh je pevné a stejné pro všechny úlohy
  - Job shop - pořadí podúloh je pevné pro každou úlohy, ale pro jinou úlohu může být jiné
- Poprvé mezi zmiňovanými problémy dává smysl nechávat na strojích volné sloty
- Pro flow a job shop není možné navrhnout algoritmus s poměrem lepším než 2, dosáhnout tento poměr je snadné