Plánování a rozvrhování

Roman Barták, KTIML

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Přednáška

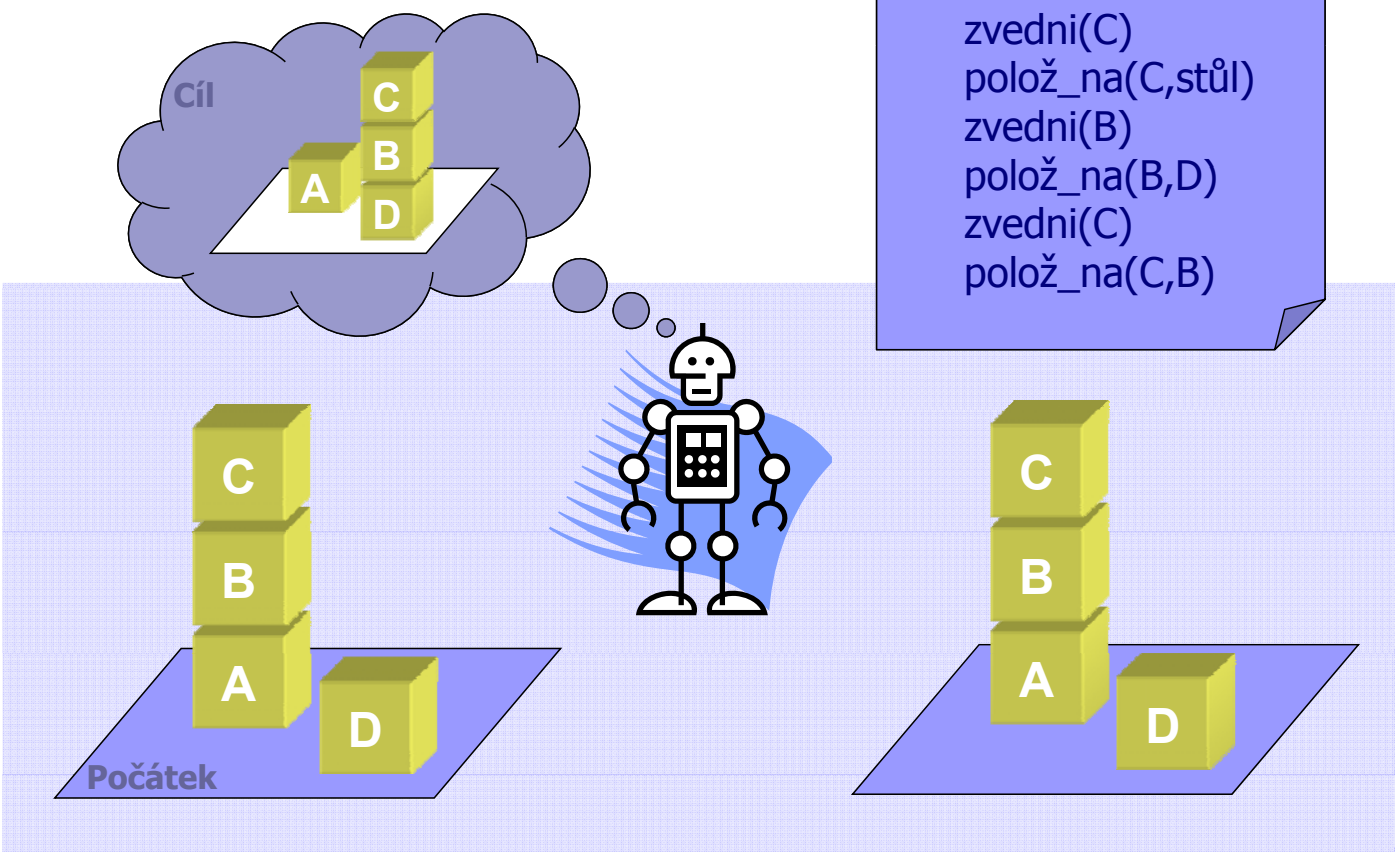
- **Co** je obsahem?
  - plánování a rozvrhování
  - ale co to vlastně je plánování a rozvrhování?
- **Proč** by mě to mělo zajímat?
  - ono se to někde používá?
  - aplikace?
- **O čem** bude přednáška?
  - modelování problémů
  - řešící algoritmy

Co?

Čím se zabývá plánování a rozvrhování?
Jaký je mezi nimi rozdíl?

Plánování v příkladě

Svět kostek



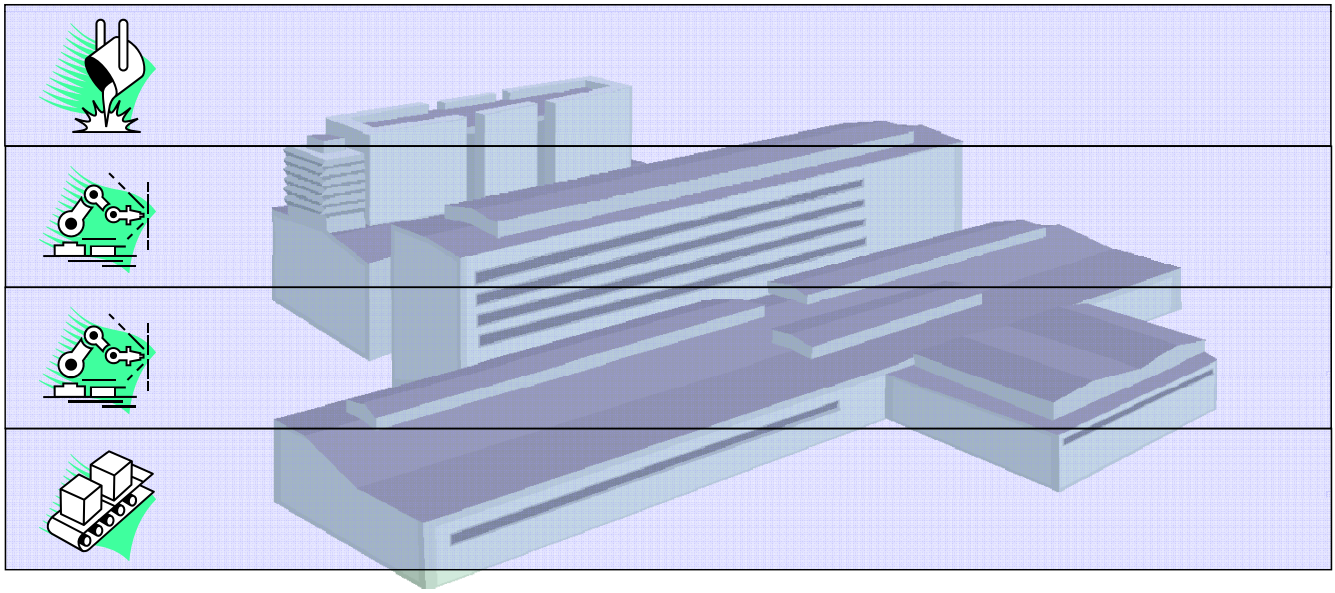
Plán

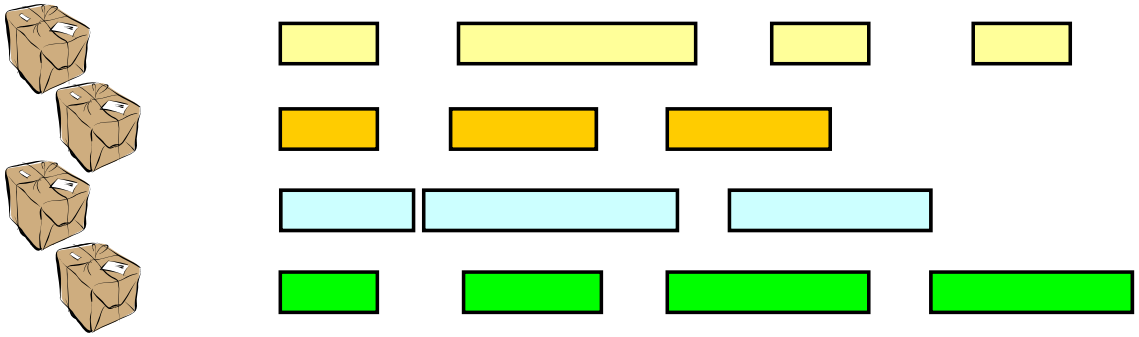
zvedni(C)
polož_na(C,stůl)
zvedni(B)
polož_na(B,D)
zvedni(C)
polož_na(C,B)

Plánování v kostce

- **Vstup:**
  - počáteční (současný) stav světa
  - popis akcí schopných měnit stav světa
  - požadovaný stav světa
- **Výstup:**
  - seznam akcí (plán)
- **Vlastnosti:**
  - akce v plánu nejsou známy dopředu
  - čas a zdroje nehrají roli

Rozvrhování v příkladě



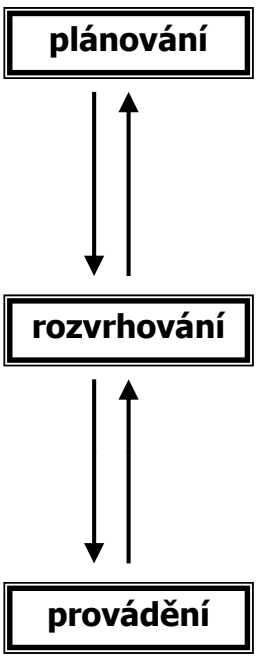


Rozvrhování v kostce

- **Vstup:**
  - skupina částečně uspořádaných aktivit
  - dostupné zdroje
- **Výstup:**
  - alokace aktivit na zdroje a v čase (rozvrh)
- **Vlastnosti:**
  - aktivity jsou známy předem
  - čas a zdroje jsou omezené

Srovnání

- **Plánování (planning)**
  - rozhodování jaké akce jsou potřeba pro dosažení daných cílů
  - téma umělé inteligence
  - složitost často horší než NP-c (obecně nerozhodnutelné)
- **Rozvrhování (scheduling)**
  - rozhodování jak zpracovat dané akce použitím omezených zdrojů a času
  - téma operačního výzkumu
  - složitost typicky NP-c



Proč?

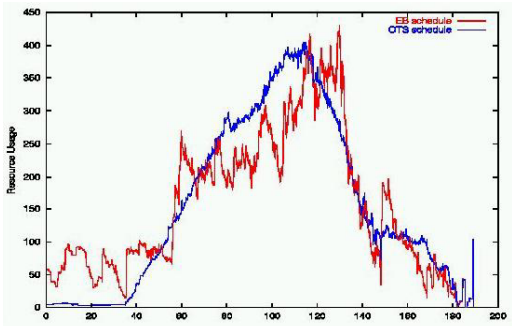
K čemu je ta technologie dobrá?
Má to praktické využití?

Výroba letadel

- 570 úloh, 17 zdrojů
- Tradiční metoda:
  - ARTEMIS
  - 20 hodin na vytvoření rozvrhu
- Metoda IP&S:
  - ARTEMIS nahrazen CSP
  - 30 minut na vytvoření optimálního rozvrhu
  - 10 - 15% kratší makespan
- **Úspory:**
  - 4 - 6 dní na letadlo
  - **0.2 - 1 milion US$ za den**

Výroba ponorek

- 1,4 milionu úloh na jeden člun
- Tradiční metoda:
  - ARTEMIS
  - 6 týdnů na vytvoření rozvrhu
  - vysoce neuniformní profil
- Metoda IP&S:
  - ARTEMIS nahrazen CSP
  - 2 dny na vytvoření rozvrhu
  - více uniformní profil



- **Úspory:**
  - **30% redukce přesčasů a subkontraktů**

Logistika

Válka v zálivu 1991:

- Tradiční metoda:
  - Stovky lidských plánovačů
  - Měsíce na vytvoření plánů
- Metoda IP&S:
  - O-PLAN2 pomáhá plánovačům
- **Úspory:**
  - Rychlejší vytvoření zázemí
  - Méně leteckých misí
  - Finanční návratnost >> **veškerý výzkum AI sponzorovaný US vládou**:
    - od roku 1956
    - nejen výzkum IP&S, ale **veškerý výzkum AI!**

Deep Space 1

Start: 24. října 1998

Cíl: Borrelliova kometa

Testování 12 nových technologií

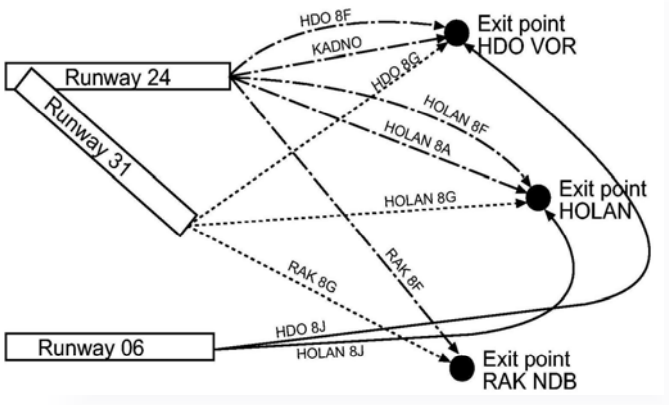
- **autonomous remote agent**
  - plánuje, provádí a monitoruje akce kosmické lodi na základě obecných příkazů operátora
  - tři zkušební scénáře
    - 12 hodin nízké autonomie (provádění a monitorování)
    - 6 dní vysoké autonomie (snímání kamerou, simulace poruch)
    - 2 dny vysoké autonomie (udržení směru)
      - **pozor na backtracking!**
      - **pozor na deadlock v plánu!**

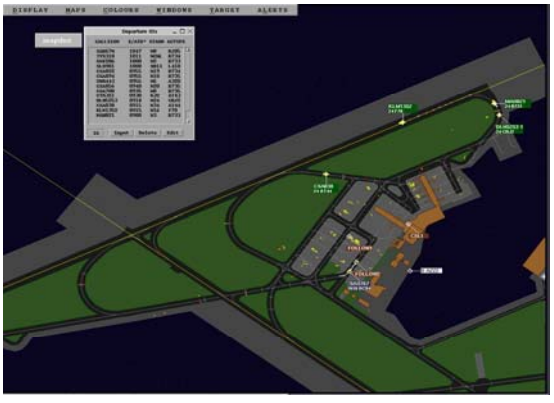
Contribution of NASA Ames

Letový provoz

Správa odletů

- předletová příprava
  - přiřazuje odletovou trasu
  - po dohodě s Bruselem
- pozemní kontrola
  - pojíždění
- kontrola na věži
  - rozjezdové dráhy
  - separace





MANTEA

(**MAN**agement of surface **T**raffic in **E**uropean **A**irports)

- implementováno v **ILOG Scheduler**
- testováno v **Praze** (27.5. – 7.6. 2002)

Contribution of NLR

O čem?

Co mi tato přednáška přinese?

Co přednáška pokrývá?

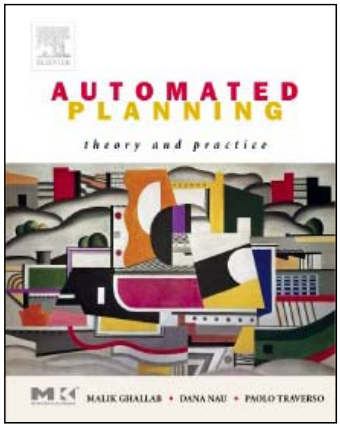
Struktura

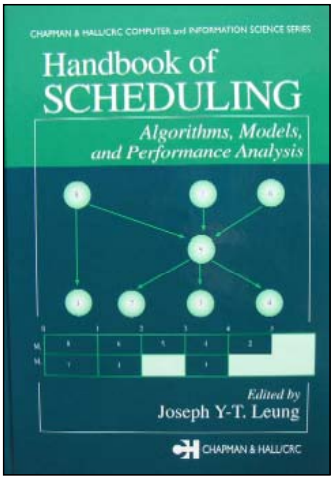
- **Úvod**
  - prohledávací algoritmy, omezující podmínky a SAT
- **Plánování**
  - klasické plánování (STRIPS)
  - neoklasické plánování (Graphplan)
  - hierarchické plánování
- **Rozvrhování**
  - tradiční rozvrhování
  - rozvrhování s omezujícími podmínkami
- **Aplikace**

Literatura

Automated Planning: Theory and Practice

- M. Ghallab, D. Nau, P. Traverso
- http://www.laas.fr/planning/
- Morgan Kaufmann





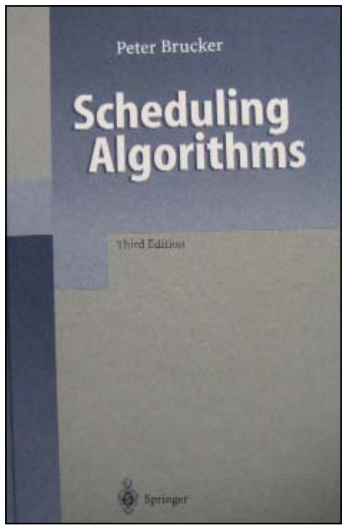
Handbook of Scheduling

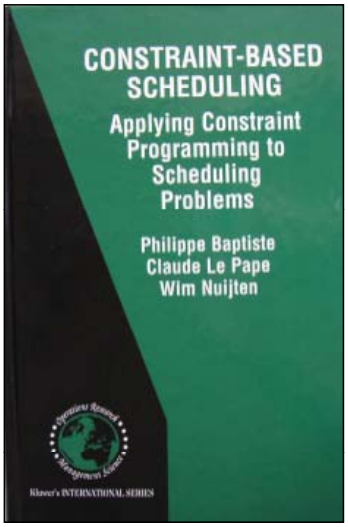
- J. Leung
- Chapman&Hall/CRC

Literatura

Scheduling Algorithms

- P. Brucker
- Springer





Constraint-based Scheduling

- P. Baptiste, C. Le Pape, W. Nuijten
- Kluwer

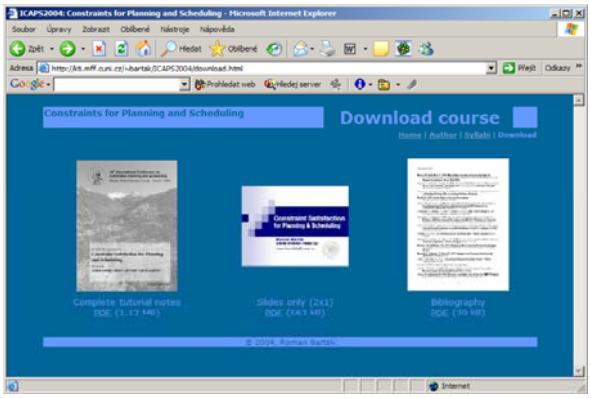
On-line zdroje

PLANET

European NoE in Planning

`http://www.planet-noe.org/`

- **materiály z letních škol**
- **prezentace z workshopů**
- **katalog plánovacích systémů**
- **aplikační oblasti**



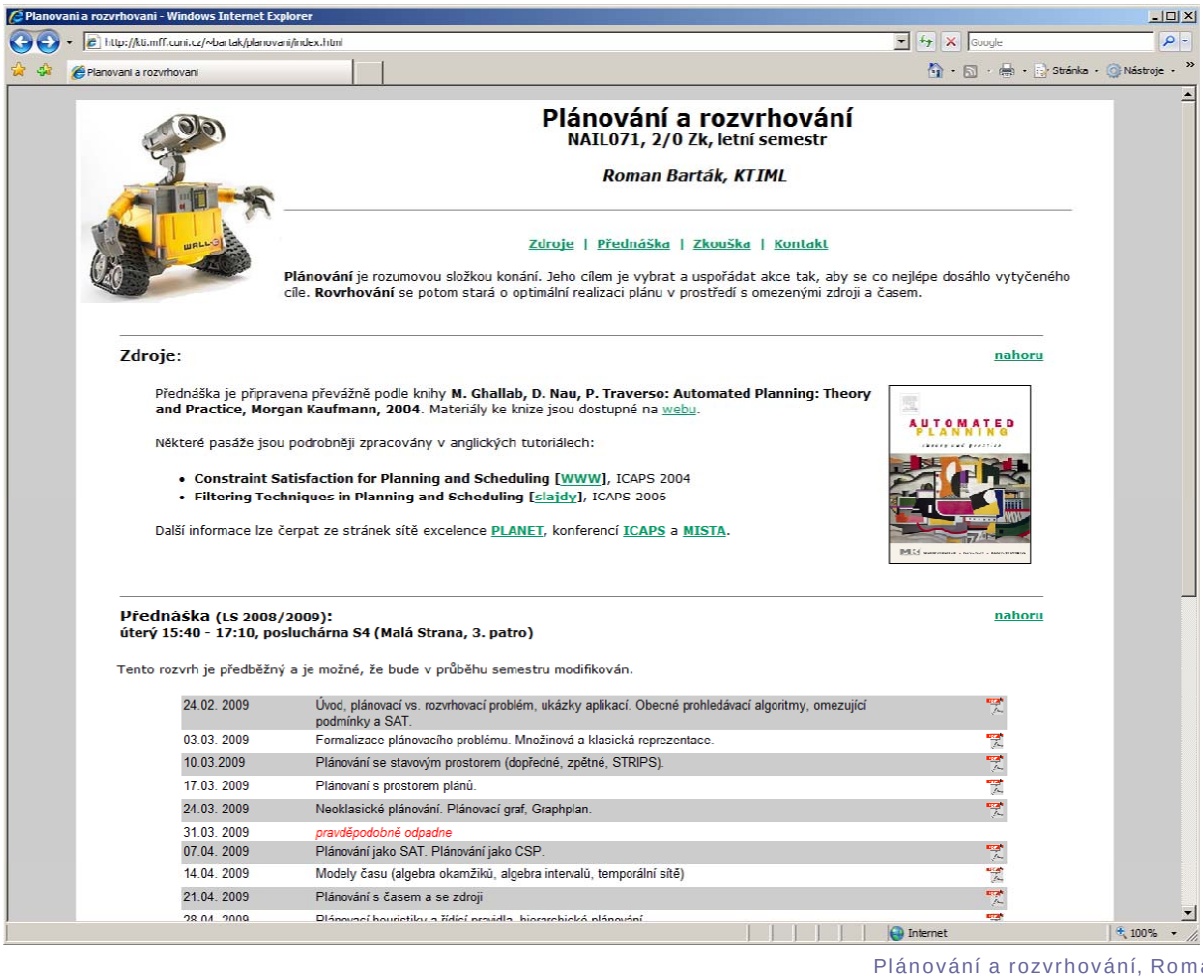
ICAPS 2004 tutorial

Constraint Satisfaction for Planning and Scheduling

`http://kti.mff.cuni.cz/~bartak/ICAPS2004/`

- **slajdy**
- **reference**

Web přednášky



Příprava

Na úvod některé základní techniky

- **prohledávání**
- **splňování omezujících podmínek**
- **SAT a logika**

Prohledávání

Prohledávání (search) je základní technikou řešení problémů umělé inteligence (a nejen UI).

Typy prohledávání:

- **Prohledávání stavového prostoru**
  - hledá se stav daných vlastností
- **Prohledávání prostoru redukcí problému**
  - hledá se rozklad na triviální problémy

Vlastnosti algoritmů

- **korektnost** (soundness)
  - To, co algoritmus vrátí, je řešením problému.
- **úplnost** (completeness)
  - Když existuje řešení, algoritmus ho nalezne.
- **přípustnost** (admissibility)
  - Algoritmus garantuje nalezení optimálního řešení.
  - Pouze, je-li definována míra optimality řešení!
  - Implikuje korektnost a úplnost.

Stavový prostor

- **Stavový prostor** S je množina uzlů (stavů), kde cílem je najít stav splňující danou podmínku g.
- Formálně je **problém** zadán trojicí (s_0,g,O):
  - s_0 je **počáteční stav**
  - g je **cílová podmínka** (tj. hledaný stav s splňuje g(s))
  - O je množina **operátorů** měnících stav
    - Stavový prostor je potom definován rekurzivně:
      - $s_0 \in S$, je-li $s \in S, o \in O$ a o(s) je definováno potom $o(s) \in S$
    - o(s) je **potomek** uzlu s

Do šířky

Breadth-First Search

jde po „patrech“

- seznam q se chová jako **fronta**

```
bfs(s0,g,O)
    q ← {s0}
    while non-empty(q) do
        s ← first(q)
        if g(s) then return s
        q ← delete_first(q)
        q ← q + {s‘ | ∃o∈O, s‘=o(s)}
    end while
    return failure
```

- **Úplný algoritmus**
- **Složitost** nalezení cílového uzlu v hloubce d při b potomcích každého uzlu:
  - čas $O(b^d)$
  - paměť $O(b^d)$

Do hloubky

Depth-First Search (backtracking)

jde zvoleným směrem

návrat v případě neúspěchu

- seznam q se chová jako **zásobník**

```
dfs(s0,g,O)
    q ← {s0}
    while non-empty(q) do
        s ← first(q)
        if g(s) then return s
        q ← delete_first(q)
        q ← {s‘ | ∃o∈O, s‘=o(s)} + q
    end while
    return failure
```

- **Úplný algoritmus**, pokud nejsou nekonečné cesty nebo je lze detekovat.
- **Složitost** nalezení cílového uzlu v hloubce d:
  - čas záleží na směru (může prohledat až celý stavový prostor, ale také může jít přímo)
  - paměť O(d)

Uspořádaně

Někdy se hledá cílový stav s minimalizující hodnotu objektivní funkce f(s).

Best-First Search

jde vždy do aktuálně nejlepšího uzlu

- seznam q se chová jako **prioritní fronta**

```
bestfs(s0,g,O,f)
    q ← {s0}
    while non-empty(q) do
        s ← best(q,f)
        if g(s) then return s
        q ← delete_best(q,f)
        q ← q ⊕ {s' | ∃o∈O, s'=o(s)}
    end while
    return failure
```

- Pokud f neklesá (s`=o(s) ⇒ f(s)≤f(s`)) potom nalezené řešení je optimální. Je-li navíc stavový prostor konečný, algoritmus je přípustný.
- Pokud existuje δ>0 tž. s`=o(s) ⇒ f(s)+δ≤f(s`), pak je algoritmus přípustný i pro nekonečné prostory.

Větve a meze

Další optimalizační algoritmus minimalizující f.

Depth-First Branch-and-Bound Search

projde vše

a pamatuje si nejlepší

- seznam q se chová jako **zásobník**

```
dfbbs(s0,g,O,f)
    s* ← dummy % f(dummy)=∞
    q ← {s0}
    while non-empty(q) do
        s ← first(q)
        q ← delete_first(q)
        if g(s) & f(s)<f(s*) then
            s* ← s
        else
            q ← {s' | ∃o∈O, s'=o(s)}+q
        end if
    end while
    return s*
```

- Pokud f neklesá a stavový prostor je konečný a bez cyklů, potom je algoritmus přípustný.

Hladově

Greedy Search

do hloubky

ale bez navracení

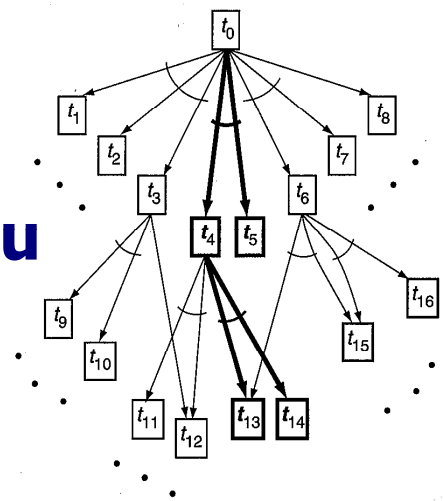
```
gs(s0,g,O,f)
    s ← s0
    while not g(s) do
        s ← best({s' | ∃o∈O, s'=o(s)},f)
    end while
    return s
```

- Negarantuje nalezení optimálního řešení
- Někdy může uspořit spousty času, který by bylo potřeba pro nalezení garantovaného optima.
- Často se používá pro nalezení prvního dobrého řešení.

AND-OR grafy

- Někdy operátor o poskytuje množinu potomků, tzv. **podproblémy**, které reprezentují část řešení rodiče. Řešení potomka je potom částí řešení rodiče.
- Vzniká tak **AND-OR graf**.

↳ **Prohledávání prostoru možných redukcí problému**



Problem Reduction Search

rozloží problém
a hledá řešení podproblémů

- nedeterministické
- naivní
  - opakovaně řeší společné podproblémy

```
preds(s,g,O)
    if g(s) then return s
    applicable ← {o∈O | o(s)↓}
    if applicable=∅ then return failure
    o ← choose_nondet(applicable)
    {s1,…,sn} ← o(s)
    for every si∈{s1,…,sn} do
        vi ← preds(si,g,O)
        if vi=failure then return failure
    end for
    return {v1,…,vn}
```


Příprava

Na úvod některé základní techniky

- prohledávání
- **splňování omezujících podmínek**
- SAT a logika

Technologie CP

založena na **deklarativním popisu problému**:

- **proměnné s doménami** (množiny možných hodnot)
  např. start akce s přiřazeným časovým oknem
- **podmínky** omezující kombinace hodnot proměnných
  např. startA+PA < startB

optimalizace s podmínkami
např. minimalizace makespanu

Proč používat techniky CP?

- snad pochopitelné
- otevřené a rozšiřitelné
- opravdu to funguje

CSP

- **Problém splňování podmínek** se skládá z:
  - konečné množiny **proměnných**
  - **domén** – konečná množina hodnot pro proměnnou
  - konečné množiny **podmínek**
    - podmínka je relace nad množinou proměnných
    - může být definována extenzionálně (množina kompatibilních n-tic) nebo intenzionálně (formulí)
- **Řešením** CSP je přiřazení hodnot všem proměnným tak, že jsou splněny všechny podmínky.

CSP jako Svatý grál

```
> Počítači vyřeš problém SEND, MORE, MONEY!

> Tady to je můj pane. Řešení je
   [9,5,6,7]+[1,0,8,5]=[1,0,6,5,2]
```

z pohledu Star Treku

```
> Sol=[S,E,N,D,M,O,R,Y],
  domain([E,N,D,O,R,Y],0,9), domain([S,M],1,9),
            1000*S + 100*E + 10*N + D +
            1000*M + 100*O + 10*R + E #=
  10000*M + 1000*O + 100*N + 10*E + Y,
  all_different(Sol),
  labeling([ff],Sol).

> Sol = [9,5,6,7,1,0,8,2]
```

současná realita

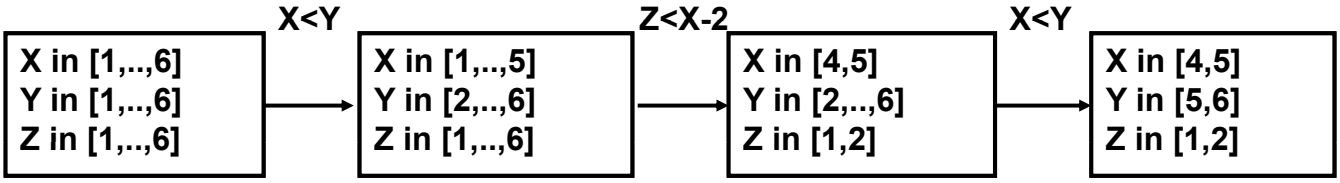
Filtrace domén

- **Příklad:**
  - D_a={1,2}, D_b={1,2,3}
  - a<b
  - ↳ Hodnota 1 může být z D_b bezpečně vyřazena.
- Podmínky se používají **aktivně pro odstranění nekonzistencí** z problému.
  - nekonzistence = hodnota, která nemůže být součástí žádného řešení (nesplňuje nějakou podmínku)
- Tato tzv. **filtrace domén** je realizována procedurou REVISE, kterou má každá podmínka.

Děláme AC

- Jak zařídit „globální" filtraci v CSP?
- Každá podmínka musí být zrevidována!

Příklad: X in [1,..,6], Y in [1,..,6], Z in [1,..,6], X<Y, Z<X-2



↳ Nestačí ale zrevidovat každou podmínku pouze jednou!

- Revize je potřeba opakovat dokud se mění doména nějaké proměnné (AC-1).

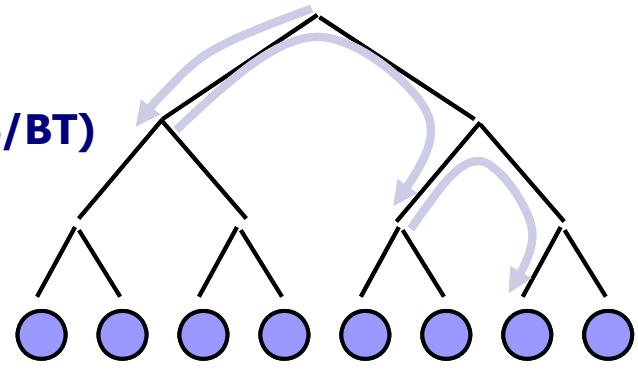
Prohledávání / Přiřazování

Konzistenční techniky jsou (obvykle) neúplné.

↳ **Potřebujeme prohledávací algoritmus, který vyřeší zbytek!**

Přiřazování (labeling)

- **prohledávání do hloubky (DFS/BT)**
  - **přiřaď hodnotu do proměnné**
  - **propaguj = udělej problém lokálně konzistentní**
  - **vrať se v případě neúspěchu**



- **X in 1..5 ≈ X=1 ∨ X=2 ∨ X=3 ∨ X=4 ∨ X=5**

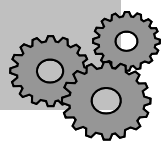
Obecně prohledávací algoritmus řeší zbylé disjunkce!

- **X=1 ∨ X≠1** (standardní přiřazovaní)
- **X<3 ∨ X≥3** (dělení domén)
- **X<Y ∨ X≥Y** (uspořádání proměnných)

Kostra ohodnocování

- Prohledávání do hloubky je kombinováno s AC, které omezuje prohledávaný prostor.
- **Technika pohledu dopředu (MAC)**

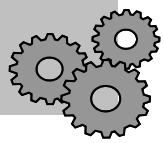
```
procedure labeling(V,D,C)
        if all variables from V are assigned then return V
        select not-yet assigned variable x from V
        for each value v from Dx do
                (TestOK,D') ← consistent(V,D,C∪{x=v})
                if TestOK=true then R ← labeling(V,D',C)
                if R ≠ fail then return R
        end for
        return fail
end labeling
```



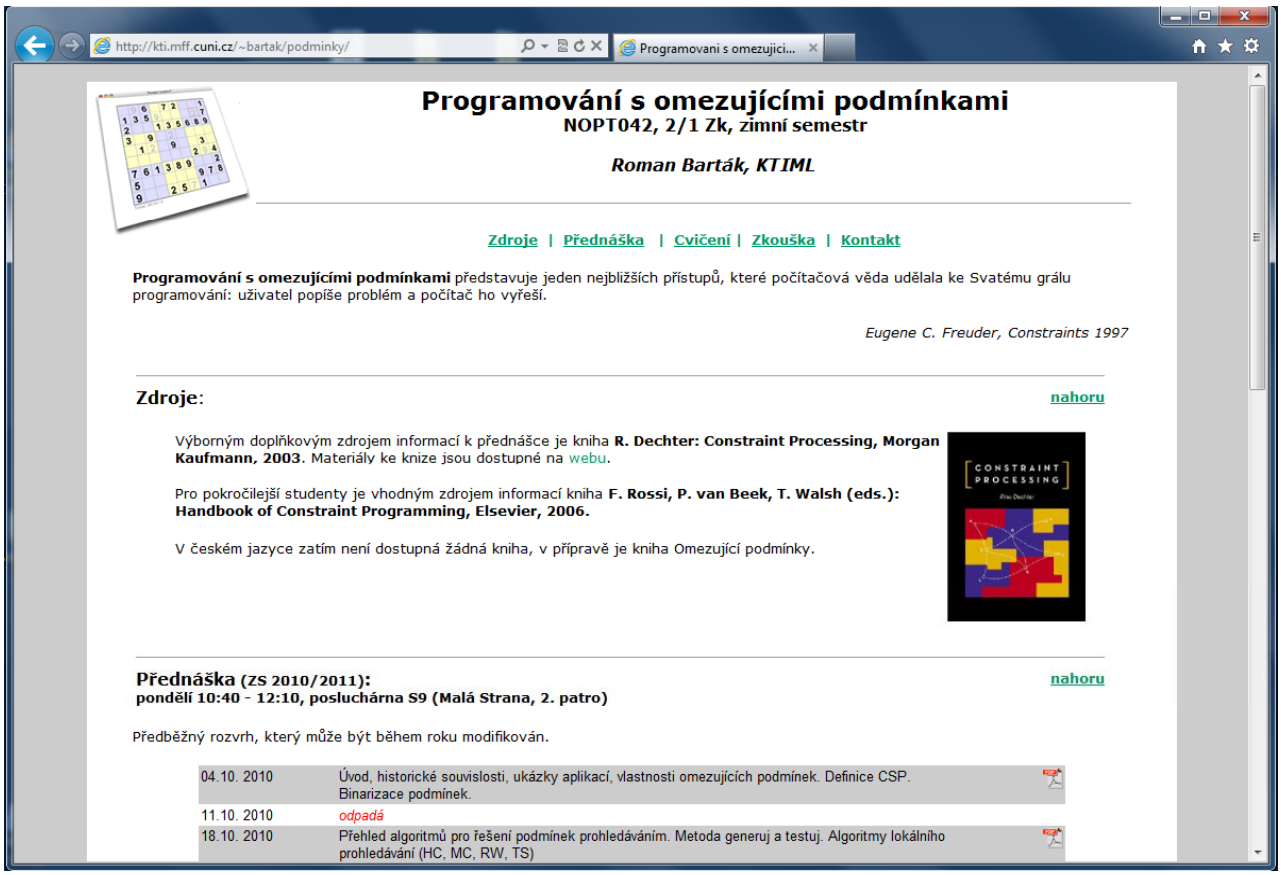
Optimalizační problémy

- **Problém optimalizace s podmínkami** (COP) = CSP + objektivní funkce
- Objektivní funkce se zakóduje do podmínky.

```
Technika branch and bound
  ┌→ najdi kompletní ohodnocení proměnných
  │       (definuje novou mez objektivní funkce)
  │  ulož nalezené ohodnocení
  │  aktualizuj mez (založ podmínku omezující
  │       hodnotu objektivní funkce tak, aby byla
  │       lepší než daná mez, což způsobí neúspěch)
  └─ pokračuj v prohledávání (do vyčerpání)
     obnov uložené ohodnocení
```



Chcete vědět více?



Příprava

Na úvod některé základní techniky

- prohledávání
- splňování omezujících podmínek
- **SAT a logika**

Logika prvního řádu

Formální systém skládající se z:

- **jazyka**
  (množiny tvrzení, obvykle nazývaných formule)
  např. $p \rightarrow q$
- **sémantiky**
  (dává význam tvrzení)
  např. je-li p, q pravda potom je $p \rightarrow q$ *pravda*
- **důkazové teorie**
  (pravidla, jak lze odvozovat nová tvrzení)
  např. pravidlo modus ponens $(p, p \rightarrow q \vdash q)$

Výroková logika

- Jazyk je **množina výroků** P definovaný induktivně nad rekurzivní množinou atomických výroků (výrokových proměnných) P_0 takto:
  - je-li $p \in P_0$ potom $p \in P$,
  - je-li $p \in P$ potom $\neg p \in P$,
  - je-li $p \in P$ a $q \in P$ potom $p \wedge q \in P$,
  - a nic víc není výrok.
- Můžeme dále definovat
  - $p \vee q$ jako zkratku za $\neg(\neg p \wedge \neg q)$
  - $p \rightarrow q$ jako zkratku za $\neg p \vee q$
- **Konjunktivní normální tvar** (CNF) výrokové formule:
  - **formule** je konjunkce klauzulí
  - **klauzule** je disjunkce literálů
    (obsahuje-li jediný literál, nazýváme ji **jednotkovou klauzulí**)
  - **literál** je výroková proměnná (pozitivní literál) nebo její negace (negativní literál)

Splnitelnost

- **Model výrokové formule** je množina přiřazení pravdivostních hodnot pravda/nepravda do výrokových proměnných tak, že formule je vyhodnocena jako pravdivá, kde:
  - $\neg p$ je pravda právě tehdy, když p je nepravda
  - $p \wedge q$ je pravda právě tehdy, když p je pravda a q je pravda.
- **Problém splnitelnosti (SAT)** je problém určení, zda existuje model dané formule.

Davis-Putnam

- Řešení SAT problému (v CNF) **prohledáváním do hloubky** a jednotkovou propagací.
- **Jednotková propagace** určuje pravdivostní hodnotu proměnných v jednotkových klauzulích
  - proměnná v pozitivním literálu ohodnocena na pravda,
  - proměnná v negativním literálu ohodnocena na nepravda.

  Přiřazená hodnota je propagována do ostatních klauzulí.

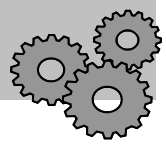
  Je-li D ohodnoceno jako pravda, potom lze:
  - klauzuli $(A \vee \neg B \vee D)$ z formule vyřadit
  - klauzuli $(C \vee \neg D \vee E)$ zjednodušit na $(C \vee E)$

  Je-li D ohodnoceno jako nepravda, potom symetricky.

Algoritmus DP

```
procedure DP(A,Assignment)
    A: is a CNF formula (represented as a set of clauses)
    A and Assignment are local within DP
    if ∅∈A then return
    if A=∅ then exit with Assignment
    Unit-Propagate(A, Assignment)
    select a variable P such that P or ¬P occurs in A
    DP(A∪{P},Assignment)
    DP(A∪{¬P},Assignment)
end DP
```

```
procedure Unit-Propagate(A,Assignment)
    A and Assignment are global within Unit-Propagate
    while there is a unit clause {l} in A do
        Assignment ← Assignment ∪ {l}
        for every clause C∈A do
            if l∈C then A ← A - {C}
            else if ¬l∈C then A ← A - {C} ∪ (C-{¬ l})
end Unit-Propagate
```



Algoritmus GSAT

- GSAT je algoritmus lokálního prohledávání řešící SAT problém postupným „překlápěním" proměnných.
- Cílem je maximalizovat (vážený) počet splněných klauzulí.

```
procedure GSAT(A,Max_Tries,Max_Moves)
    A: is a CNF formula
    for i:=1 to Max_Tries do
        S ← random assignment of variables
        for j:=1 to Max_Moves do
            if A satisfiable by S then return S
            V ← the variable whose flip yield the most important
                raise in the number of satisfied clauses
            S ← S with V flipped
        end for
    end for
    return the best assignment found
end GSAT
```