Programování s omezujícími podmínkami

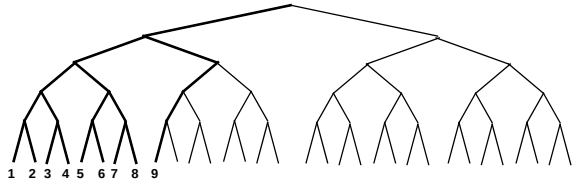
10

Roman Barták
Katedra teoretické informatiky a matematické logiky

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

Prohledávání do hloubky
DFS

Vraťme se k základům: DFS = Depth First Search



Pozorování:

Praktické problémy mají tak velké prostory možných ohodnocení, že není možné je celé prohledat.

Je možné prohledat jen omezený prostor!

↳ **Neúplná stromová prohledávání**

Neúplná prohledávání

Neprohledají celý stavový prostor.

- negarantují dokázání neexistence řešení ani nalezení řešení (úplnost)
  - někdy lze úplnost zaručit, ale při větší časové složitosti

V řadě případů **najdou řešení rychleji.**

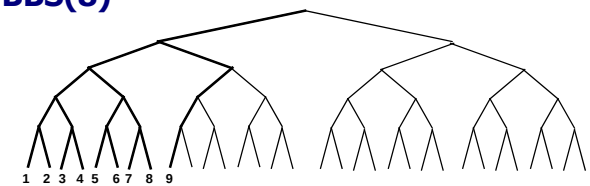
Často **odvozeny od úplného algoritmu** (DFS)

- **přerušení běhu** (cutoff)
  - po vyčerpání přidělených prostředků (čas, návraty, kredit, …)
  - může být globální (pro celý strom) i lokální (pro daný podstrom nebo uzel)
- **opakování běhu** (restart)
  - s jinak nastavenými parametry (např. s více přidělenými prostředky)
  - pro další běh lze použít učící se techniky

Bounded Backtrack Search
BBS

- **Omezený počet návratů** (cutoff)
  - za návrat se nepočítá návrat do bodu, kde už není další alternativa
  - „omezený počet navštívených listů"
- Při neúspěchu **zvětšíme počet návratů o jedna** (restart).

Příklad: BBS(8)



Implementace:

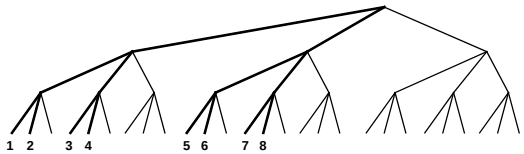
- počítáme počet návratů (neúspěchů)
- při překročení meze se prohledávání ukončí

Iterative Broadening

IB

- **Omezený počet alternativ** (šířka) v každém uzlu (cutoff).
  - v každém uzlu se zkouší jen daný počet hodnot
  - pozor, pořád exponenciální složitost!
- Při neúspěchu **zvětšíme šířku o jedna** (restart).

Příklad: IB(2)



Implementace:

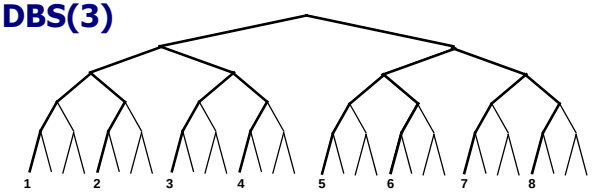
- omezí se počet pokusů v každém uzlu (vnitřní for-cyklus)

Depth Bounded Search

DBS

- **Omezená hloubka** úplného prohledávání (cutoff).
  - do dané hloubky stromu se zkouší všechny alternativy
  - ve zbytku stromu se může použít jiná neúplná metoda
- Při neúspěchu **zvětšíme hloubku o jedna** (restart).

Příklad: DBS(3)



Implementace:

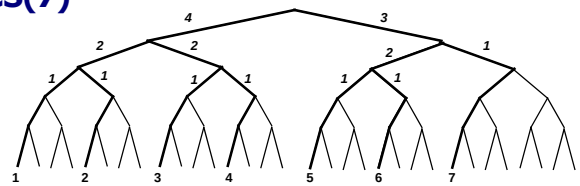
- udržujeme pořadové číslo přiřazované proměnné
- je-li pořadové číslo větší než daná mez, zkouší se pouze jedna alternativa – BBS(0)

Credit Search

CS

- **Omezený kredit** (počet návratů) pro prohledávání (cutoff).
  - kredit se dělí mezi alternativní větve prohledávání
  - jednotkový kredit zakazuje možnost volby (hodnoty)
- Při neúspěchu **navýšíme kredit o jedna** (restart).

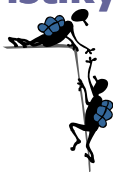
Příklad: CS(7)



Implementace:

- v každém uzlu se nejednotkový kredit (rovnoměrně) rozdělí mezi alternativní podstromy
- při jednotkovém kreditu se neberou alternativy

Prohledávání a heuristiky



- **Při řešení reálných problémů často existuje nápověda odvozená ze zkušeností s „ručním" řešením problému.**

Heuristiky - rádce kudy se vydat při prohledávání

- doporučují hodnotu pro přiřazení (value ordering)
- často vedou přímo k řešení

Co ale dělat, když heuristika neuspěje?

- DFS se stará se hlavně o konec prohledávání (spodní část stromu)
- spíše tedy opravuje poslední použité heuristiky než první
- předpokládá, že dříve použité heuristiky ho navedly dobře

Pozorování 1:

Počet porušení heuristiky na úspěšné cestě je **malý**.

Pozorování 2:

Heuristiky jsou **méně spolehlivé na začátku** prohledávání než na jeho konci (na konci máme více informací a méně možností).

Zotavení se z chyb heuristiky

Jak můžeme udělat prohledávání efektivnější než backtracking?

- Backtracking je „slepý" pokud jde o heuristiky.

Diskrepance = porušení heuristiky (je vybrána jiná hodnota)

Základní principy prohledávání s diskrepancemi:

- změníme pořadí prohledávaných větví podle vlastností heuristik
- **dříve prozkoumáme větve s** méně diskrepancemi

- **dříve prozkoumáme větve s** dřívějšími diskrepancemi

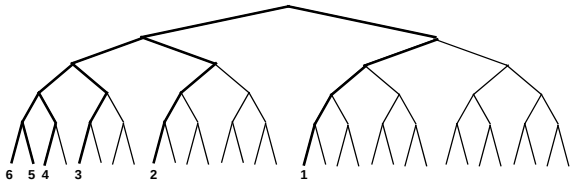
(Harvey & Ginsberg, IJCAI 1995)

Limited Discrepancy Search

LDS

- **Omezený maximální počet diskrepancí na cestě** (cutoff)
  - cesty s diskrepancemi na začátku jsou prozkoumány dříve
- Při neúspěchu **navýšíme počet povolených diskrepancí** o jedna (restart).
  - tj. nejprve jdeme tak, jak doporučuje heuristika
  - potom jdeme po cestách s maximálně jednou diskrepancí

Příklad: LDS(1), heuristika doporučuje vydat se levou větví



Poznámka pro **nebinární domény**:

- nedoporučené hodnoty se berou jako **jedna diskrepance** (zde)
- každá další nedoporučená hodnota se bere jako **navýšení diskrepance** (tj. třetí hodnota = dvě diskrepance)

(Harvey & Ginsberg, IJCAI 1995)

Algoritmus LDS

```
procedure LDS-PROBE(Unlabelled,Labelled,Constraints,D)
    if Unlabelled = {} then return Labelled
    select X in Unlabelled
    Values_X ← D_X - {values inconsistent with Labelled using Constraints}
    if Values_X = {} then return fail
    else select HV in Values_X using heuristic
        if D>0 then
            for each value V from Values_X-{HV} do
                R ← LDS-PROBE(Unlabelled-{X}, Labelled ∪ {X/V}, Constraints, D-1)
                if R≠ fail then return R
            end for
        end if
        return LDS-PROBE(Unlabelled-{X}, Labelled∪{X/HV}, Constraints, D)
    end if
end LDS-PROBE
procedure LDS(Variables,Constraints)
    for D=0 to |Variables| do        % D určuje max. počet povolených diskrepancí
        R ← LDS-PROBE(Variables,{},Constraints,D)
        if R≠ fail then return R
    end for
    return fail
end LDS
```


(Korf, AAAI 1996)

Improved LDS

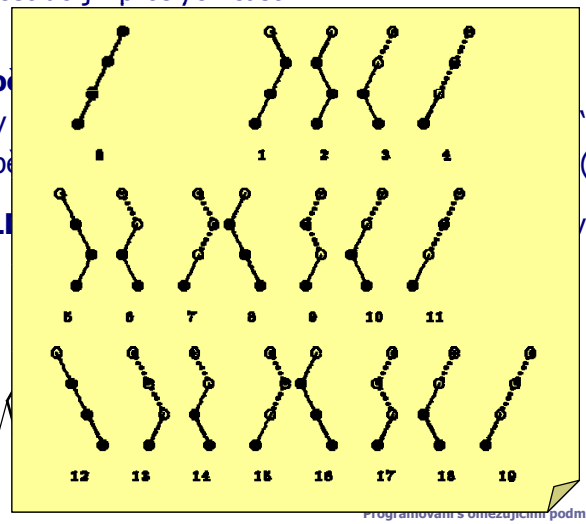
ILDS

LDS v každé další iteraci **prochází i větve z předchozí iterace**, tj. opakuje již provedený výpočet a navíc se v rámci iterace musí vracet do již prošlých částí.

↳ **ILDS:**

- **Daný po**
  - „cesty
- Při neúspě (restart)

Příklad: ILI větví



(Korf, AAAI 1996)

Improved LDS

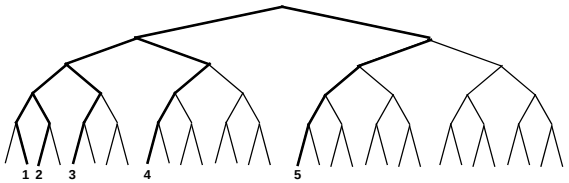
ILDS

LDS v každé další iteraci **prochází i větve z předchozí iterace**, tj. opakuje již provedený výpočet a navíc se v rámci iterace musí vracet do již prošlých částí.

↳ **ILDS:**

- **Daný počet diskrepancí na cestě** (cutoff)
  - „cesty s diskrepancemi na konci prozkoumány dříve"
- Při neúspěchu **navýšíme počet diskrepancí** o jedna (restart)

Příklad: ILDS(1), heuristika doporučuje vydat se levou větví



(Korf, AAAI 1996)

Algoritmus ILDS

```
procedure ILDS-PROBE(Unlabelled, Labelled, Constraints, D)
    if Unlabelled = {} then return Labelled
    select X in Unlabelled
    Values_X ← D_X - {values inconsistent with Labelled using Constraints}
    if Values_X = {} then return fail
    else select HV in Values_X using heuristic
        if D<|Unlabelled| then
            R ← ILDS-PROBE(Unlabelled-{X}, Labelled∪{X/HV}, Constraints, D)
            if R≠ fail then return R
        if D>0 then
          for each value V from Values_X - {HV} do
            R ← ILDS-PROBE(Unlabelled-{X}, Labelled∪{X/V}, Constraints, D-1)
            if R≠ fail then return R
          end for
    end if
end ILDS-PROBE

procedure ILDS(Variables,Constraints)
    for D=0 to |Variables| do        % D určuje max. počet povolených diskrepancí
        R ← ILDS-PROBE(Variables,{},Constraints,D)
        if R≠ fail then return R
    end for
    return fail
end LDS
```

Rozdíl oproti LDS

(Walsh, IJCAI 1997)

Depth-Bounded Discr. Search

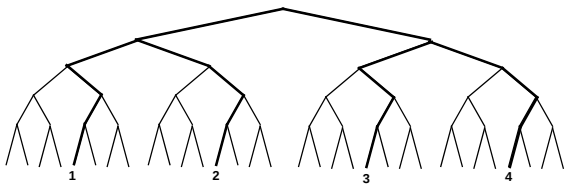
DDS

ILDS bere cesty s diskrepancemi na konci dříve.

↳ **DDS:**

- **Diskrepance povoleny pouze do dané hloubky** (cutoff)
  - v této hloubce musí být vždy diskrepance čímž se zabrání procházení větví z předchozí iterace
  - hloubka zároveň omezuje počet diskrepancí
  - cesty s diskrepancemi na začátku prozkoumány dříve
- Při neúspěchu **navýšíme hloubku o jedna** (restart)

Příklad: DDS(3), heuristika doporučuje vydat se levou větví

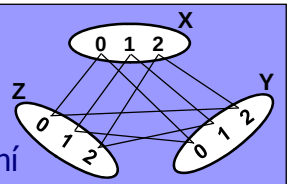


Enumerace – ano či ne?

- Dosud jsme předpokládali, že prohledávací algoritmus dělá tzv. labelling, tj. přiřazování hodnot proměnným.
  - přiřadíme hodnotu, propagujeme a v případě neúspěchu zkusíme přiřadit další hodnotu
    - proto se tato technika nazývá **enumerace**
  - propagace se využije pouze po přiřazení hodnoty

Příklad:

- X,Y,Z in 0,…,N-1 (N je konstanta)
- X=Y, X=Z, Z=(Y+1) mod N
  - problém je AC, ale nemá žádné řešení
  - enumerace bude zkoušet všechny hodnoty
  - pro $n=10^7$ běží 45 s. (na 1.7 GHz P4)



Šlo by labelling udělat lépe?

Další větvící strategie

- **Enumerace** vlastně řeší disjunkce tvaru X=0 ∨ X=1 … X=N-1
  - pokud přiřazení neexistuje, musí vyzkoušet všechny hodnoty
- **Můžeme využít propagaci i pokud zjistíme, že hodnota není v řešení!**
  - příslušnou hodnotu prostě z domény smažeme, což vyvolá propagaci, která dále ořeže domény
  - řešíme disjunkce tvaru X=H ∨ X≠H
  - jedná se o tzv. **step labelling** (bývá implicitní/default strategií)
  - příklad z předchozího slajdu vyřeší za 22 s. pouze vyzkoušením a zamítnutím hodnoty 0 pro X

    Proč tak dlouho?
    - V každém cyklu AC algoritmu se odstraní právě jedna hodnota.
- Další běžné větvení je půlení domény (**bisection/domain splitting**)
  - řešíme disjunkce tvaru X≤H ∨ X>H, kde H je hodnota uprostřed domény

Řešící techniky v kostce

Problémy s omezujícími podmínkami můžeme řešit:

- **lokálním prohledáváním**
- **kombinací prohledávání** do hloubky a hranové **konzistence**

Poznámky ke **konzistenčním technikám**:

- Hranová konzistence **slouží k** prořezání **prohledávaného stromu.**
- **Obecně** silnější konzistence odstraní více větví stromu**, které nevedou k řešení.**
- Silnější konzistence **od PC jsou ale** výpočtově drahé!

Optimalizace podmínek

- Dosud jsme hledali libovolné řešení splňující všechny podmínky.
- Často je potřeba najít optimální řešení, kde kvalita je definována objektivní funkcí.

Definice:

- Problém optimálního splňování podmínek (Constraint Satisfaction Optimisation Problem - CSOP) se skládá z CSP problému P a objektivní funkce *f* mapující řešení P na čísla.
- Řešením CSOP je řešení problému P minimalizující / maximalizující hodnotu funkce *f*.
- Pro řešení CSOP obecně potřebujeme procházet všechna řešení, tj. používají se metody schopné poskytnout všechna řešení CSP.

Metoda větví a mezí

- **Metoda větví a mezí** (branch and bound) je nejběžnější optimalizační technika založená na ořezávání větví, ve kterých se nenachází řešení.
- Používá **heuristickou funkci** h odhadující objektivní funkci.
  - korektní heuristika pro minimalizaci splňuje h(x) ≤ f(x)

    [korektní heuristika pro maximalizaci splňuje f(x) ≤ h(x)]
  - lepší je heuristika bližší k objektivní funkci
- Při prohledávání stromu řešení je **podstrom uříznut**, pokud:
  - v podstromu **neexistuje žádné řešení**
  - v podstromu **není optimální řešení**
    - mez ≤ h(x), kde mez je maximální hodnota přípustného řešení

Jak získáme mez?

- například hodnota dosud nejlepšího řešení

BB a splňování podmínek

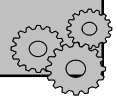
S objektivní funkcí pracujeme jako s podmínkou

„optimalizujeme" hodnotu proměnné v, kde v = f(x)

- první řešení získáme bez omezení proměnné v
- po dalších řešeních požadujeme, aby byla lepší (v < Bound)
- opakujeme, dokud nalézáme lepší řešení

Algoritmus Branch & Bound

```
procedure BB-Min(Variables, V, Constraints)
    Bound ← sup
    NewSolution ← fail
    repeat
        Solution ← NewSolution
        NewSolution ← Solve(Variables,Constraints ∪ {V<Bound})
        Bound ← value of V in NewSolution (if any)
    until NewSolution = fail
    return Solution
end BB-Min
```



Poznámky k BB

- Heuristika h je ukryta v propagaci podmínky v = f(x).
- Efektivita řešícího algoritmu je závislá na:
  - dobré heuristice (propagace přes objektivní funkci)
  - včasném nalezení dobrého řešení
    můžeme pomoci přesnější počáteční mezí
- Nalezení optimálního řešení může být rychlé
  - dlouho trvá důkaz optimálnosti (musí se projít zbylá řešení)
- Většinou nepotřebujeme optimální řešení, stačí dobré řešení
  - BB můžeme ukončit po dosažení stanovené meze
- Zrychlení BB získáme zpřesňováním dolní a horní meze půlením.

```
repeat
    TempBound ← (UBound+LBound) / 2
    NewSolution ← Solve(Variables,Constraints ∪ {V≤TempBound})
    if NewSolution=fail then
        LBound ← TempBound+1
    else
        UBound ← TempBound
until LBound = UBound
```