Herní algoritmy, předn. 4

Proof number search
Lambda search

Jan Hric, Petr Baudiš

NAIL103, ZS

20. listopadu 2013

Obsah

1. Úvod
   - Úvod
2. PNS alg.
   - Idea
3. PNS varianty
   - Varianty
4. Depth first PNS
   - DF-PNS
5. Další PNS
6. Lambda search
   - Lambda search

Proof number search, úvod

- Používá se typicky pro hledání (úplné) útočné nebo obranné strategie, tj. dokazování, zda je pozice vyhraná
- Idea: některé větve mají mnohem lehčí (tj. menší) důkaz
- Hodí se uspořádaní tahů
- Uniformní prohledávání do stejné hloubky (jako alfabeta) má nevýhody. Hluboké, ale většinou *vynucené* větve lze dokázat snadněji.
- V mnoha hrách, větvící faktor není uniformní:
  - šachy: král musí uniknout z šachu, jinak prohra
  - dáma: skákat se musí (v pravidlech). Taky to zjednodušuje pozici.
  - go: u kamenů ”blízko” života stačí spočítat málo útočných tahů, ostatní jsou neúčinné
- důsledek: omezený větvící faktor, zvyšuje se hloubka prohledání

PNS

- PNS: Victor Allis 1994; předtím McAllester: *conspiracy numbers*, min. počet vrcholů, které ”změní” hodnotu kořene
- najde důkaz anebo vyvrácení pro pozici
- konstruuje oba stromy současně (proof, disproof), rozvíjí vždy jeden (nejpřínosnější) uzel
- alg. v průběhu výpočtu některé uzly dokáže anebo vyvrátí, ale většina uzlů je ve stavu *neznámý* (...)
- alg. končí, až je kořen známý, tj. *dokázaný* nebo *vyvrácený*
- v principu: informované prohledávání do šířky, nemusí najít nejmenší strom

PNS (2)

- Mějme průběžný neúplný strom: jak daleko jsme od dokázání/vyvrácení? (Výsledek nevíme dopředu.)
- Najdeme minimální (proti-)důkazovou množinu ((dis-)proof set): množina vrcholů, které musí být dokázány/vyvr., aby jsme dok./vyvr. kořen
- Princip: optimizmus při neurčitosti (bude i v MCTS)
- Předpokládejme cenu neznámeho uzlu 1 - to je dolní mez
- Důkaz máme, když má důkazová množina velikost 0. (analogicky pro vyvrácení)
- Idea: expandujeme vrchol z min. důkazové a min. vyvracecí množiny (analogie PV)

Nejslibnější uzel

- Klíčová vlastnost PNS: vždy existuje ve stromě nejslibnější uzel (Most-Promising Node, MPN)
- MPN je v průniku minimální důkazové a minimální vyvracecí množiny
- Vyřešení MPN pomůže při důkaze nebo vyvracení, protože zmenší velikost příslušné minimální množiny pro kořen
- V průběhu alg. po rozvinutí jednoho MPN velikosti min. množin typicky vzrostou (kromě koncových pozic).
- Pozn: MPN není jednoznačný, což nevadí.

PNS algoritmus

- Proof number vrcholu *n*: velikost minimální důkazové množiny pro *n*. Optimistický odhad nejkratšího důkazu *n*.
- Disproof number: velikost minimální vyvracecí množiny pro *n*. Optimistický odhad nejkratšího vyvrácení *n*.
- PNS alg.: najdi MPN a expanduj. Přepočítej hodnoty PN a DN. Cykli, dokud není kořen dokázaný nebo vyvrácený (a jsou zdroje).
- (MPN skáče mezi podstromy; podobně jako expandovaný uzel v MCTS)

Proof a Disproof numbers v listech

- Značení: n.pn = proof number vrcholu n
- Značení: n.dn = disproof number vrcholu n
- List, výhra: n.pn = 0, n.dn = ∞
- List, prohra: n.pn = ∞, n.dn = 0
- List, neznámý: n.pn = n.dn = 1
- Výhra a prohra: podle pravidel hry

Proof a Disproof numbers ve vnitřních vrcholech

- připomenutí: pro důkaz kořene potřebujeme dokázat jednoho syna v OR-uzlu a všechny syny v AND-uzlu (OR=MAX)
- počítání čísel zdola; když máme hodnoty v synech, pak
- pro OR-uzel je PN *minimum* z PN-hodnot synů a DN je *součet* DN-hodnot všech synů
- pro AND-uzel duálně: PN je součet a DN je minimum
- předpoklad: podstromy se řeší nezávisle (neplatí vždy (tj. použitelné na DAG/DCG); způsobí neoptimalitu (až přetečení), ale ne chybný výsledek)

Vzorce

Backup fáze (terminologie: přepočítání?), zdola

- OR-uzel: n.pn = $\min_{s \in synove(n)} s.pn$
- OR-uzel: n.dn = $\sum_{s \in synove(n)} s.dn$
- AND-uzel: n.pn = $\sum_{s \in synove(n)} s.pn$
- AND-uzel: n.dn = $\min_{s \in synove(n)} s.dn$
- operace s ∞: $\infty + \infty = \infty + c = \infty$, $\min(c, \infty) = c$...

Dokázané a vyvrácené vrcholy

- Dokázaný vrchol n: n.pn = 0, n.dn = ∞
- Vyvrácený vrchol n: n.pn = ∞, n.dn = 0
- Výhry a prohry se propagují podle očekávání:
- Pro OR-uzel syn s_i je výhra:
  - n.pn = $\min(s_1.pn, ...s_i.pn, ...) = min(..., 0, ...) = 0$
  - n.dn = $\sum(s_1.dn, ...s_i.dn, ...) = \sum(..., \infty, ...) = \infty$
- Pro AND-uzly duálně, tj. prohodíme .pn a .dn

PNS impl.

- základní impl.: informovaně do šířky, uzly v paměti
  - pro OR-uzly podle min PN
  - pro AND-uzly podle min DN
- optimizace: při přepočítávání zdola můžeme zastavit, pokud se hodnoty pn a dn nemění (nastává, když víc synů mělo stejné pn, resp. dn). Výběr MPN spustíme odtud.

Analýza

- pro jedno rozvinutí MPN: (bez optimalizací)
- Výběr MPN: hloubka MPN $\times$ prům. faktor větvení
- Úprava PNs: stejné
- Expanze MPN: počet synů $\times$ cena vytvoření a inicializace uzlu
- Paměť na iteraci: počet synů $\times$ velikost uzlu
- Pokud je strom vyvážený a faktor větvení omezený konstantou: hloubka je $O(\log n)$ pro n uzlů, čas $O(\log n)$, $O(1)$ paměti za iteraci
- Celkový čas pro strom s n uzly, omezené větvení: $\Theta(n \log n)$ pro vyvážený strom, $\Theta(n^2)$ pro degenerovaný (malé větvení, hluboké větve)

PNS, komentáře

- Best-first, vhodný pro nevyvážené stromy
- Adaptuje se na hluboké, ale úzké důkazy
- Nezaručuje krátkou výhru a malý strom důkazu - ignoruje cenu dosud vygenerované části
- Chová se spíš jako (lokální) ”čistě heuristické prohledávání” z jedno-agentového prohledávání stavového prostoru než jako optimální A* (např. nejlepší řešení může být ”schované” za jedním nevynuceným, přípravným tahem)
- Existuje alg. AO*, ekvivalent k A*, pro hledání nejlepších řešení v AND/OR stromech. (Ale v obecnosti chceme hledat v grafech, typicky acyklických)

PNS, komentáře 2

- náročný na paměť, všechny uzly jsou v paměti (depth first PNS to odstraňuje)
- dokázané a vyvr. stromy lze vypouštět z paměti (pokud si je nechci uschovat/prohlížet; můžu vypsat do logu)
- lze vypouštět nekritické větve
- kompromis: pro účely znovuprohledání si pamatuju PN a DN kořene vypouštěné větve
- líné vypouštění, když dochází paměť
- (viz alg. SMA*, analogie k A*, ale s omezenou pamětí)

Použití

- na vyřešení her: ..., piškvorky
- na koncovky; ... např. shogi (japonské šachy na 9x9)
- pro praktické použití potřeba přeformulovat do depth first varianty
- nezávislý na hře: nepotřebuje heuristiku (řídí se stromem), ale nepodporuje přímou integraci heuristiky (viz)
  - některé hry (např. Arimaa) nemají významně různý počet tahů pro dobré a špatné pozice

PNS na DAG

víc realistická varianta

- je dobře definovaný, počítání hodnot zdola
- Problém 1: vrchol má víc než jednoho rodiče
- Problém 2: nadhodnocení PN a DN

na DAG, víc rodičů

Problém 1: vrchol má víc rodičů

- Přepočítání všech? Cena se zvýší, nejhůř z $\Theta(\log n)$ na $\Theta(n)$ na iteraci
- Přepočítání pouze jednoho? Hodnoty se rozsynchronizují a MPN bude nepřesně počítán
- Obvykle: přepočítání jednoho rodiče, ostatní až/pokud jsou zpracovány
- Otevřený problém: jde to lépe?

na DAG, nadhodnocení

Problém 2: nadhodnocení PN a DN

- připomenutí: PN a DN odhadují min. počet listů, které se musí vyřešit
- v DAG se stejný list může započítat po několika cestách
- toto nadhodnocení může být exponenciálně horší
- důsledek: lehký vrchol vypadá (velmi) těžce; následně vrcholy mimo dis/proof set musíme rozpracovat o (hodně) víc než při těsnějších mezích
- vyskytuje se v praxi: tsume shogi (úlohy, (dlouhé) koncovky), go
- možnosti: Proof-set search (Nagai), úpravy df-pns

Heuristická inicializace

- Heuristická inicializace pn a dn (stejně nepočítá nejmenší stromy)
- Připomenutí: pn a dn jsou dolní meze pro cenu vyřešení uzlu
- Inicializace hodnotou 1 je naivní
- Lze najít lepší odhad? Závisí na pozici. Lze použít doménovou znalost, např. bezpečnost krále, go: ”vzdálenost k životu”
- Jsou pn a dn nepřímo úměrné? Ne zcela.
- Významné zlepšení v praxi
- Pouze *částečná* a nepřímá heuristika, netýká se stromu
- Otevřená otázka: lze použít strojové učení? A jak?

Vylepšení: výhled jednoho tahu

- Jak dopadne expanze jednoho uzlu? V případě, že se ho nepodaří vyřešit.
  - OR uzel: pn = 1 před a po, dn = 1 před a počtu dětí po
  - AND uzel: dn = 1 před a po, pn = 1 před a počtu dětí po
- v mnoha hrách je laciné spočítat nebo odhadnout(!) počet následníků; go: počet volných průsečíků je dobrý odhad
- inicializujme dn v OR- a pn v AND-uzlu odhadem počtu dětí
- v podstatě to znamená, máme výhled o 1 tah dál
- Jak tato inicializace interaguje s jinými (např. doménovými) metodami?

Vyhodnocení listu

okamžité vs. odložené ohodnocení; immediate resp. delayed evaluation

- *Okamžité ohodnocení* : rozvinu list a nové syny hned ohodnotím
- mám víc informací, tj. řízení je lepší
- *Odložené ohodnocení* : list se ohodnotí, až je vybrán jako MPN
- za speciálních okolností, viz př.
- - může být několik ”úrovní” vyhodnocení: 1. rychlé, nepřesné; 2. přesné
- Př, v PN^2: PN_2 má okamžité ohodnocení, PN_1 odložené (ale pamatuju si dolní odhady PN a DN)

Rozšíření na vícehodnotový případ

např. výhra - remíza - prohra; go: život, seki a (různé druhy) ko, přesné počítání koncovek

- Série boolovských hledání; např. binární nebo sekvenční hledání
- Každé hledání (průchod) používá mez pro listy, podobně jako hledání s nulovým oknem
- Otevřený problém: Jak využít stromy důkazu z minulých průchodů?
  - Lehký případ: minulé hledání bylo s přísnějšími mezemi než aktuální
  - Těžký případ: nový cíl je těžší

Případová studie: Dáma/Checkers

- Řešení dámy, Schaffer a kol.
- Použití *jádra* (seed) pro generování stromu: nejlepší varianty jsou navrženy lidskými experty
- Pseudo-důkaz: předpokládejme, že všechno se (statickým) ohodnocením > 150 je výhra, < -150 je prohra. Vytvoří se strom.
- Až se strom s mezí ± 150 vybuduje, zvýší se mez na ± 200, atd. Až mez dosáhne $\pm\infty$, máme úplný důkaz.

Případová studie: Dáma/Checkers (2)

- Pro rozšiřování stromu lze použít *simulace* už hotových stromů - stejný tah/podstrom zkoušíme u bratra. Navíc nejlepší tahy už typicky máme: jádro, eval. okolo 0.
- Listy v pseudodůkazu: v koncovkách je strom malý (resp. použijeme endgame tabulky)
- Listy uvnitř: máme problém, pokud pseudolist *n* je ”výhra”, ale pozice je reálně prohra - pak musíme najít jiný důkaz otce (nebo vzdálenějšího předchůdce) *n*. Málo časté, pouze když heur. fce je nepřesná
- Speciálně: v dobrých pozicích můžu udělat horší tah, abych se dostal do již dokázané části stromu, resp. na položky TT (kde je eval. okolo 0).

Předchůdce PNS: Konspirační čísla pro AB

- navrženo McAllister 1985, 1988
- Kolik vrcholů v Alfabetě se musí dohodnout, aby změnili minimaxovou hodnotu kořene?
- Připomenutí: PV není jedinečná, musí se změnit vrcholy ve ”všech” PV (v nichž platí $val = PV.val$)
- Zjednodušení, omezení na boolovský případ vedlo V. Allise k PNS

PNS: dodatky

- dobré a špatné příklady pro PNS
- správa paměti a garbage collection
- dvojúrovňový PNS - PN^2 (jedno z prvních vylepšení)
- negamax formulace PNS
- úprava na (iterované) prohledávání do hloubky - df-pns
- prohledávání s omezenou šířkou: case study: hex; použitelné obecně, hodí se uspořádání tahů (tj. integrace heur./doménové info)
- (další: PN*, PDS, λPNS, EF-PN (evaluation function-based))

Chování

Dobré případy

- nerovnoměrné větvení
- rychlé důkazy/vyvrácení pro některé větve (chybu soupeře rychle potrestáme, máme jádro pro simulace)
- počet tahů koreluje s pravděpodobností výhry; šachy: málo obranných tahů (při šachu) může znamenat král v problémech

Špatné případy

- ”všude to vypadá stejně”
- uniformní větvení, bez rychlých výher/proher; přípravné tahy
- v podstatě: drahý způsob pro neinformované prohledávání

PNS :-(

Velmi špatné případy

- Když PNS aktivně ”zavádí na scestí”
- Př: krátký důkaz potřebuje 3 uzly, ale existuje velký strom s $pn \leq 2$
- Mnoho vynucovacích tahů, které nefungují. Funguje pouze ”tichý” tah.
- Go: větvící se schody, všechny větve selžou, ale ”gheta” (síť) funguje
- př. Arimaa: zmrazení nebo braní (nepodstatných) figur soupeře (nejčastěji je cíl dosažen vlastní volností (goal) a ne omezením soupeře (swarm a bez tahu; bez králíků))
- Q: vztah k problému horizontu?

Dvojúrovňový PNS - PN^2

- První PNS běží v kořeni
- V listech běží druhý PNS, omezený (prahem, pamětí): jeho informace se propagují nahoru
- Strom druhého PNS se zahazuje - šetří to paměť, ale případně plýtvá zdroji. Např. se nechávaní synové druhotného kořene v prvním stromu
- Nevýhoda: Kořenový PNS je celý v paměti
- impl.: pro M = velikost paměti a x = # uzlů v hlavním stromě, druhotný strom rozvine $min(M, 1 + e^{\frac{x-a}{b}})$, vhodné a, b (b určuje kvocient, a bázi)
- $\rightarrow$ druhotné stromy rostou zhruba geometrickou řadou (”kapitánský krok”), díky přeskakování MPN prozkoumáváme různé větve se zvyšujícím se úsilím

Negamax formulace PNS

- $n.\phi = n.pn$ v OR-uzlu, $n.dn$ v AND-uzlu
- $n.\delta = n.dn$ v OR-uzlu, $n.pn$ v AND-uzlu
- Vzorec pro výpočet: $n.\phi = \min(s_1.\delta, ..., s_k.\delta)$
- Vzorec pro výpočet: $n.\delta = \sum(s_1.\phi, ..., s_k.\phi)$
- Důsledek: z hlediska výpočtu nerozlišuju OR a AND uzly

Úprava na prohledávání do hloubky

- Depth-first verze PNS: df-pn (následně několik variant alg.)
- Nagai, 2002
  - df-pn se chová jako PNS, expanduje MPN
  - pro úplnost: obecně je víc MPN, výběr z nich se může lišit
  - základní verze df-pn: pouze iterace v kořeni
- Snižuje opakované expanze vnitřních vrcholů
- Používá TT, chová se dobře i s malými tabulkami
- df-pn(r): Kishimoto, Müller: úprava df-pn pro hry s opakováním (dáma, ko v go), tj. DCG, nejen DAG (impl. počítá min. vzdálenost od kořene)
- předchůdci: PDS: iterativní prohlubování (pouze) v kořeni, tj. zvyšování prahů; PN*: zvyšování pouze pro pn; ...

Přínosy Df-pn

- Efektivita
- Paměť
- Opakované iterativní prohlubování ve vnitřních vrcholech
- Rozšíření pro vyhnutí se dvojitému počítání v DAG
- Rozšíření v df-pn+ umožní ocenit hrany, najde optimální řešení v And/Or grafech (jako AO*)

Df-pn: Idea pro výpočet Min

- v PNS, hledání často ”uvízne” v jednom podstromě na dlouhou dobu
- pokud víme určit MPN (most proving node), nezajímají nás konkrétní pn a dn čísla - lze odložit backup
- př: n.pn = min(80,70,**20**,65,*50*) = 20
- lokálně, zůstaneme v podstromě s pn = 20, dokud jeho pn (ostře) *nepřevýší* druhé nejmenší pn_2 mezi syny, tj. 50 v př.
- globálně, musíme kontrolovat zda se volba větve nezmění někde výš ve stromě. To lze poslat z rodiče dolů jako práh.
- Vzorec pro nový práh *pn*: min($parent.pn$,$pn_2 + 1$)

Df-pn: Idea pro výpočet Sum

- $n.pn = \sum(s_1.pn, ...s_k.pn)$
- Máme práh pro vrchol n, $n.mezpn$
- Q: Jak dlouho budeme pracovat v s_i?
- A: Dokud $n.pn \geq n.mezpn$ anebo zvýšení v s_i nepřevýší rozdíl $n.mezpn - n.pn$.
- Vzorec: $s_i.mezpn = s_i.pn + (n.mezpn - n.pn)$
- tj. vyčerpali jsme rezervu, je jedno, kde (z hlediska důkazu)

Opakované iterativní prohlubování

Multiple Iterative Deepening (MID)

- Iterativní ”prohlubování” v každém vrcholu, nejen v kořeni
- Prohloubení znamená zvýšení *prahu*, nikoli hloubky
- Vhled: pn a dn se můžou i snížit, nejen zvýšit (když se dokáže syn nějakého sum-uzlu)
- df-pn volá z kořene MID s prahy $\pm\infty$
- používá TT

df-pn+

Myšlenky:

- Heuristická inicializace v listech: určuje obtížnost důkazu/vyvrácení pro list
- Ohodnocení hran
  - Měří ”dychtivost” zkoumání tohoto tahu
  - vysoká cena hrany: preferuj plytké stromy, víc do šířky
  - nízká cena: preferuj tuto větev

Šetření pamětí

- Běží s malou spotřebou paměti
- Prořezávání (Nagai): SmallTreeGC, SmallTreeReplacement
- SmallTreeGC: GC se předají vrcholy s malými podstromy
- SmallTreeReplacement: Hašování pomocí otevřené adresace, vyzkoušej pár (např. 10) položek a nahraď tu s nejmenším stromem
- Jiná možnost: Hašování se zřetězením - ukládá se víc než 1 položka na 1 adresu (i zde potřebuje uvolňovat)
- DC impl.: jak uvolňovat globálně nejmenší? Chceme on-line, jako side-efekt (plovoucí mez)

Weak PNS

- Pokud pracujeme na DAG, PNS nadhodnocuje pn a dn, protože jeden vrchol je započítán po několika cestách
- Weak PNS odstraňuje tuto nevýhodu: místo sumy pro dn v OR- (a analogicky pn v AND-uzlu) počítá
- $dn(n) = \max dn(s) + n.nevyresenySyn - 1$ pro OR-uzel
- tj. další děti, kromě nejtěžšího, započítá pouze hodnotou 1

Focused DFPNS, FDFPNS

- PNS prohledává všechny uzly, neprořezává jako Alfabeta
- Když hra neposkytuje vodítko, prohledávám i zbytečné větve (které nejsou ve výsledném stromě; typicky na začátku)
- Př: **Hex**: ve stejné hloubce stejné větvení; Arimaa: velké větvení řádu 10^4 (protože v tahu mám až 4 kroky za sebou)
- Idea: omezit šířku, tj. počet aktivních synů jednoho uzlu.
- Pokud se některého syna podaří dokázat, další ještě nezpracovaný syn se stane aktivní
- Využívá uspořádání, preferuje nejlepší uzly (!za oba, tj. nejtěžší, nejúčinnější obranu). (Ostatní chci přeskočit, ne (projít a) vyvrátit)
- Q: Integrace s jinými technikami

Další

- PNS skáče mezi větvemi, to není kompatibilní s inkrementální úpravou d.s./stavových informací. Např. pro Hex se engine bez inkrementálního počítání 2*x* spomalil
- Lambda DFPN, LDFPN - kombinace PNS a lambda stromů: pomocí PNS řeší λ_i-strom pro nejmenší *i*
- $1 + \epsilon$ trick, Lew Lukasz 2006: prahy v DFPNS zvětšuju geometrickou řadou, to zmenšuje počet přeskoků mezi podstromy, které se případně nevejdou do TT
  - (použitelné i v MCTS: spočítat pevnou část (např. 1/10) počtu návštěv v akt. uzlu a neopouštět uzel, dokud se nevykoná aspoň tolik playoutů v podstromě)
- Q: srovnání s SMA* (Simplified Memory-Bounded A*) - explicitně zahazuje nejméně slibné vrcholy a pamatuje si dolní odhady uvolněných vrcholů

Lambda Search

- Allis a kol. 1996, Thomsen 2000, Cazenave 2002
- Idea: používame hrozby různého řádu *n*
- Řád hrozby neformálně: kolikrát může soupeř vynechat tah (tj. null move/pas) než prohraje
- V λ^n-stromě se prohledávají pouze hrozby řádu *n*, bez ohledu na hloubku stromu.
- Pokud ve vrcholu není λ^n-tah, vrchol je listem, prohraným (pro hráče na tahu)
- Rozlišujeme útočníka, který hraje v kořeni a snaží se o *splnění cíle*, a obránce (attacker, defender) - nesymetrické role
- příklad pro představu: piškvorky

Lambda Strom

- Značení: λ^n=1, pokud útočník může vyhrát v dané pozici pomocí hrozby řádu n, jinak λ^n=0
- Rekurzivní definice λ^n-stromu, podle hloubky:
- Úspěšný λ^0-strom pro útočníka: obsahuje jeden λ^0-tah, který splní cíl
- λ^n-strom obsahuje λ^n-tahy, listy jsou prohrané pro hráče na tahu (tj. nemá λ^n-tah)
- λ^n_a-tah pro útočníka je tah, že pokud obránce pasuje, existuje λ^i-strom s λ^i=1 pro $0 \leq i \leq n-1$. Tj. útočník hrozí vyhrát hrozbou nižšího řádu
- λ^n_d-tah pro obránce je tah, že po něm neexistuje žádný λ^i-strom s λ^i=1 pro $0 \leq i \leq n-1$. Tj. obránce odvrátil všechny hrozby nižšího řádu
- Př: Go: schody: λ^1_a- a λ^1_d-tahy; při neúspěšných schodech v listě neex. další λ^1_a-tah

Lambda Search

- Důsledek: λ^n tahy jsou určeny λ^{n-1}-prohledáváním.
- Obvykle je λ^{n-1} strom mnohem menší než λ^n-strom
- Následně λ-prohledávání prořeže tahy podle λ úrovně účinně, s malým úsilým

Použitelnost

- λ-search je nesymetrický, cíl je pro útočníka
- použitelné na globální i lokální cíle
- integrovatelné se znalostmi pro konkrétní hru: Hex, Havannah, Piškvorky, Go
  - (vše jsou nepohyblivé, ”spojovací” hry)
  - řád hrozby se dá odhadnout z pozice (piškvorky, svobody v go)
  - i (ne-)relevantní útočné a obranné tahy
  - (nejen ”nepohyblivá”) hra dovoluje hledat místa, které jsou relevantní (př: tahy dvojího účelu)
  - (impl: pro vyloučení tahů Ú/O potřebujeme opačné meze odhadu)

Vlastnosti

- Vlastnosti, vztahy:
- Pokud λ^n=1, pak λ^k=1 pro $n \leq k$
- Pokud λ^n=0, pak λ^i=0 pro $0 \leq i \leq n$
- Korektnost:
- Výsledek λ^n=1 je korektní, pokud hra má tah pas nebo zugzwang není cílem hry/úlohy (!, neformálně)
- λ^n=0 znamená, že řešení neexistuje anebo je na vyšší úrovni hrozby $n' > n$

Lambda Search, Doplňky

- Zobecněné hrozby: Cazenave 2002 - počítá uzly na jednotlivých úrovních
- Simulace: Kawano 1996 (PNS search) - využívání informací z prohledaných částí stromu (!)
- ($\alpha - \beta$ jako driver pro PNS)

Dual Lambda Search

- Dual lambda search: Soeda 2005 (μ-search): (tsume shogi)
- motivace: původní λ-search je nesymetrický. Soupeř má taky cíl
- Idea: Vyhrává ten, kdo má hrozbu nižšího řádu - hrozby soupeře musím nejdřív odvrátit
- λ^n-stromy útočníka nesmí obsahovat úspěšný strom (ostře) nižšího řádu pro soupeře
- Při prohledávaní musím odvracet hrozby soupeře: širší strom
- Při rovnosti řádu hrozeb výhoda tempa. Př: piškvorky; go: semeai (boj)