Submodularity and meta-reasoning in Monte-Carlo Tree Search

Solomon Eyal Shimony

Ariel Felner, David Tolpin, Shahaf Shperberg

E. S. Shimony: General Research Topics

1. Graphical probability models
2. Probabilistic reasoning
3. Decision-making under uncertainty and applications
   - Robotics: formal plans, real-time decision making
   - Optimal information gathering in graphical models
   - AI in games
   - Metrology: setup, learning
   - Smart grid: electrical load optimization
4. Meta-reasoning in search
   - Deciding which heuristics to compute when in A*, IDA*, CSP
   - Node selection in MCTS (this talk)

Outline

Motivation

- Monte-Carlo tree search (MCTS) achieves major improvements in hard search problems.
- Many well-known success stories, such as Go, Poker, Canadian traveler problem, real-time strategy games.
- The most commonly used MCTS are UCT and its derivatives.
- In fact, UCT is based on a completely irrelevant "exploration-exploitation" **cumulative regret** criterion (as are almost all its variants).
- Using the more relavant **simple regret** criterion should (and does) improve the state of the art in MCTS.
- We show how to "do the right thing" (in the terms of (Russell and Wefald 1991)) in selecting nodes for sampling in MCTS.

Motivation

- Monte-Carlo tree search (MCTS) achieves major improvements in hard search problems.
- Many well-known success stories, such as Go, Poker, Canadian traveler problem, real-time strategy games.
- The most commonly used MCTS are UCT and its derivatives.
- In fact, UCT is based on a completely irrelevant "exploration-exploitation" **cumulative regret** criterion (as are almost all its variants).
- Using the more relavant **simple regret** criterion should (and does) improve the state of the art in MCTS.
- We show how to "do the right thing" (in the terms of (Russell and Wefald 1991)) in selecting nodes for sampling in MCTS.

Motivation

- Monte-Carlo tree search (MCTS) achieves major improvements in hard search problems.
- Many well-known success stories, such as Go, Poker, Canadian traveler problem, real-time strategy games.
- The most commonly used MCTS are UCT and its derivatives.
- In fact, UCT is based on a completely irrelevant "exploration-exploitation" **cumulative regret** criterion (as are almost all its variants).
- Using the more relavant **simple regret** criterion should (and does) improve the state of the art in MCTS.
- We show how to "do the right thing" (in the terms of (Russell and Wefald 1991)) in selecting nodes for sampling in MCTS.

Motivation

- Monte-Carlo tree search (MCTS) achieves major improvements in hard search problems.
- Many well-known success stories, such as Go, Poker, Canadian traveler problem, real-time strategy games.
- The most commonly used MCTS are UCT and its derivatives.
- In fact, UCT is based on a completely irrelevant "exploration-exploitation" **cumulative regret** criterion (as are almost all its variants).
- Using the more relavant **simple regret** criterion should (and does) improve the state of the art in MCTS.
- We show how to "do the right thing" (in the terms of (Russell and Wefald 1991)) in selecting nodes for sampling in MCTS.

Motivation

- Monte-Carlo tree search (MCTS) achieves major improvements in hard search problems.
- Many well-known success stories, such as Go, Poker, Canadian traveler problem, real-time strategy games.
- The most commonly used MCTS are UCT and its derivatives.
- In fact, UCT is based on a completely irrelevant "exploration-exploitation" **cumulative regret** criterion (as are almost all its variants).
- Using the more relavant **simple regret** criterion should (and does) improve the state of the art in MCTS.
- We show how to "do the right thing" (in the terms of (Russell and Wefald 1991)) in selecting nodes for sampling in MCTS.

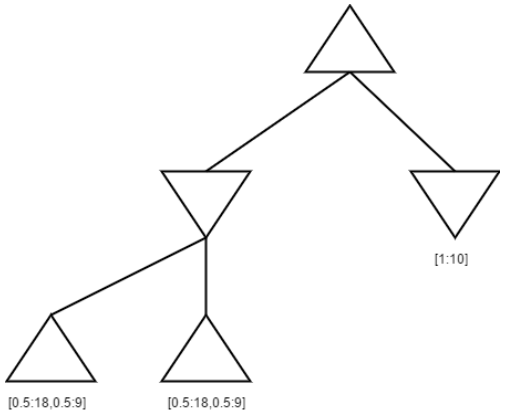
Motivation

- Monte-Carlo tree search (MCTS) achieves major improvements in hard search problems.
- Many well-known success stories, such as Go, Poker, Canadian traveler problem, real-time strategy games.
- The most commonly used MCTS are UCT and its derivatives.
- In fact, UCT is based on a completely irrelevant "exploration-exploitation" **cumulative regret** criterion (as are almost all its variants).
- Using the more relavant **simple regret** criterion should (and does) improve the state of the art in MCTS.
- We show how to "do the right thing" (in the terms of (Russell and Wefald 1991)) in selecting nodes for sampling in MCTS.

Outline

Monte-Carlo Tree Search (MCTS)

Usually used for searching huge game trees, where exhaustive search is impossible, and available heuristics are poor.

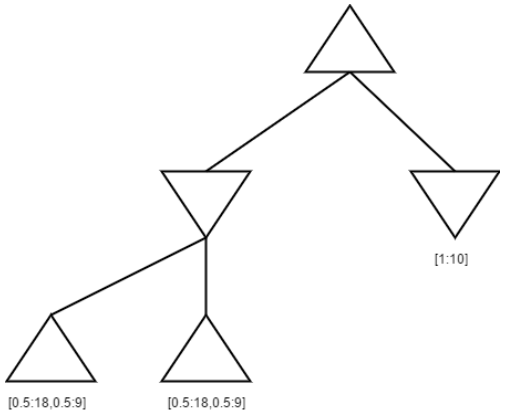


The types of search problems to which we apply MCTS are:

- MIN-MAX (deterministic zero-sum games, as in Go).
- EXPECTI-MAX (stochastic/incomplete information “games against nature”, as in the Canadian traveler problem).
- EXPECTI-MIN-MAX (stochastic/incomplete information adversarial games, as in Starcraft).

Monte-Carlo Tree Search (MCTS)

Usually used for searching huge game trees, where exhaustive search is impossible, and available heuristics are poor.

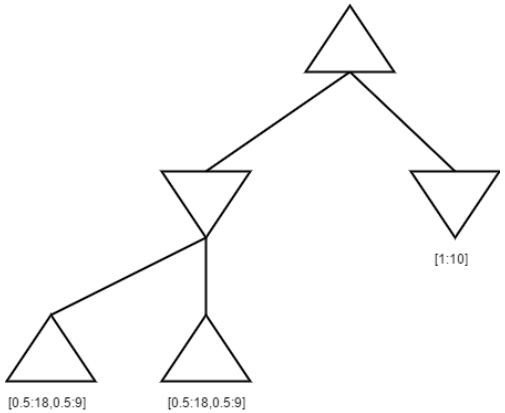


The types of search problems to which we apply MCTS are:

- MIN-MAX (deterministic zero-sum games, as in Go).
- EXPECTI-MAX (stochastic/incomplete information “games against nature”, as in the Canadian traveler problem).
- EXPECTI-MIN-MAX (stochastic/incomplete information adversarial games, as in Starcraft).

Monte-Carlo Tree Search (MCTS)

Usually used for searching huge game trees, where exhaustive search is impossible, and available heuristics are poor.

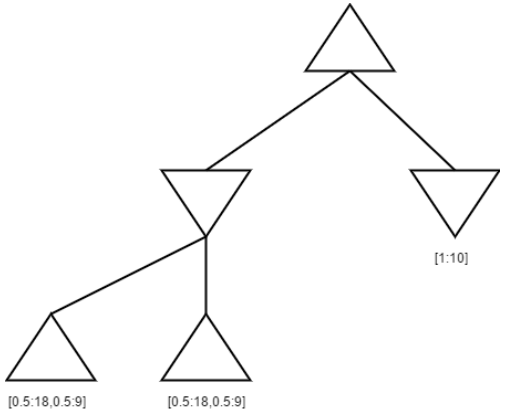


The types of search problems to which we apply MCTS are:

- MIN-MAX (deterministic zero-sum games, as in Go).
- EXPECTI-MAX (stochastic/incomplete information "games against nature", as in the Canadian traveler problem).
- EXPECTI-MIN-MAX (stochastic/incomplete information adversarial games, as in Starcraft).

Monte-Carlo Tree Search (MCTS)

Usually used for searching huge game trees, where exhaustive search is impossible, and available heuristics are poor.



The types of search problems to which we apply MCTS are:

- MIN-MAX (deterministic zero-sum games, as in Go).
- EXPECTI-MAX (stochastic/incomplete information “games against nature”, as in the Canadian traveler problem).
- EXPECTI-MIN-MAX (stochastic/incomplete information adversarial games, as in Starcraft).

In all of them computing and/or storing an optimal policy is infeasible, so search decides on a move to make at every turn.

Generic MCTS Algorithm

Function MCTS(root)
while *computation budget not exceeded* **do**
    $v \leftarrow$TreePolicy(root)
    $U \leftarrow$ DoRollout(v)
    Backup(v, U)
return action(BestChild(root))

TreePolicy consists of expanding nodes and selecting nodes from which to perform rollouts (also called “simulations” or “samples”).

In **this talk** we focus on the **node selection** for rollouts, in comparison to the almost ubiqutous **UCT** mechanism.

Multi-Armed Bandits, UCB and UCT

Consider facing a bank of "one armed bandit"s, i.e. gambling machines with unknown payoffs.

To optimize the cumulative payoff, need to trade off exploitation (pulling the current best arm) with exploration (trying unknown arms).

The upper confidence bounds (UCB) method asymptotically minimizes the cumulative regret (vs. always pulling the unknown optimal arm).

$$UCB1(j) = \bar{X}_j + \sqrt{\frac{2\ ln\ n}{n_j}}$$

The UCT algorithm simply applies the UCB1 formula at every level of the search tree, treating rollout outcomes as money (actual gain).

Why is UCT Wrong for MCTS

- Simply: a rollout (sample) does **not deliver** any value (other than its information value)!
- The value only accrues from the final choice of a move.
- UCT **appears** to work because it gives some weight to relatively unexplored nodes, which makes sense.
- But it samples the current best-appearing move, even if its value is certain, thus these samples are completely wasted.
- In fact, even if the algorithm re-uses samples in the next turn, the sampling should not depend on the value difference that way.

Why is UCT Wrong for MCTS

- Simply: a rollout (sample) does **not deliver** any value (other than its information value)!
- The value only accrues from the final choice of a move.
- UCT **appears** to work because it gives some weight to relatively unexplored nodes, which makes sense.
- But it samples the current best-appearing move, even if its value is certain, thus these samples are completely wasted.
- In fact, even if the algorithm re-uses samples in the next turn, the sampling should not depend on the value difference that way.

Why is UCT Wrong for MCTS

- Simply: a rollout (sample) does **not deliver** any value (other than its information value)!
- The value only accrues from the final choice of a move.
- UCT **appears** to work because it gives some weight to relatively unexplored nodes, which makes sense.
- But it samples the current best-appearing move, even if its value is certain, thus these samples are completely wasted.
- In fact, even if the algorithm re-uses samples in the next turn, the sampling should not depend on the value difference that way.

Why is UCT Wrong for MCTS

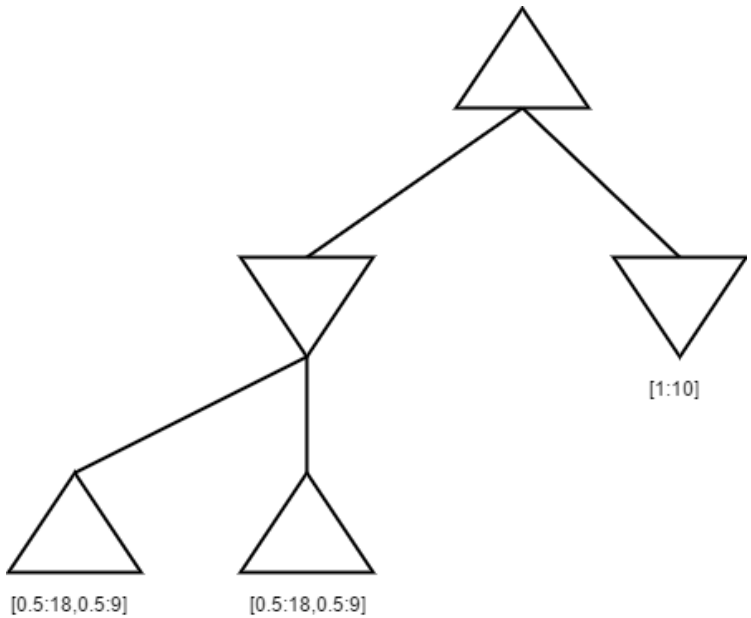
- Simply: a rollout (sample) does **not deliver** any value (other than its information value)!
- The value only accrues from the final choice of a move.
- UCT **appears** to work because it gives some weight to relatively unexplored nodes, which makes sense.
- But it samples the current best-appearing move, even if its value is certain, thus these samples are completely wasted.
- In fact, even if the algorithm re-uses samples in the next turn, the sampling should not depend on the value difference that way.

Why is UCT Wrong for MCTS

- Simply: a rollout (sample) does **not deliver** any value (other than its information value)!
- The value only accrues from the final choice of a move.
- UCT **appears** to work because it gives some weight to relatively unexplored nodes, which makes sense.
- But it samples the current best-appearing move, even if its value is certain, thus these samples are completely wasted.
- In fact, even if the algorithm re-uses samples in the next turn, the sampling should not depend on the value difference that way.

Rational Metareasoning in Search (Russel & Wefald 1991)

Given the current state of the search,
which search operator should we rationally perform (or none)?



Search meta-level problem is decision-making under uncertainty.

This problem is even more intractable than the original search problem!

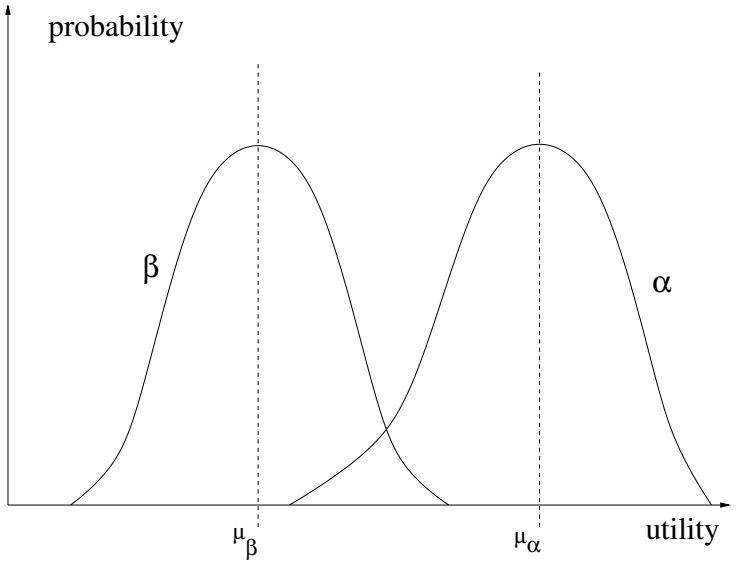
Rational Metareasoning: MGSS*

Define “value of computation” under simplifying assumptions:

1. Meta-greedy
2. Single step
3. Subtree independence

Compare expected utility of expanding a fringe node and then deciding on a move, vs. stopping the seach and deciding now.

Rational Metareasoning: MGSS*



$$VPI(S_\alpha) = \int_{-\infty}^{U(\beta_1)} p_{\alpha s_\alpha}(x)(U(\beta_1) - x)dx \quad (1)$$

$$VPI(S_{\beta_i}) = \int_{U(\alpha)}^{\infty} p_{\beta_i s_{\beta_i}}(x)(x - U(\alpha))dx \quad (2)$$

Simpler Setting: Selection Problem (Tolpin & Shimony 2012)

Given a set of items $\mathscr{I}$ you must pick $I_i \in \mathscr{I}$, gaining utility u_i.

But u_i is unknown (a distibution over the u_i is known).

Can pay C_i to measure u_i. Total measurement budget limit C.

What is the best way to measure and **then** pick an item?
(Minimize **simple** regret.)

Also called the “Bayesian selection and ranking” problem.

Multiple applications in experimental design, setup under uncertainty, and selection of physical items (apartments, employees, wine, potential oil fields), as well as in metareasoning.

Selection Problem: batch vs. sequential

Can consider **sequential** or **batch** settings.

- **batch** setting: can receive information about values only **once**.
- **sequential** setting: observations after each measurement, can be used to decide on additional measurements.

Obtaining optimal solution (even for batch setting) is NP-hard)!
Reduction from Knapsack problem (Reches, Gal, Kraus 2013).

Greedy algorithm: near-optimality for submodular VOI
(Krause & Guestrin 2011).

In general, neither the VOI nor the net VOI are submodular.

Outline

VOI in the Batch Selection Problem

Batch selection problem properties (Shperberg & Shimony 2017):

Theorem (Known α)

Independent utilities and u_α known $\Rightarrow$ submodular VPI.

Extends to imperfect information with single measurement per item.

Theorem (Selection is NP-Hard)

The batch selection problem with a budget is NP-hard, even under the conditions of Theorem 1.

Generalizing the Conditions

How far can we push generalization of the submodularity conditions?

Dependent item utility distributions: still submodular for 2 items (other than α), but not 3.

If an item may be picked only if its utility is known exactly: submodular even with dependencies, for any number of items (Azimi et. al 2016).

If u_α unknown, obviously neither submodular nor supermodular even for α and **one** additional item.

Theorem (Unknown α)

Theorem 3: if u_α is "sufficiently high", then VPI for sets of measurements constrained to measure α is submodular.

"Sufficiently high" means: $P(u_\alpha < \mu_{\beta_i}) = 0$ for all i.

Application: Compound Greedy Algorithm

Standard greedy algorithms: start from empty set, add "best" item, repeat as long as there is gain and budget not exhausted.

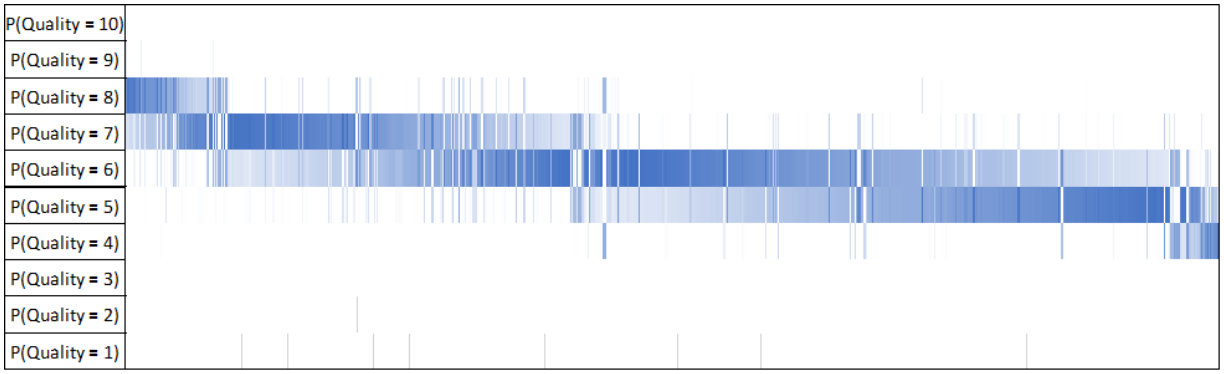
1. Additive greedy: use highest net VPI.
2. Rate greedy: use highest VPI divided by measurement cost.

Argument: submodular implies greedy is near-optimal.
Not including α causes the greatest deviation from submodularity.

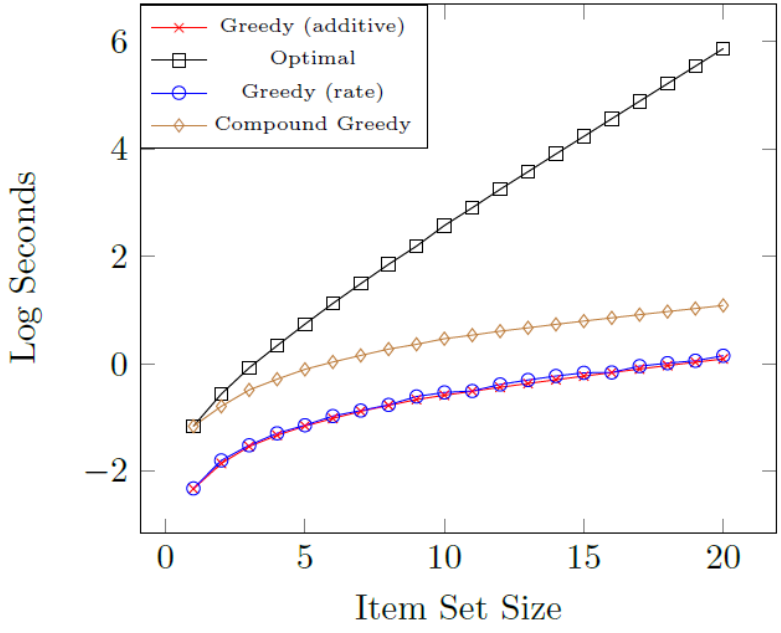
Compound greedy: run greedy starting with set consisting of α, as well as standard greedy. Return the better of both runs.

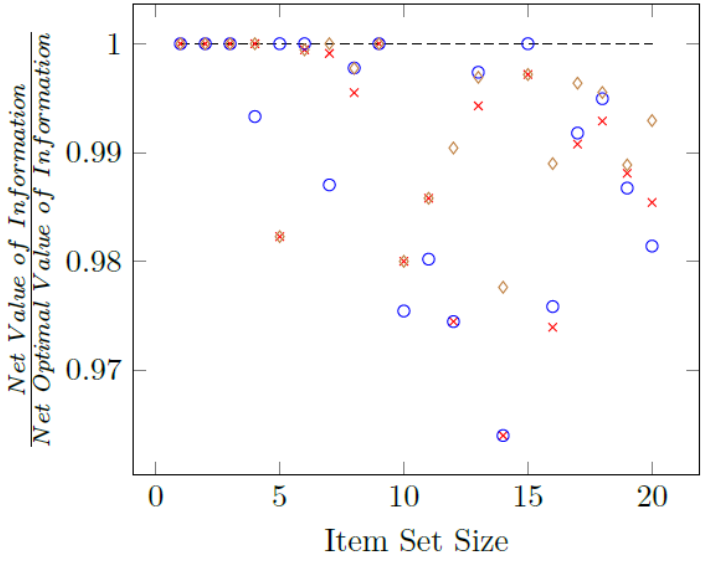
Note: in general (e.g. dependent case), computing VPI is NP-hard.

Results on Wine Selection Problem



Dataset: wine tasting data from the UCI repository.





Repeated Measurements: Blinkered Method

Repeated measurements: non-diminishing returns problem is severe.

(Tolpin & Shimony 2012) suggested a “blinkered” extension to greedy.

Considers VOI of batches of measurements of a single item.

Considerable improvement over pure greedy scheme, both theoretically and empirically.

Unknown: can “blinkered” be combined with compound greedy?

Outline

MCTS Based on Simple Regret (Tolpin & Shimony 2012)

UCT does the wrong thing for search: optimizes **cumulative** regret instead of **simple** regret.

UCB1 formula is very bad at optimizing simple regret, can be seen theoretically and empirically.

But using VPI as in MGSS* is problematic, since one sample does not provide even nearly perfect information on a node.

In addition, optimizing VOI during search is challenging: hard problem, needs to be solved in **negligible** runtime.

Blinkered MCTS (Hay, Russell, Shimony, Tolpin 2012)

Assuming bounded utilities and i.i.d. samples,
develop bounds on VOI for selection (at top tree level).

Theorem

Λ_i^b is bounded from above as

$$\Lambda_\alpha^b \leq \frac{N\overline{X}_\beta^{n_\beta}}{n_\alpha} \Pr(\overline{X}_\alpha^{n_\alpha+N} \leq \overline{X}_\beta^{n_\beta})$$

$$\Lambda_{i|i\neq\alpha}^b \leq \frac{N(1-\overline{X}_\alpha^{n_\alpha})}{n_i} \Pr(\overline{X}_i^{n_i+N} \geq \overline{X}_\alpha^{n_\alpha}) \quad (3)$$

Theorem

The probabilities in equations (3) are bounded from above as

$$\Pr(\overline{X}_{\alpha}^{n_\alpha+N} \leq \overline{X}_{\beta}^{n_\beta}) \leq 2\exp\left(-\varphi(\overline{X}_{\alpha}^{n_\alpha} - \overline{X}_{\beta}^{n_\beta})^2 n_\alpha\right)$$
$$\Pr(\overline{X}_{i|i\neq\alpha}^{n_\alpha+N} \geq \overline{X}_{\beta}^{n_\beta}) \leq 2\exp\left(-\varphi(\overline{X}_{\alpha}^{n_\alpha} - \overline{X}_{i}^{n_i})^2 n_i\right) \quad (4)$$

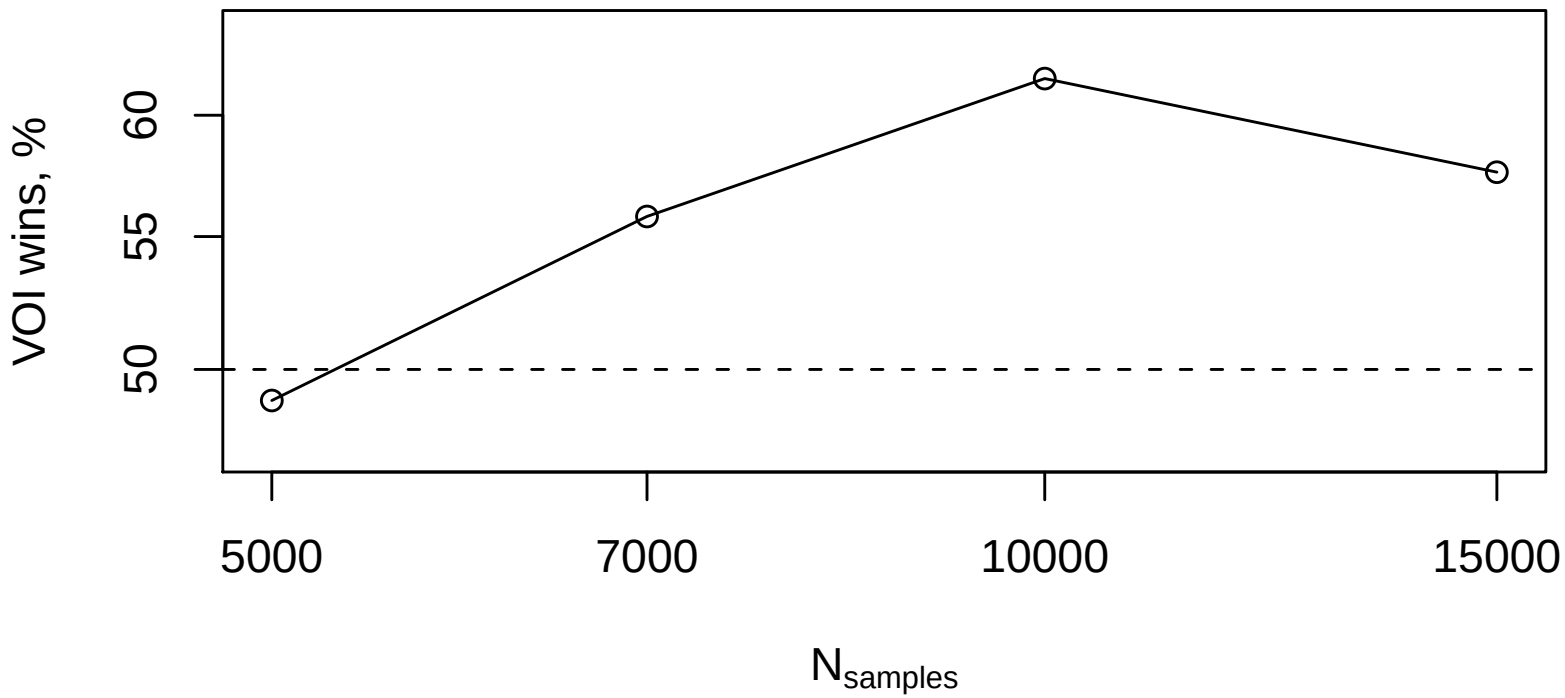
where $\varphi = \min\left(2(\frac{1+n/N}{1+\sqrt{n/N}})^2\right) = 8(\sqrt{2}-1)^2 > 1.37.$

Substitute (4) into (3) to bound the VOI estimate Λ_i^b:

$$\Lambda_{\alpha}^{b} \leq \hat{\Lambda}_{\alpha}^{b} = \frac{2N\overline{X}_{\beta}^{n_\beta}}{n_\alpha}\exp\left(-\varphi(\overline{X}_{\alpha}^{n_\alpha} - \overline{X}_{\beta}^{n_\beta})^2 n_\alpha\right)$$
$$\Lambda_{i|i\neq\alpha}^{b} \leq \hat{\Lambda}_{i}^{b} = \frac{2N(1-\overline{X}_{\alpha}^{n_\alpha})}{n_i}\exp\left(-\varphi(\overline{X}_{\alpha}^{n_\alpha} - \overline{X}_{i}^{n_i})^2 n_i\right) \quad (5)$$

Blinkered MCTS (Hay, Russell, Shimony, Tolpin 2012)

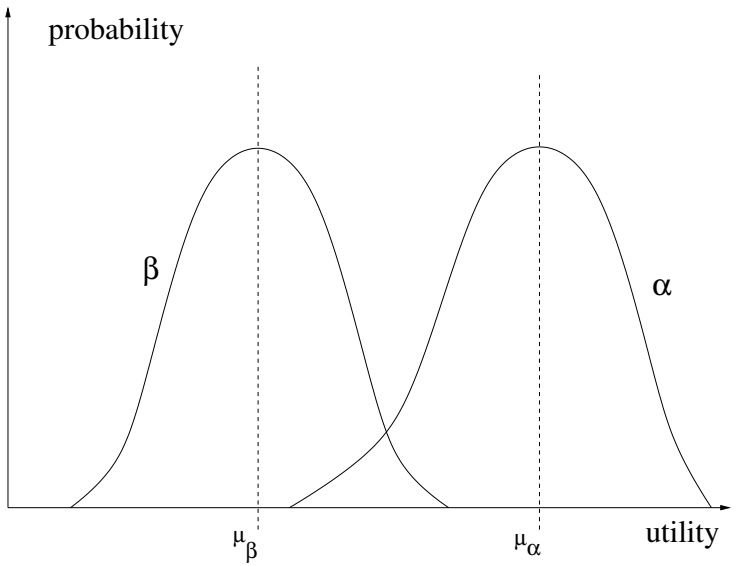
Using the VOI bounds instead of UCT on first level,
in the Pachi Go program, compared to vanilla Pachi:



Shortcomings of Blinkered MCTS

Use of UCT further down was because “we did not have anything better” at the time, **not** because it was correct.
Was not clear what **is** the correct thing to do at deeper levels.

Blinkered MCTS overcomes non-diminishing returns at one node, but not that due to non-submodularity at multiple nodes.



Latest idea is: go back to ideas from MGSS*, and extend them.

VI of a Batch Efficient Selection (Shperberg&Shimony 2017)

Extend the Russell and Wefald value of computation to:

$$BVPI(S) = \int_{y>x} p_{\alpha\beta S}(x,y)(y-x)dxdy \tag{6}$$

Where the distributions are given by:

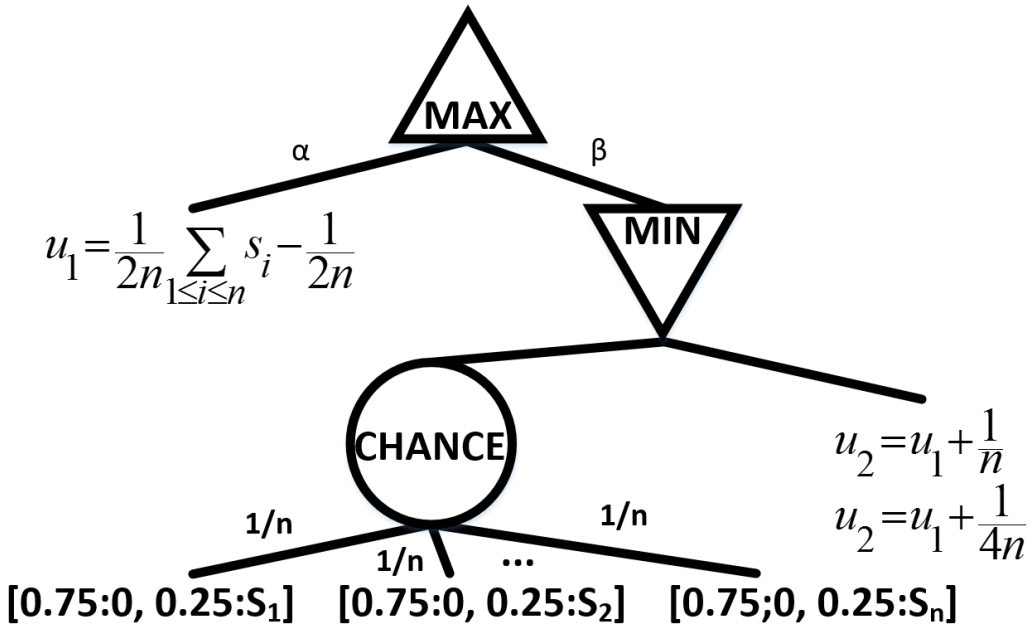
$$U_S(v) = \begin{cases} P_v & LEAF(v) \wedge v \in S \\ E_v[P_v] & LEAF(v) \wedge v \notin S \\ \max_{c \in ch(v)}\{U_S(c)\} & \text{MAXnode(v)} \\ \min_{c \in ch(v)}\{U_S(c)\} & \text{MINnode(v)} \\ \sum_{c \in ch(v)} p(c)U_S(c) & \text{CHANCEnode(v)} \end{cases} \tag{7}$$

Complexity of BVPI computation

Theorem (BVPI complexity)

Computing BVPI(S) for a given set of leaves S in expecti-mini-max trees is NP-hard.

Proof: by reduction from the partition problem.



Approximating the BVPI

Theorem (BVPI approximation)

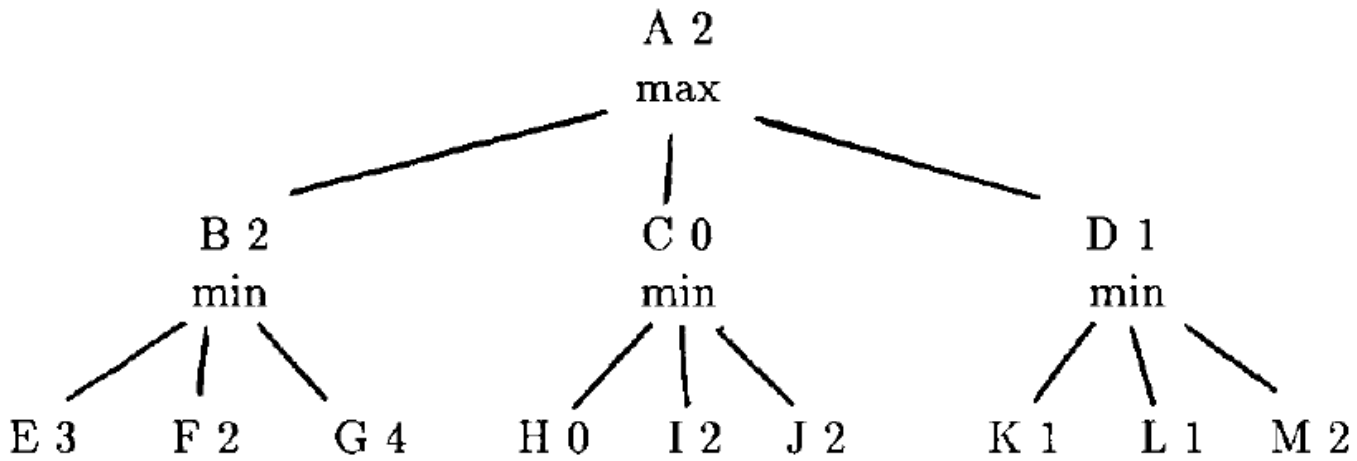
Given a game tree T with discrete distributions, the value $BVPI(S)$ where S is a subset of the leaves of T can be deterministically approximated within additive error ε in time polynomial in the (explicit) description size of T, $\frac{1}{\varepsilon}$, and utility bound $|U|$.

Method: bottom-up evaluation, trim distributions to $n = f(|T|, |U|, \frac{1}{\varepsilon})$ values (f is a polynomial function).
Causes "inverse Kolmokorov" distance below $\frac{2\,|U|}{n}$.

But still need to evaluate an exponential number of node-sets.

Conspiracy Numbers (McAllester 1988)

How many leaf nodes must change for the root value to become u?



As mentioned in (Russell&Wefald 1991),
$\#C > 1$ implies premature stopping of MGSS*
(and non-submodularity of the VPI).

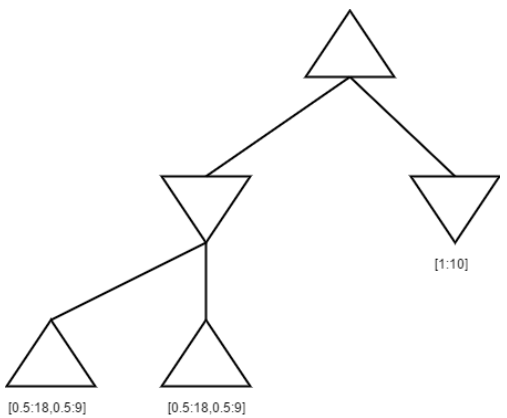
Adapting Conspiracy Numbers to BVPI Optimization

Similar to conspiracy numbers, define $\downarrow \phi(v, V)$ as the approximate **probability** of bringing the value of v down to at most V. Symmetrically define $\uparrow \phi(v, V)$.

Find a set of nodes which optimizes the expression:

$$\max_{V<V'}((V'-V)(\downarrow \phi(\alpha, V) \max_{c \in ch(r)-\{\alpha\}} \uparrow \phi(c, V')) \tag{8}$$

For efficient estimation, limit the domain of V, V' to a few discrete values (we used $\{0.95U(\alpha), U(\alpha), 1.05U(\alpha)\}$).



Application Domains

1. Canadian Traveler Problem (CTP): minimial weighted graph traversal with potentially blocked edges.
   This is a “deterministic POMDP”, shown to be PSPACE-hard.
   Good quality policies delivered by the UCTO algorithm (explores EXPECI-MAX trees).
2. StarCraft battle micro-management.
   A hard 2-player game.
   UCT-based algorithm works well (explores MIN-MAX trees).

Optimal vs. Greedy vs. Conspiracy on Trees

EXPECTI-MAX trees from the Canadian traveler problem

#	Tree Size	Tree Height	Max BF	Exhaustive		Conspiracy		Greedy	
				Net BVPI	T (**s**)	Net BVPI	T (**ms**)	Net BVPI	T (**ms**)
1	30	9	3	**359**	0.217	296	**0.28**	92	**0.28**
2	45	13	4	**453**	0.443	382	0.29	215	**0.28**
3	61	15	3	**126**	1.868	103	**0.29**	19	**0.29**
4	106	25	4	**527**	357	480	**0.34**	319	**0.34**
5	125	20	6	**812**	7130	681	0.36	681	**0.35**
6	153	42	5	**219**	20677	**219**	0.4	71	**0.38**
7	1018	33	6	T/O	T/O	**113**	0.78	0	**0.73**
8	3077	29	6	T/O	T/O	**65**	1.89	13	**1.88**
9	6820	37	5	T/O	T/O	**53**	4.2	0	**3.9**
10	15321	69	7	T/O	T/O	**277**	8.5	12	**8**

MIN-MAX trees from StarCraft

#	Tree Size	Tree Height	Max BF	Exhaustive		Conspiracy		Greedy	
				Net BVPI	T (**s**)	Net BVPI	T (**ms**)	Net BVPI	T (**ms**)
1	30	4	20	**283**	0.198	234	0.25	205	**0.24**
2	45	4	20	**356**	0.413	**356**	**0.25**	**356**	**0.25**
3	60	6	20	**326**	1.719	312	**0.28**	283	**0.28**
4	100	11	20	**147**	300	**147**	0.33	105	**0.31**
5	125	11	20	**63**	7189	**63**	**0.36**	**63**	**0.36**
6	150	12	20	**96**	19939	71	0.39	0	**0.38**
7	1000	23	20	T/O	T/O	**33**	0.72	**33**	**0.70**
8	3000	28	20	T/O	T/O	**116**	1.85	42	**1.82**
9	7000	42	20	T/O	T/O	**25**	4.33	7	**4.01**
10	15000	79	20	T/O	T/O	**47**	8.41	0	**7.96**

Moving from BVPI Optimization to Choosing Rollouts

Recall that deciding on rollouts must take *negligible* time in order to actually improve search performance.

We amortized the overhead over N rollouts (typically 3 to 5).
And for algorithms where we decide on a **set** of nodes also over K nodes (typically 5).
New algorithms we developed:

- FT-VIBES: finds node-set with (near) optimal BVPI.
- FL-VIBES: near-optimal choice at first level, then UCTO.
- RFL-VIBES: near-optimal choice at each level, recursive.
- C-VIBES: uses conspiracy scheme to find node-set.
- Greedy: using MGSS* formula.

Empirical past rollout value distributions assumed “correct”.

Results on CTP Instances (3 seconds per move)

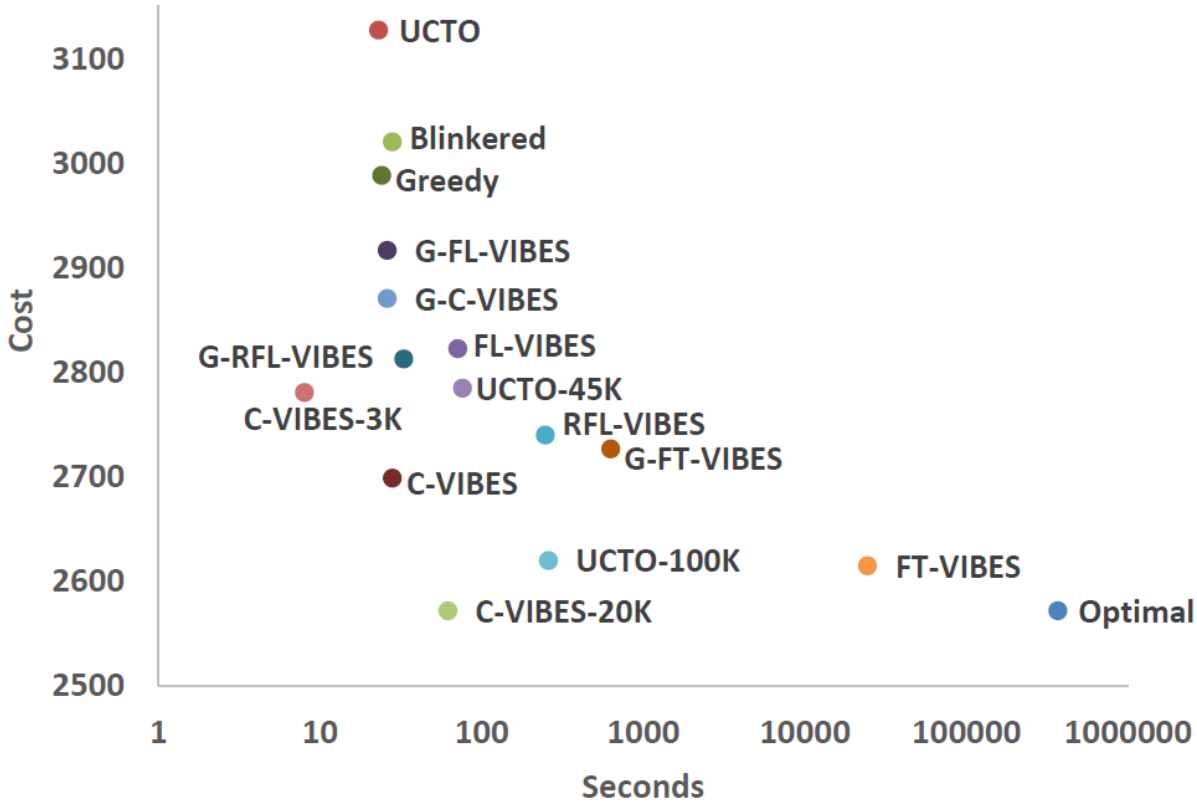
Settings	UCTO	Blink'd	FL-V	RFL-V	FT-V	B-V	C-V	Greedy	G-FL-V	G-RFL-V	G-FT-V	G-C-V
n: 30, p = 0.2	3,006	2,911	2,989	2,830	3,625	2,785	**2,598**	2,902	2,709	2,668	2,973	2,747
n: 30, p = 0.4	3,600	3,490	3,564	3,397	4,356	3,336	**3,130**	3,473	3,253	3,224	3,563	3,294
n: 30, p = 0.6	4,198	4,070	4,172	3,959	5,095	3,901	**3,630**	4,098	3,804	3,758	4,149	3,829
n: 30, p = 0.8	4,801	4,646	4,768	4,510	5,829	4,469	**4,165**	4,599	4,344	4,298	4,742	4,388
n: 30, p = 1	5,401	5,241	5,378	5,099	6,521	5,028	**4,675**	5,226	4,893	4,838	5,346	4,933
Avg. of n=30	4,201	4,072	4,174	3,959	5,085	3,904	**3,640**	4,060	3,801	3,757	4,155	3,838
n: 60, p = 0.2	5,049	4,836	4,870	4,804	6,157	4,753	**4,340**	4,842	4,590	4,480	5,091	4,495
n: 60, p = 0.4	7,074	6,775	6,805	6,735	8,611	6,665	**6,095**	6,762	6,412	6,252	7,140	6,309
n: 60, p = 0.6	8,572	8,237	8,280	8,179	10,465	8,084	**7,396**	8,268	7,745	7,568	8,630	7,651
n: 60, p = 0.8	10,082	9,704	9,698	9,603	12,335	9,501	**8,696**	9,667	9,175	8,915	10,224	9,030
n: 60, p = 1	11,572	11,091	11,213	11,049	14,179	10,937	**9,968**	11,102	10,534	10,266	11,708	10,366
Avg. of n=60	8,470	8,129	8,173	8,074	10,349	7,988	**7,299**	8,128	7,691	7,496	8,559	7,570
n: 100, p = 0.2	7,506	7,277	7,392	7,210	8,929	6,965	**6,240**	7,299	6,961	6,912	7,273	6,668
n: 100, p = 0.4	10,966	10,614	10,763	10,546	13,005	10,160	**9,109**	10,581	10,151	10,139	10,488	9,753
n: 100, p = 0.6	13,459	13,032	13,232	12,932	15,961	12,450	**11,162**	13,096	12,424	12,422	12,810	11,957
n: 100, p = 0.8	15,916	15,469	15,634	15,264	18,923	14,740	**13,226**	15,490	14,683	14,634	15,157	14,089
n: 100, p = 1	18,346	17,886	18,076	17,702	21,851	17,066	**15,273**	17,787	17,005	17,014	17,461	16,284
Avg. of n=100	13,239	12,855	13,019	12,731	15,734	12,276	**11,002**	12,851	12,245	12,224	12,638	11,750

Results on a typical CTP instance (3 seconds per move)

Selection alg.	Cost	Subopt.	#RollOuts/Move
Optimal (VI)	2,572	0%	N/A
UCTO	3,128	21.6%	**12,371**
Blinkered	3,021	17.5%	11,016
FL-VIBES	3,096	20.4%	4,284
RFL-VIBES	2,931	14.0%	1,576
FT-VIBES	3,778	46.9%	43
B-VIBES	2,890	12.4%	745
C-VIBES	**2,699**	**4.5%**	9,989
Greedy	2,989	16.2%	11,904
G-FL-VIBES	2,823	9.8%	10,978
G-RFL-VIBES	2,784	8.2%	8,732
G-FT-VIBES	3,084	19.9%	611
G-C-VIBES	2,847	10.7%	10,633

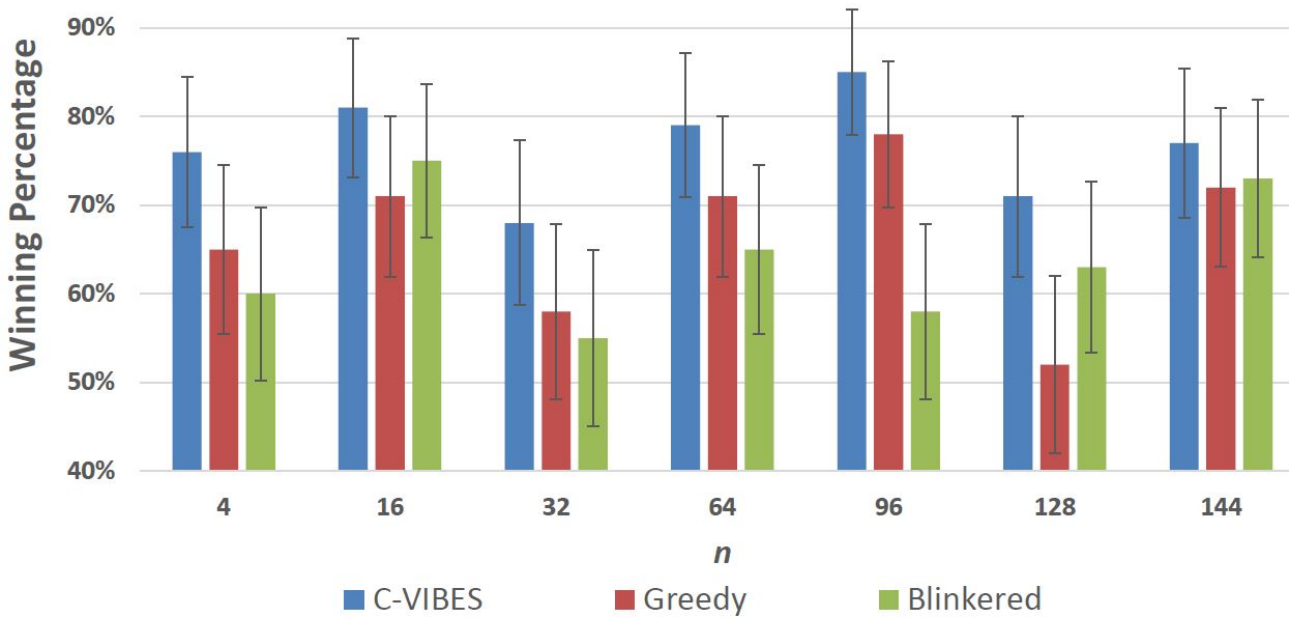
Results on Same CTP Instance (limited rollouts per move)

Unless specified otherwise, limit was 10,000 rollouts per move



Results in StarCraft Domain

Win rates vs. UCT (100 games).
Error bars are 95% confidence intervals.
n is the army size in each battle.



Outline

Summary

- Submodularity is an important property for approximately optimizing VOI, but unfortunately does not always hold.
- We examined the boundries of submodularity for selection.
- Severe points of deviations from diminishing returns (including submodularity) require careful extensions to the greedy algorithms.
- In some cases these extensions can be efficiently approximated:
  1. Compound-greedy for pure selection.
  2. Conspiracy-based VPI optimization in game trees.
- Applying VOI optimization to MCTS selection reaps large rewards, **despite** overhead due to metareasoning.

Future Work

- Find other cases of deviation from submodularity that can be easily finessed.
  E.g. “multi-item selection” (with Yuri Shafet)
- Applying the information-gathering optimizations to other types of game-tree computations, such as node expansions.
- We computed VOI under the (incorrect) assumption that the empirical distribution from the rollouts is the fringe node value distribution.
  1. Extending “blinkered-like” bounds to multiple nodes?
  2. Assume Bayesian updating of fringe-node distributions.
- Experiment in additional domains, we still want to try alpha-GO...

Thank You