To UCT, or not to UCT? (Position Paper)

Carmel Domshlak and **Zohar Feldman**
Faculty of Industrial Engineering and Management
Technion, Israel

Abstract

Monte-Carlo search is successfully used in simulation-based planning for various large-scale sequential decision problems, and the UCT algorithm (Kocsis and Szepesvári 2006) seems to be the choice in most (if not all) such recent success stories. Based on some recent discoveries in theory and empirical analysis of Monte-Carlo search, here we argue that, if online sequential decision making is your problem, and Monte-Carlo tree search is your way to go, then UCT is unlikely to be the best fit for your needs.

Introduction

In online sequential decision making, the agent focuses on its current state s_0, deliberates about the set of possible courses of action from s_0 onwards, and, when interrupted, uses the outcome of that exploratory deliberation to choose what action to perform at s_0. Sampling-based, or Monte-Carlo (MC), approximation methods, became a prominent tool in online planning for different types of large-scale sequential decision problems. Using MC methods, the agent deliberates about a model of the problem by simulating experiences induced by sampled sequences of actions starting from s_0. The respective MC methods are usually referred to as MC tree search (MCTS) algorithms (Browne et al. 2012).

The success stories of MCTS are numerous, from Markov decision processes (MDP) (Balla and Fern 2009; Keller and Eyerich 2012) to partially observable Markov decision processes (POMDP) (Bjarnason, Fern, and Tadepalli 2009; Eyerich, Keller, and Helmert 2010) to sequential multi-person games (Gelly and Silver 2011; Sturtevant 2008; Lorentz 2008; Winands and Björnsson 2009; Finnsson and Björnsson 2010). Remarkably, most (if not all) such recent success stories report on relying on a very specific MCTS algorithm called UCT (Kocsis and Szepesvári 2006). Introduced in the context of online planning for MDPs, UCT extends the seminal MC scheme UCB1 (Auer, Cesa-Bianchi, and Fischer 2002) from stochastic multi-armed bandit (MAB) problems to general MDPs.

This unanimous adoption of UCT suggests its unique attractiveness among various possible MCTS schemes. This perception is probably also stratified by the fact that UCB1, the beating heart of UCT, is known to be optimal in a certain context of reinforcement learning in MABs (Auer, Cesa-Bianchi, and Fischer 2002). Somewhat surprisingly, however, when we started our journey in MCTS, we found in literature neither convincing theoretical nor convincing empirical evidence for this or another superiority of UCT. In fact, as we discuss later on, a closer inspection of various successful problem solvers using UCT suggests that the marginal contribution of UCT specifics to their success was rather secondary. In particular, that even includes the state of the art Go playing agents, binding of which to UCT has already become a lore.

In what follows, we examine the chronology of the remarkable UCT adoption, discuss the role of UCT in various success stories around it, and take a stand on what made UCT such a popular choice in online planning. Then, based on some recent formal and empirical results on online best action identification (and, the intimately related, "simple regret" minimization) in sequential decision problems, we communicate our position: If online sequential decision making is your problem, and Monte-Carlo tree search is your way to go, then UCT is highly unlikely to be the best fit for your needs. We backup our position with discussing some recent MCTS algorithms that outperform UCT both formally and (on the benchmarks that have been examined so far) also empirically. We believe that these findings should in particular encourage the community to further expand research on MCTS techniques for sequential decision problems.

Online Planning and UCT

While MCTS algorithms are used these days in different types of sequential decision problems, most of them (including UCT) were first introduced in the context of planning and reinforcement learning in MDPs. Adaptations to more general problem classes is usually rather direct, and thus here we also describe things mostly in terms of (finite horizon) MDPs. An MDP $\langle S, A, Tr, R\rangle$ is defined over states S, actions A, a stochastic transition function $Tr : S \times A \times S \to [0, 1]$, and a reward function $R : S \times A \times S \to \mathbb{R}$. In the finite horizon setting, the reward is accumulated over some predefined number of steps H. Henceforth, $A(s) \subseteq A$ denotes the actions applicable in state s, the operation of drawing a sample from a distribution $\mathcal{D}$ over set $\aleph$ is denoted by $\sim \mathcal{D}[\aleph]$,

$\mathcal{U}$ denotes uniform distribution, and $[\![n]\!]$ for $n \in \mathbb{N}$ denotes the set $\{1, \ldots, n\}$.

Setup/Algorithm Taxonomy

Our focus here is on online MCTS planning. The notion of "online planning", however, is somewhat ill-defined, and thus here we explicitly specify and justify the axes along which we distinguish between various MCTS algorithms, and, in these terms, what type(s) of algorithms are of our interest.

Prior knowledge. In many works, the very notion of onlineness corresponds to "no prior (explicit or learned) knowledge of state/action value is required". That is, while online algorithms can possibly exploit such a prior knowledge, they remain well-defined even if no such prior knowledge is available, and their key formal properties (whatever they are) stand steel.

Objective. When it comes to action recommendation, different algorithms may focus on different notions of action quality. Most often, the quality of the action a, chosen for state s with H steps-to-go, is assessed in terms of the probability that a is sub-optimal, or in terms of the (closely related) measure of simple regret. The latter captures the performance loss that results from taking a and then following an optimal policy π^* for the remaining $H-1$ steps, instead of following π^* from the beginning (Bubeck and Munos 2010). Here we focus on algorithms that aim at optimizing simple regret, but note that other (e.g., risk related) planning objectives have also been motivated and studied in control theory (Sobel 1982; Mannor and Tsitsiklis 2011).

Soundness of convergence. If the action-choice error probability goes to zero as a function of the computational effort, then we say that the algorithm is *sound*.[1] For instance, offline, feature-based learning of action values (such as, e.g., variants of temporal difference learning (Sutton 1988)), are very popular in the context of sequential decision processes (Buro 1999; Schaeffer, Hlynka, and Jussila 2001; Gelly and Silver 2007). However, such offline learning algorithms can be sound only subject to representability of the value function in the feature space in use, which in general cannot be efficiently verified, and most typically does not hold. That, in particular, gave rise to considering online action value assessment.

Smoothness of convergence. In terms of the planning setup, we should distinguish between *guarantee-contract*, *resource-contract*, and *interruptible* algorithms for MDP planning (Zilbershtein 1993). A guarantee-contract algorithm is posed a guarantee requirement on the quality of the recommended action, and it should aim at achieving the required guarantee as fast as possible. For instance, the well-studied PAC bounds (ϵ, δ) correspond to guaranteeing a simple regret of at most ϵ with probability of at least δ (Kearns, Mansour, and Ng 1999; Even-Dar, Mannor, and Mansour 2002). In resource-contract setting, the setup is reversed: the algorithm is given a budget of computation units (e.g., time) and it should aim at achieving within that budget the best action recommendation possible. In planning for MDPs, there is a very close connection between these two contract setups (Even-Dar, Mannor, and Mansour 2002).

In contrast, in the interruptible setting, the planning time is not known in advance, and when interrupted, the algorithm is required to provide a recommendation in time $O(1)$. In principle, any resource-contract algorithm can be compiled into an interruptible algorithm with only a constant factor loss in recommendation quality for any interruption point (Theorem 4.1 of Zilbershtein (1993)), the resulting interruptible algorithm will improve the quality of its recommendation only in time steps that are typically unacceptably large.[2] Hence, interruptibility in practice is a *de facto* synonym to *continuous, smooth* improvement of the recommendation quality over time.

In terms of this algorithm taxonomy, by ***online planning algorithms*** here we refer to algorithms that do not rely on prior domain knowledge, aim at simple regret minimization, and converge in a sound and smooth way. This type of MCTS algorithms seems to dominate applications of MCTS in sequential decision processes, with UCT (as we discuss below) being the most popular such algorithm these days. Finally, we note that MCTS is clearly not the only way to go when it comes to sequential decision processes (Powell 2011; Mausam and Kolobov 2012; Bonet and Geffner 2012; Kolobov, Mausam, and Weld 2012; Busoniu and Munos 2012). However, non-MCTS methods are not in the scope of our discussion here.

The UCT Algorithm

MCTS, a canonical scheme underlying various MCTS algorithms for online MDP planning, is depicted in Figure 1a. Starting with the current state s_0, MCTS performs an iterative construction of a tree[3] $\mathcal{T}$ rooted at s_0. At each iteration, MCTS rollouts a state-space sample ρ from s_0, which is then used to update $\mathcal{T}$. First, each state/action pair (s, a) is associated with a counter $n(s, a)$ and a value accumulator $\widehat{Q}(s, a)$, both initialized to 0. When a sample ρ is rolled out, for all states $s_i \in \rho \cap \mathcal{T}$, $n(s_i, a_{i+1})$ and $\widehat{Q}(s_i, a_{i+1})$ are updated on the basis of ρ by the UPDATE-NODE procedure. Second, $\mathcal{T}$ can also be expanded with any part of ρ; The standard choice is to expand $\mathcal{T}$ with only the first state along ρ that is new to $\mathcal{T}$ (Coulom 2006; Kocsis and Szepesvári 2006). In any case, once the sampling is interrupted, MCTS uses the information stored at

[1] While "converging" may seem a more natural name for this property, it may also be somewhat misleading as some MCTS algorithm converge, but not necessarily to the right thing.

[2] For MDPs, these steps do not have to grow exponentially with time as in the general compilation scheme of Zilbershtein (1993). However, the size of the step will still be of the order of the size of the state space reachable from s_0 in H steps. For all but very simple MDPs, such steps are clearly of theoretical interest only.

[3] In MDPs, there is no reason to distinguish between nodes associated with the same state at the same depth. Hence, the graph $\mathcal{T}$ constructed by MCTS instances typically forms a DAG. Nevertheless, for consistency with prior literature, we stay with the term "tree".

MCTS: [input: $\langle S, A, Tr, R\rangle$; $s_0 \in S$]
search tree $\mathcal{T} \leftarrow$ root node s_0
for $n \leftarrow 1 \ldots$ time permits **do**
PROBE$(s_0, 0)$
return $\arg\max_a \widehat{Q}(s_0, a)$

PROBE (s : state, d : depth)
if END-OF-PROBE(s, d) **then return** EVALUATE(s, d)
$a \leftarrow$ ROLLOUT-POLICY(s)
$s' \sim P(S \mid s, a)$
$r \leftarrow R(s, a, s') +$ PROBE$(s', d+1)$
UPDATE-NODE(s, a, r)
return r

(a)

END-OF-PROBE (s : state, d : depth)
if $s \notin \mathcal{T}$ **then**
add s to $\mathcal{T}$ and **return** true
else if $d = H$ **then return** true **else return** false

UPDATE-NODE (s : state, a: action, r : reward)
$n(s, a) \leftarrow n(s, a) + 1$
$\widehat{Q}(s, a) \leftarrow \widehat{Q}(s, a) + \frac{r - \widehat{Q}(s,a)}{n(s,a)}$

ROLLOUT-POLICY (s : state)
if $n(s, a) = 0$ for some $a \in A(s)$ **then**
return $a \sim \mathcal{U}[\{a \in A(s) \mid n(s, a) = 0\}]$
else
$n(s) \leftarrow \sum_{a \in A(s)} n(s, a)$
return $\text{argmax}_a \left[\widehat{Q}(s, a) + c\sqrt{\frac{\log n(s)}{n(s,a)}}\right]$

EVALUATE (s : state, d : depth)
for $t \leftarrow d \ldots H$ **do**
$a \sim \mathcal{U}[A(s)]$
$s' \sim P(S \mid s, a)$
$r \leftarrow r + R(s, a, s')$
$s \leftarrow s'$
return r

(b)

Figure 1: (a) Monte-Carlo tree search template, and (b) the UCT specifics.

the tree's root to recommend an action to perform in s_0.

Concrete instances of MCTS vary mostly[4] along the implementation of the ROLLOUT-POLICY sub-routine, that is, in their policies for directing the rollout within $\mathcal{T}$. Numerous concrete instances of MCTS have been proposed, with UCT (Kocsis and Szepesvári 2006) apparently being the most popular such instance these days. The specific ROLLOUT-POLICY of UCT is shown in Figure 1b. This policy is based on the deterministic decision rule UCB1 of Auer, Cesa-Bianchi, and Fischer (2002) for optimal balance between exploration and exploitation for cumulative regret minimization in simultaneous acting and learning in stochastic multi-armed bandit (MAB) environments (Robbins 1952). Adaptation of UCT to multi-person games is straightforward (Sturtevant 2008); for instance, in two-person zero-sum games, ROLLOUT-POLICY as in Figure 1 should simply be adapted to apply argmax and argmin at the player's and opponent's turns, respectively. This is also true for many other rollout schemes such ε-greedy, Boltzmann exploration, etc.

[4] MCTS instances may also vary along the implementation of EVALUATE. However, it is hard to think of an implementation other than that in Figure 1b, unless the algorithm is provided with a problem-specific prior and/or heuristic.

Adoption of UCT: A Closer Look

The initial adoption of UCT had been driven by advantages it offered comparatively to some other algorithms that were used for sequential decision problems, such as offline feature-based learning, sparse sampling, non-sound interruptible MC algorithms such as flat MC, and primarily, alpha-beta minimax search (Browne et al. 2012). In turn, these successes created an information cascade, and many recent works report on adopting UCT because it was widely adopted already. But what are the specific properties of UCT that have attracted so many practitioners across various types of sequential decision problems, and was this attraction firmly justified? Considering a wide sample of works on online MCTS planning, the arguments for adopting UCT appear to be as follows.

Onlineness. Unlike the algorithms that UCT came to replace, UCT is a truly online planning algorithm in the sense we defined above: it requires no prior domain knowledge, the estimated values converge to the true (regular or minimax, depending on the setting) Q-values of the actions, and the algorithm smoothly improves its estimates over time. Together and separately, all these aspects of UCT's onlineness are discussed in favor of UCT across the literature.

In the MAB setting, the UCB1 action-selection strategy employed by UCT guarantees that the proportion of samples devoted to the best action goes to 1 as the overall number of samples goes to infinity. The claim of the UCT convergence to the true action values relies precisely on (and only on) that guarantee to carry on to MDPs. Yet, UCB1 is not unique in that respect, as other strategies, such as ε-greedy action selection with time-adaptive value of ε (Auer, Cesa-Bianchi, and Fischer 2002), also provide such a guarantee. Thus, instances of MCTS that incorporate these action-selection strategies with appropriate adjustments may very well be proven sound as well.

What does single out UCB1 is that its soundness does not depend on the choice of this or another set of control parameters that cannot be determined a priori, a quality that other sound strategies lack. In other words, UCT offers not only sound and smooth, but also *parameter-free* convergence of action choice in MDPs/games. Somewhat surprisingly, however, we found no mention/discussion of this issue in the literature, and this despite the clear *practical* value of that uniqueness of UCT. Hence, while that property of UCT could have explained the rapid adoption of this algorithm, there must be something else responsible for the matter.

Favors more promising parts of the search tree. Following the UCB1 action selection policy at its rollouts, UCT biases sampling and tree construction towards the actions

that are estimated at the moment to be of higher value. Together with the fact that UCB1 ensures that no action is ignored for good, this property of UCT is almost unanimously accounted in favor of UCT (Gelly et al. 2006; Gelly and Silver 2007; Lorentz 2008; Winands and Björnsson 2009; Balla and Fern 2009). While at first view this asymmetry of rollout dynamics seems to be appealing, this appeal was never actually justified on a firm basis. Furthermore, certain evidence suggested that the issue is not as straightforward as it may look like.

On the side of empirical experience, numerous works revealed high sensitivity of the UCT performance with respect to the UCB1's balancing parameter c, that is, sensitivity to precisely that favoring of the search space regions that appear to be more attractive at the moment (Lorentz 2008; Balla and Fern 2009; Gelly et al. 2006). In fact, in some domains, extensive tuning revealed no single value for c that works reasonably well across the state space (Balla and Fern 2009). Furthermore, right at the first days of applying UCT in Go, Gelly et al. (2006) observed that "it is obvious that the random variables involved in UCT are not identically distributed nor independent. [...] the bias is amplified when passing from deep level to the root, which *prevents* the algorithm from finding quickly the optimal arm at the root node" [emphasis added]. Still, our formal understanding of UCT's convergence dynamics back then was rather preliminary, and thus there was no clear argument against the seemingly appealing sampling asymmetry of the UCT's rollout policy.

Effective in theory. It took few more years until some first formal properties of UCB1 in the context of simple regret minimization have been revealed. First, Coquelin and Munos (2007) showed that the number of samples after which the bounds of UCT on simple regret become meaningful might be as high as hyper-exponential in the horizon H. While these "bad news" are mostly of theoretical interest (since being online, UCT continuously improves its estimates right from the first rollouts), having even a theoretical cold-start period of such a magnitude rings a bell. Later on, Bubeck, Munos, and Stoltz (2011) showed in the context of stochastic multi-armed bandits that sampling with bias towards seemingly better actions may considerably slow down the reduction of simple regret over time. In particular, Bubeck, Munos, and Stoltz (2011) showed that UCB1 achieves only *polynomial-rate* reduction of simple regret over time, and this result immediately implies that the simple regret reduction rate of UCT is at most polynomial over time. Of course, that in itself does not mean that a *much* better convergence rate can be achieved with any other online MCTS algorithm. In fact, none of the MCTS instances suggested until very recently broke the barrier of the worst-case polynomial-rate reduction of simple regret over time. On the other hand, some of these alternative MCTS instances, such as ε-greedy+UCT algorithm of Tolpin and Shimony (2012) or a MAB-based enhancement of UCT proposed by Coquelin and Munos (2007), do provide somewhat better convergence rates for simple regret. At least, that suggests that the speed of convergence does not constitute a unique attractiveness of UCT. For now, however, we postpone the issue of convergence rate to the next section, and proceed with examining the empirical attractiveness of UCT.

Effective in practice. At the end of the day, what probably matters the most is the empirical effectiveness of planning. In that respect, it is most valuable to examine the reported experiences with UCT in the earlier works that adopted this algorithm—as we already mentioned, more recent work seem to follow UCT adoption simply because "it is known to work well". Interestingly, a closer look reveals that UCT-based systems that were truly successful in their domains were all using UCT only in conjunction with some other, either ad hoc or problem specific, tools for heuristic action value estimation such as rapid action value estimation (RAVE) (Gelly and Silver 2007), first-play urgency (FPU) (Gelly et al. 2006), progressive bias (PB) (Winands and Björnsson 2009), move-average sampling (MAST) (Finnsson and Björnsson 2008), predicate-average sampling (PAST) (Finnsson and Björnsson 2010), features-to-action sampling (Finnsson and Björnsson 2010), heuristic Q-value initialization (Keller and Eyerich 2012), optimistic Q-value initialization (Eyerich, Keller, and Helmert 2010), to name just a few[5].

More importantly, it appears that the value of using UCT (and not some other MCTS scheme) in these systems is rather marginal, and the key effectiveness actually comes from the aforementioned auxiliary tools. For instance, in their analysis of combining UCT with rapid action value estimation (RAVE) in *MoGo*, one of the strongest Go programs these days, Gelly and Silver (2007) write that "... rapid action value estimate is worth about the same as several thousand episodes of UCT simulation." In fact, later Silver (2009) writes that setting the bias parameter of UCB1 to zero (that is, reducing it to greedy MCTS) is the best configuration of UCT in pair with RAVE in *MoGo*. Similar in spirit observations on the crucial importance of heuristically guiding UCT have also been made in the context of award-winning solvers for general game playing (Björnsson and Finnsson 2009) and MDP planning (Keller and Eyerich 2012).

Online MCTS Better Than UCT?

In line with our findings above, Browne et al. (2012) summarize their survey on MCTS techniques by writing that, "although basic implementations of MCTS provide effective play for some domains, results can be weak if the basic algorithm is not enhanced." Still, the *relative effectiveness of the basic MCTS algorithms* does not seem to be systematically studied in the prior work. Thus, the way we see it, two basic and important questions in the context of "to UCT or not to UCT" remained open, notably

- Can the barrier of only polynomial-rate reduction of simple regret over time be removed in the context of online MCTS algorithms?, and

[5]For a comprehensive survey of such tools up to 2011, we refer the reader to (Browne et al. 2012).

- Should UCT a priori be expected to be the most effective basic online MCTS algorithm around?

In what follows, we discuss some recent findings in theory and empirical analysis of MCTS that provide a crisp answer to the first of these questions, and a solid answer to the second one. In contrast to what seems to be a common belief these days, these findings suggest that, if online sequential decision making is your problem, and MCTS is your way to go, then UCT is unlikely to be the best fit for your needs.

Exponential-rate convergence?

Bubeck, Munos, and Stoltz (2011) showed that an exponential-rate reduction of simple regret over time is achievable in stochastic multi-armed bandit (MAB) problems. The respective result is distribution-dependent, with the dependency of the bounds on the arm reward distributions being reflected through the difference Δ between the expected rewards of the best and second-best arms. Since then, the question of whether a similar rate of convergence can be achieved for online MDP planning became even more relevant than before, and recently we have provided an affirmative answer to this question (Feldman and Domshlak 2012).

The respective MCTS algorithm, BRUE, is an instance of a non-standard MCTS scheme MCTS2e, a refinement of MCTS that implements a principle of "separation of concerns". According to that principle, different parts of each sample are devoted exclusively either to search space exploration or to action value estimation. In MCTS2e (Figure 2a), rollouts are generated by a two-phase process in which the actions are selected according to an exploratory policy until an (iteration-specific) switching point, and from that point on, the actions are selected according to an estimation policy. A specific instance of MCTS2e, dubbed BRUE, was shown to achieve an exponential-rate reduction of simple regret over time, with the bounds on simple regret becoming meaningful after only exponential in H (in contrast to UCT's hyper-exponential in H) number of samples (Feldman and Domshlak 2012).

The specific MCTS2e sub-routines that define the BRUE algorithm are shown in Figure 2b. Similarly to UCT, each node/action pair (s, a) is associated with variables $n(s, a)$ and $\widehat{Q}(s, a)$, but with the latter being initialized to $-\infty$. BRUE instantiates MCTS2e by choosing actions uniformly at the exploration phase of the sample, choosing the best empirical actions at the estimation phase, and changing the switching point in a round-robin fashion over the entire horizon. Importantly, if the switching point of a rollout $\rho = \langle s_0, a_1, s_1, \ldots, a_H, s_H \rangle$ is σ, then only the state/action pair $(s_{\sigma-1}, a_\sigma)$ is updated by the information collected by ρ. That is, the information obtained by the estimation phase of ρ is used only for improving the estimate at state $s_{\sigma(n)-1}$, and is not pushed further up the sample. While that may appear wasteful and counterintuitive, this locality of update is required to satisfy the formal guarantees of BRUE on exponential-rate reduction of simple regret over time (Feldman and Domshlak 2012). Practice-wise, we note that BRUE is completely non-parametric, and thus neither its convergence *nor its performance* require any parameter tuning.

MCTS2e: [input: $\langle S, A, Tr, R \rangle$; $s_0 \in S$]
  search tree $\mathcal{T} \leftarrow$ root node s_0; $\sigma \leftarrow 0$
  for $n \leftarrow 1 \ldots$ time permits **do**
    $\sigma \leftarrow$ SWITCH-FUNCTION(n, σ)
    PROBE$(s_0, 0, \sigma)$
  return $\arg\max_a \widehat{Q}(s_0, a)$

PROBE (s : state, d : depth, $\sigma \in [\![H]\!]$)
  if END-OF-PROBE(s, d) **then return** EVALUATE(s, d)
  if $d < \sigma$ **then**
    $a \leftarrow$ EXPLORATION-POLICY(s)
  else
    $a \leftarrow$ ESTIMATION-POLICY(s)
  $s' \sim P(S \mid s, a)$
  $r \leftarrow R(s, a, s') +$ PROBE$(s', d+1, \sigma)$
  if $d = \sigma$ **then** UPDATE-NODE(s, a, r)
  return r

(a)

END-OF-PROBE (s : state, d : depth)
  if $d = H$ **then return** true **else return** false

EVALUATE (s : state, d : depth)
  return 0

UPDATE-NODE (s : state, a: action, r : reward)
  if $s \notin \mathcal{T}$ **then** add s to $\mathcal{T}$
  $n(s, a) \leftarrow n(s, a) + 1$
  $\widehat{Q}(s, a) \leftarrow \widehat{Q}(s, a) + \frac{r - \widehat{Q}(s,a)}{n(s,a)}$

SWITCH-FUNCTION (n : iteration, $\sigma \in [\![H]\!]$)
  return $H - ((n-1) \mod H)$ *// round robin on $[\![H]\!]$*

EXPLORATION-POLICY (s : state)
  return $a \sim \mathcal{U}[A(s)]$

ESTIMATION-POLICY (s : state)
  return $a \sim \mathcal{U}\left[\{a \mid \arg\max_{a \in A(s)} \widehat{Q}(s, a)\}\right]$

(b)

Figure 2: Monte-Carlo tree search with "separation of concerns" (a), and the BRUE specifics (b).

Empirical performance?

While BRUE guarantees exponential-rate reduction of simple regret over time, it does not make special efforts to home in on a reasonable alternative fast (Feldman and Domshlak 2012). Of course, "good" is often the best one can hope for in large sequential decision problems of interest under practically reasonable planning-time allowance. Adopting the principle of *connected* search tree expansion of the MCTS instances, recently we came up with a simple modification of BRUE, BRUE$_\mathcal{I}$, that performs as effectively under short planning times as UCT and flat MC, while preserving both the attractive formal properties of BRUE, as well as the empirical strength of the latter under permissive planning time allowances (Feldman and Domshlak 2013). Furthermore,

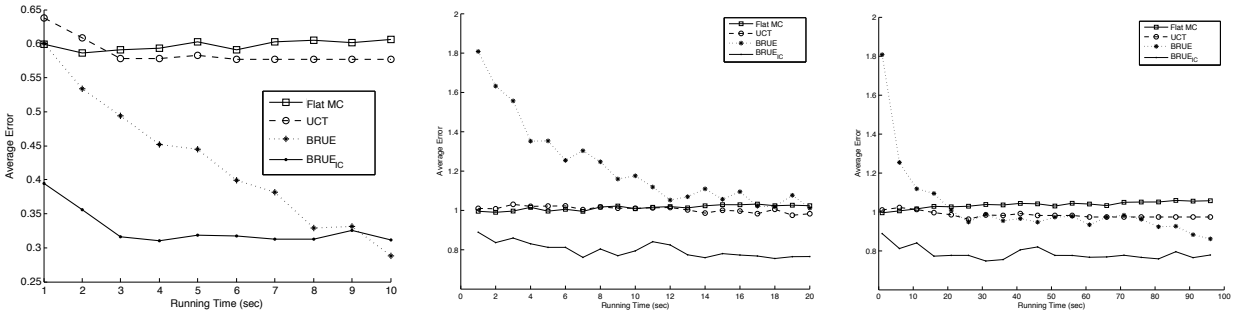

Figure 3: Absolute performance of the algorithms in terms of average simple regret on the *Sailing* domain problems with 10×10 and 20×20 grid maps.

a more involved extension of BRUE, baptized as $\text{BRUE}_{\mathcal{IC}}$, is even more effectively, favorably competing with other MCTS algorithms under short planning times in a robust manner, while still preserving exponential-rate reduction of simple regret over time. Inspired by certain insights provided by the surprising effectiveness of flat MC across IPC domains, $\text{BRUE}_{\mathcal{IC}}$ is based on selective tree expansion, conditioned on the information provided by a variant of flat MC state-value estimation.

The technical details of the $\text{BRUE}_{\mathcal{I}}$ and $\text{BRUE}_{\mathcal{IC}}$ algorithms are less important for our discussion here, and for these details we refer the reader to Feldman and Domshlak (2013). In what follows, we only outline the key high-level constructs of these two successive extensions of BRUE.

The only substantial difference between BRUE and $\text{BRUE}_{\mathcal{I}}$ is in what states are added to the search tree at each rollout, and in the related SWITCH-FUNCTION procedure. BRUE expands the search tree by the state at the switching point of the rollout, and the switching point is changing between the iterations in a round-robin fashion over the entire horizon. In contrast, $\text{BRUE}_{\mathcal{I}}$ expands the search tree by the shallowest state along the rollout up to the switching point that is not yet in the tree, and if the tree expansion happens not at the switching point, then the round-robin over switching points re-starts from the depth of one. This way, $\text{BRUE}_{\mathcal{I}}$ ensures connectedness of the search tree right from the first iteration, while BRUE constructs it search tree by growing a forest of its subtrees that gradually get connected into a single tree. Note that the latter aspect of BRUE is not a bug but a feature: in long term, this configuration of MCTS2e ensures a higher rate of convergence to the optimal action at the root. However, under severe restrictions on planning time, $\text{BRUE}_{\mathcal{I}}$ is expected to home in on a quality action faster than BRUE, and this because it expedites backpropagation of values to the root. In any case, from some point on, $\text{BRUE}_{\mathcal{I}}$ behaves identically to BRUE, ensuring exponential rate reduction of simple regret over time.

Building on top of $\text{BRUE}_{\mathcal{I}}$, $\text{BRUE}_{\mathcal{IC}}$ is devoted to even faster acquisition of a good alternative at the root via a notion of "selective tree expansion". While the protocol for selecting new states for the search tree in $\text{BRUE}_{\mathcal{IC}}$ is similar to this in $\text{BRUE}_{\mathcal{I}}$, the candidate states are added to the tree only if their predecessors pass a certain test. Until a leaf state s in the tree passes that test, its value $V(s)$ is estimated via the expected total reward of a policy sampled uniformly at random, that is, in a manner that closely resembles the basic (in principle, not sound) flat MC. Once the test is passed, the successors of s are allowed to be added to the tree, and the Q-values of actions at s are estimated as in BRUE. To ensure that all the desired convergence properties are satisfied, the specific test used by $\text{BRUE}_{\mathcal{IC}}$ to condition expansion of the search tree ensures that no leaf is placed on hold for too long.

At first view, not expanding the selected tree leaf and putting it "on hold" may appear counterintuitive. However, it is not all that simple. First, while each tree leaf can be seen as sampling a single random variable, each internal node has to sample b random variables, where b is the number of actions applicable at a state. Thus, the estimates of the random variables sampled at the leaves converge much faster than the estimates of the random variables sampled at the internal nodes. Second, while random variables estimated at an internal node s aim at representing Q-values of the actions at s, that is, the quantities of our actual interest, the quality of these estimates totally depends on the quality of the information that is backpropagated from the descendants of s. Hence, at certain stages of online planning, a slower estimation of the right thing can be less valuable than a faster estimation of its approximation. Of course, the cruder the approximation is, the eagerer we are to expand the node, and vice versa, and this is precisely what the selective expansion test of $\text{BRUE}_{\mathcal{IC}}$ is devised to assess.

In (Feldman and Domshlak 2013) we examined the simple regret reduction of BRUE and $\text{BRUE}_{\mathcal{IC}}$ under varying time budgets, using the experimental settings of Feldman and Domshlak (2012) for *Sailing* domain (Péret and Garcia 2004). Figure 3 plots the simple regret of the actions recommended by flat MC, UCT, BRUE, and $\text{BRUE}_{\mathcal{IC}}$ under different planning time windows, averaged over 300 instances of 10×10 and 20×20 grids. Importantly, no domain-specific heuristics have been employed—the algorithms were tested in their pure form.[6] First, these results show that BRUE is continually improving towards an op-

[6]That, in particular, explains the difference between the UCT's

timal solution, rather quickly obtaining results better than UCT (Feldman and Domshlak 2012). However, as it is somewhat expected from the dynamics of the value back-propagation in UCT and BRUE, UCT sometimes manages to identify reasonably good actions rather quickly, while BRUE is still "warming up". This is not so with BRUE$_{\mathcal{IC}}$. Figure 3 shows that, not only does BRUE$_{\mathcal{IC}}$ significantly outperforms UCT uniformly over time, right from the beginning of planning, but also that its simple regret reduction rate is comparable to BRUE's in the longer term. Finally, at least on this specific domain, there appears to be no substantial difference between the performance of flat MC and this of UCT within the tested planning time allowances.

We also conducted a comparative evaluation of the same set of algorithms on five MDP benchmarks from the last International Probabilistic Planning Competition (IPPC-2011), namely *Game-of-Life*, *SysAdmin*, *Traffic*, *Crossing*, and *Navigation* (Feldman and Domshlak 2013). These benchmarks appear ideal for our purpose of evaluating algorithms under tight time constraints: Most IPPC-2011 domains induce very large branching factors, and thus allow only a very shallow sampling of the underlying search tree in reasonable time. The average-over-horizon planning time was set to just five seconds. In sum, the results of this evaluation (Feldman and Domshlak, 2013—Table 1 and Figure 6) show that BRUE$_{\mathcal{IC}}$ exhibits a robustly good performance across the domains, finishing top performer on all the domains, and in fact, on most problem instances across the domains. Interestingly, similarly to what we observed in *Sailing* domain, the overall score of flat MC on the five domains used in the evaluation was roughly the same as of UCT, while adopting BRUE$_{\mathcal{IC}}$ does make a difference.

Summary

Summarizing the major emerging research directions around MCTS, Browne et al. (2012) call for improving the performance of general-purpose MCTS, as well as for improving our understanding of the behavior of various MCTS algorithms. In the current practice of online action planning with MCTS, the UCT algorithm seems to be the choice of most, domain-specific and domain-independent, solvers for various sequential decision problems. However, the question of why UCT should in fact be preferred to other online MCTS schemes remained largely unanswered.

Taking a closer look at the trail of developments that resulted in such a strong vote for UCT these days, we questioned both the rational behind this choice of platform for MCTS problem solvers, as well as a common perception that UCT is formally and/or empirically the best generic thing to do. In particular, we discuss some recent results on MCTS, and in particular, the BRUE and BRUE$_{\mathcal{IC}}$ algorithms, that both break the barrier of polynomial-rate convergence to the optimal choice over time, as well as substantially outperform UCT empirically, on numerous standard benchmarks. The journey, of course, remains brand open: There is no reason to believe that either BRUE, or BRUE$_{\mathcal{IC}}$, or even their backbone MCTS scheme MCTS2e, constitute the last word in effectiveness of basic MCTS algorithms for online planning. However, they do show that, even the most agreed upon, practices of MCTS-based planning should be questioned and investigated. In sum, we believe that further rigorous investigation of online MCTS planning is likely to be both interesting and fruitful.

performance on the *Sailing* domain we report here, in Figure 3, and the one reported in the original paper on UCT (Kocsis and Szepesvári 2006): In the latter case, UCT was run with a strong domain-specific heuristic that estimated state values within error of at most 10%.

Acknowledgements This work was partially supported by the EOARD grant FA8655-12-1-2096, and the ISF grant 1045/12.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3):235–256.

Balla, R., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *IJCAI*, 40–45.

Bjarnason, R.; Fern, A.; and Tadepalli, P. 2009. Lower bounding Klondike Solitaire with Monte-Carlo planning. In *ICAPS*.

Björnsson, Y., and Finnsson, H. 2009. CadiaPlayer: A simulation-based general game player. *IEEE Trans. on Comp. Intell. and AI in Games* 1(1):4–15.

Bonet, B., and Geffner, H. 2012. Action selection for MDPs: Anytime AO* vs. UCT. In *AAAI*.

Browne, C.; Powley, E. J.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of Monte-Carlo tree search methods. *IEEE Trans. on Comp. Intell. and AI in Games* 143.

Bubeck, S., and Munos, R. 2010. Open loop optimistic planning. In *COLT*, 477–489.

Bubeck, S.; Munos, R.; and Stoltz, G. 2011. Pure exploration in finitely-armed and continuous-armed bandits. *Theor. Comput. Sci.* 412(19):1832–1852.

Buro, M. 1999. From simple features to sophisticated evaluation functions. In *ICCG*, 126145.

Busoniu, L., and Munos, R. 2012. Optimistic planning for Markov decision processes. In *AISTATS*, number 22 in JMLR (Proceedings Track), 182–189.

Coquelin, P.-A., and Munos, R. 2007. Bandit algorithms for tree search. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 67–74.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *ICCG*, 7283.

Even-Dar, E.; Mannor, S.; and Mansour, Y. 2002. PAC bounds for multi-armed bandit and Markov decision processes. In *COLT*, 255–270.

Eyerich, P.; Keller, T.; and Helmert, M. 2010. High-quality policies for the Canadian Traveler's problem. In *AAAI*.

Feldman, Z., and Domshlak, C. 2012. Simple regret optimization in online planning for markov decision processes. *CoRR* arXiv:1206.3382v2 [cs.AI].

Feldman, Z., and Domshlak, C. 2013. Monte-Carlo planning: Theoretically fast convergence meets practical efficiency. In *UAI*.

Finnsson, H., and Björnsson, Y. 2008. Simulation-based approach to general game playing. In *AAAI*, 259264.

Finnsson, H., and Björnsson, Y. 2010. Learning simulation control in general game-playing agents. In *AAAI*.

Gelly, S., and Silver, D. 2007. Combining online and offline knowledge in UCT. In *ICML*, 273–280.

Gelly, S., and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *AIJ* 175(11):1856–1875.

Gelly, S.; Wang, Y.; Munos, R.; and Teytaud, O. 2006. Modication of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 1999. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *IJCAI*, 1324–1231.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic planning based on UCT. In *ICAPS*, 119–127.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In *ECML*, 282–293.

Kolobov, A.; Mausam; and Weld, D. 2012. LRTDP vs. UCT for online probabilistic planning. In *AAAI*.

Lorentz, R. 2008. Amazons discover Monte-Carlo. In *ICCG*, 1324.

Mannor, S., and Tsitsiklis, J. N. 2011. Mean-variance optimization in Markov decision processes. In *ICML*, 177–184.

Mausam, and Kolobov, A. 2012. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool.

Péret, L., and Garcia, F. 2004. On-line search for solving Markov decision processes via heuristic sampling. In *ECAI*, 530–534.

Powell, W. B. 2011. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley Series in Probability and Statistics. Wiley.

Robbins, H. 1952. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* 58(5):527535.

Schaeffer, J.; Hlynka, M.; and Jussila, V. 2001. Temporal difference learning applied to a high-performance game-playing program. In *IJCAI*, 529–534.

Silver, D. 2009. *Reinforcement Learning and Simulation-Based Search in Computer Go*. Ph.D. Dissertation, University of Alberta, Canada.

Sobel, M. J. 1982. The variance of discounted Markov decision processes. *J. App. Probability* 794–802.

Sturtevant, N. 2008. An analysis of UCT in multi-player games. In *CCG*, 3749.

Sutton, R. 1988. Learning to predict by the method of temporal differences. *ML* 3:9–44.

Tolpin, D., and Shimony, S. E. 2012. MCTS based on simple regret. In *AAAI*.

Winands, M., and Björnsson, Y. 2009. Evaluation function based Monte-Carlo LOA. In *ACG*, 33–44.

Zilbershtein, S. 1993. *Operational Rationality through Compilation of Anytime Algorithms*. Ph.D. Dissertation, University of California at Berkeley.