Monte-Carlo Tree Search using Batch Value of Perfect Information

Shahaf S. Shperberg
CS Department
Ben Gurion University
shperbsh@post.bgu.ac.il

Solomon Eyal Shimony
CS Department
Ben Gurion University
shimony@cs.bgu.ac.il

Ariel Felner
ISE Department
Ben Gurion University
felner@bgu.ac.il

Abstract

This paper focuses on the selection phase of Monte-Carlo Tree Search (MCTS). We define *batch value of perfect information* (BVPI) in game trees as a generalization of *value of computation* as proposed by Russell and Wefald, and use it for selecting nodes to sample in MCTS. We show that computing the BVPI is NP-hard, but it can be *approximated* in polynomial time. In addition, we propose methods that intelligently find sets of fringe nodes with high BVPI, and quickly select nodes to sample from these sets. We apply our new BVPI methods to partial game trees, both in a stand-alone set of tests, and as a component of a full MCTS algorithm. Empirical results show that our BVPI methods outperform existing node-selection methods for MCTS in different scenarios.

1 INTRODUCTION

Monte-Carlo tree search (MCTS) algorithms, such as the UCT algorithm and its many variants [Browne *et al.*, 2012], are state-of-the-art in numerous domains. A crucial phase in MCTS is the *selection phase* where a fringe node of the partially expanded tree is selected for sampling (initiating rollouts). Although UCT is a prominent approach, its node-selection criterion, based on optimization of *cumulative regret*, is actually inappropriate for move selection: it was shown in [Hay *et al.*, 2012; Feldman and Domshlak, 2013] that *simple regret* and value of information (VOI) criteria are more appropriate, and result in more efficient search. A scheme similar to the "value of computation" of [Russell and Wefald, 1991b; 1991a] can be used to define the VOI (see Section 2.2). Since a single rollout cannot find the true utility of a node, the "blinkered" scheme in [Hay *et al.*, 2012] provided bounds on the VOI for a *number of samples* at a node. Despite its success, the "blinkered" scheme has two shortcomings. First, it applies only at the first level of the tree. Second, it only considers the VOI of individual nodes.

Numerous authors attempted to optimize VOI for multiple sources of information. Such an optimization is intractable in general [Reches *et al.*, 2013; Krause and Guestrin, 2011; 2009]. Approximations thereof using myopic assumptions and greedy search are a common way to alleviate this problem [Krause and Guestrin, 2011]. The myopic-greedy schemes have a basis in theory, due to the fact that if the VOI is submodular, greedy algorithms are provably near-optimal [Krause and Guestrin, 2009; 2011; Papachristoudis and Fisher III, 2012]. However, the VOI is not submodular in general [Krause and Guestrin, 2009], thus myopic-greedy metareasoning can be far from optimal.

We address the above shortcomings by defining **batch value of perfect information** (BVPI) of multiple nodes in game trees, as a generalization of *value of computation* [Russell and Wefald, 1991a]. First, we examine the computational complexity of BVPI, and show a deterministic approximation (Section 3). Second, we introduce algorithmic variants that quickly choose a set of nodes S with high BVPI, and evaluate them on trees generated by MCTS algorithms (Section 4). Third, we present variants that quickly select nodes in S for rollouts. We provide empirical evidence of improved rollouts effectiveness without incurring too much overhead. Finally, we show how all ideas can be plugged into any MCTS algorithm by modifying only its selection phase (Section 5). Experimental results (Section 6) on two disparate domains show that our methods significantly outperform the node selection schemes used by UCT and "blinkered", as well as that of an adaptation of MGSS* [Russell and Wefald, 1991a] to MCTS.

2 BACKGROUND

This paper uses techniques from Monte-Carlo tree search, value of computation (value of information) in game trees, and conspiracy numbers. We briefly examine each below.

2.1 MONTE-CARLO TREE SEARCH

Monte-Carlo tree search (MCTS) is an algorithmic schema commonly used to search huge trees. In general, MCTS grows a search tree, using four phases: node **selection**,

node **expansion**, **simulation** (also called rollouts or sampling), and **updating** (also called backup). Algorithm 1 depicts the MCTS, following [Browne *et al.*, 2012]).

Algorithm 1: Monte-Carlo Tree Search

```
1 function MCTS(root):
2     while computation budget not exceeded do
3         v ← TreePolicy(root)
4         U ← DoRollout(v)
5         Backup(v, U)
```

TreePolicy() consists of the **node selection** and **expansion**, DoRollout() performs simulation (sampling) from the selected node v. The computation budget (line 2) can be the number of rollouts, a time limit, etc. There are numerous schemes for deciding which nodes to expand, which nodes to select, how to propagate updates, and how to do rollouts (See [Browne *et al.*, 2012] for a deep survey). A popular approach for node selection is the Upper Confidence Bounds for Trees (UCT) [Kocsis and Szepesvári, 2006], which selects a node recursively, starting from the root, down to the fringe. At each node v, select a child c which maximizes the score:

$$UCTscore(c) = Q(c) + b\sqrt{\frac{2\ln N(v)}{N(c)}} \qquad (1)$$

where $Q(c)$ is the previous average value of c, $N(v)$ is the number of past visits of v, and b is a constant. If v is a fringe node, a rollout is initiated from v.

2.2 INFORMATION GATHERING IN TREES

We are given a game tree T consisting of MAX, MIN, and CHANCE nodes (root r is a MAX node). Each leaf has known utility value distribution P_v, as in Figure 1, but its true utility $u(v)$ is unknown. Leaf utility distributions are assumed independent (the commonly used "subtree independence" assumption [Russell and Wefald, 1991a]). The notation $[p_1 : u_1, ..., p_n : u_n]$ represents P_v, meaning that p_i is the probability that the true utility is u_i.

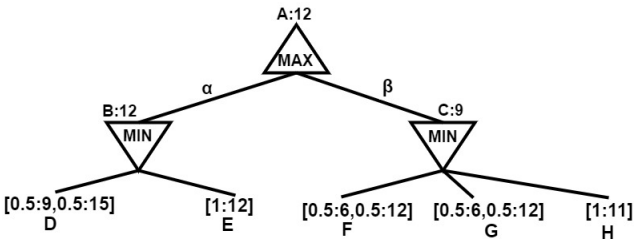

Figure 1: MIN-MAX tree with leaf utility distributions

For each leaf v of T, we can choose to perform a measurement that costs $C(v)$, obtaining further information about v, such as the true utility of v. Optionally, there is also a budget constraint that limits the total number, or cost, of allowed measurements. The problem of *optimal information gathering* is: what is the optimal policy of performing measurements, and then selecting the action at the root, so as to achieve maximum overall expected utility?

There are two standard settings of this problem. In the *batch setting*, all measurements are made in a single batch. Only then, the decision maker gets to observe the results of the measurements. In the *sequential setting*, the decision maker selects a measurement and immediately observes the result. This is repeated until a decision to stop is reached. In both settings, after stopping the measurement process, the decision maker selects the root action that achieves the best expected utility for the tree given all the observations.

In this section, as well as in Section 3, we assume that the distributions P_v are given, and that the measurements are abstract operators that have a known cost and reveal the true utility $u(v)$. But in the context of a search algorithm, T is actually the partially developed game tree, of which v is a fringe node. A measurement is done by a computation, obtaining additional (usually **noisy**) information about v by expanding it, or (in MCTS) by initiating additional rollouts from v. The initial distributions P_v are obtained by heuristics, by rollouts (as in Section 5) etc.

2.2.1 Value of Perfect Information

Suppose that the agent has developed the tree T. If no additional measurements were allowed, a MAX player should compute an EXPECTI-MINI-MAX value $\bar{U}(T)$ of the tree, treating the utility of each leaf v of T as if it were equal to its expected value. Let α be the child of the root which propagated $\bar{U}(T)$ to the root node in EXPECTI-MINI-MAX. We call this move α the "current best" (see Figure 1). In order not to introduce additional notation, as far as utility distributions are concerned, we henceforth refer to a move, and to its respective child node, interchangeably.

In a nutshell (See [Russell and Wefald, 1991a]), the *Value of (Perfect) Information* (VPI), (called *Value of Computation* in [Russell and Wefald, 1991a]) is the expected gain from picking another move β_i as a result of performing the additional measurements either under α or under β_i. Let β_1 be the next-best move after α, and denote by $\bar{U}(\gamma)$ the current expected utility of any move γ. Let S_α be a set of leaves under α, and let $p_{\alpha S_\alpha}(x)$ be the probability density function for the utility of move α, given the utility observations at the leaves S_α. The VPI for S_α is defined as:

$$VPI(S_\alpha) = \int_{-\infty}^{\bar{U}(\beta_1)} p_{\alpha S_\alpha}(x)(\bar{U}(\beta_1) - x)dx \qquad (2)$$

Intuitively, $VPI(S_\alpha)$ is the gain due to preferring β_1 over α because measurements below α revealed that the new value of α (given the observations) is worse than the current

expected value of β_1. Likewise, for a move β_i (not the current best), let S_{β_i} be a set of leaves under β_i, and denote by $p_{\beta_i S_{\beta_i}}(x)$ the probability density function for the utility of move β_i, given utility observations at leaves S_{β_i}. Then:

$$VPI(S_{\beta_i}) = \int_{\bar{U}(\alpha)}^{\infty} p_{\beta_i S_{\beta_i}}(x)(x - \bar{U}(\alpha))dx \quad (3)$$

The distributions in Eqs. 2, 3 were defined in [Russell and Wefald, 1991a] for MIN-MAX trees. Equation 4 below is an alternative statement that incorporates CHANCE nodes.

MGSS* (Meta-Greedy Single-Step) [Russell and Wefald, 1991a] uses Eqs. 2, 3, and assumes that the measured node-set S is a single node: their "single-step assumption" of defining the value of computation under the assumption that only one step of computation will be done before the final move decision (see Example 1). The scheme was extended into MGSS2 that estimates the value of computation for more than one node.

Example 1. *In Figure 1, α is the current best move, since $\bar{U}(\alpha) = 12 > 9 = \bar{U}(\beta)$, due to the MIN-MAX computation in the tree, using the expected value of leaf node distributions as a "known value". For example, the value of leaf node D is taken to be 12. Obtaining the true utility $u(v)$ of any one leaf cannot change the move selected by MAX. Thus, the VPI of any individual leaf node here is 0, so MGSS* stops further computational actions. Examining value of computation for multiple nodes under the same move (such as $\{F, G\}$) does not help in this case. But if we measure $\{D, F, G\}$, then with probability 0.5, D will show utility 9, so B will have utility 9 as well. With probability 0.25, both F and G will show utility 12, so C will have a utility of 11. So with overall probability 0.125, C will be better than B, and MAX would change the first move to C. The set $\{D, F, G\}$ has a value of computation greater than 0, and should be measured if its cost is sufficiently low. This paper extends VOI to consider such sets.*

For any set of nodes to be measured, its value of information minus the cost of the measurement is called the **net value of information**. By extension, we likewise use the terms "net VPI" and "net BVPI" (below) to mean the appropriate type of VOI minus measurement costs.

2.3 CONSPIRACY NUMBERS

Conspiracy numbers [McAllester, 1988] denote the minimal number of nodes that need to change their values so as to cause the tree's value to change to a given value u.

Example 2. *In Figure 2, the minimax value of the root is 10. In order to increase its value to any $u \in (10, 15]$, the minimal number of leaf nodes that need to change is 1 (node F), thus, there is a single conspirator required. In order to increase the root's value to $u \in (15, 20]$, the values of both E and F need to change (two conspirators), and three conspirators are required to change it to any $u > 20$.*

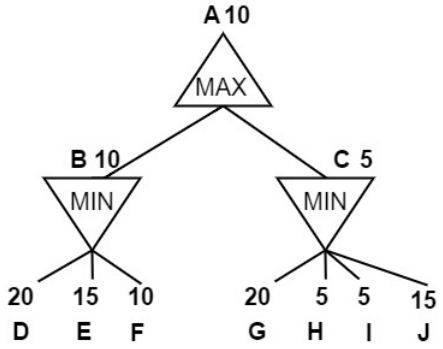

Figure 2: Conspiracy numbers in MIN-MAX trees

Originally the idea was used to focus game tree search on such conspiracy nodes. Although conspiracy numbers are defined for known leaf values, they are loosely related to the value of computation [Russell and Wefald, 1991a]. Generally speaking, a conspiracy number greater than 1 (when true leaf utilities are assumed to be equal to their expected utility as in Example 1 and Figure 1) indicates that MGSS* will see a VPI of 0 for every node, whereas the value of computation for numerous nodes may be non-zero (in which case the VPI is not submodular). In this paper we adapt conspiracy numbers in order to find sets of nodes that have a high *probability* of affecting the value at the root.

3 BATCH VALUE OF INFORMATION

We now define *Batch Value of Perfect Information* (BVPI). The *value of computation* (=VPI) in [Russell and Wefald, 1991a] was defined for nodes all under a single child of the root. BVPI is a straightforward generalization of VPI allowing measurements at arbitrary sets S of leaves. Denote by $U_S(v)$ the utility of node v, given that measurements will be performed at a set of leaf nodes S. Note that before getting the actual observed values, $U_S(v)$ is a random variable. Since measurements are assumed to be perfect, $U_S(v)$ (at leaf nodes) is distributed as P_v. Denote the children of v by $ch(v)$, the probability of a child node c of a chance node by $p(c)$, and use appropriate predicates ($LEAF(v)$ is true if v is a leaf node, etc.) to denote node types. The distribution of $U_S(v)$ is defined recursively in Eq. 4.

$$U_S(v) \sim \begin{cases} P_v & \text{LEAF}(v) \wedge v \in S \\ [1 : E_v[P_v]] & \text{LEAF}(v) \wedge v \notin S \\ \max_{c \in ch(v)}\{U_S(c)\} & \text{MAXnode}(v) \\ \min_{c \in ch(v)}\{U_S(c)\} & \text{MINnode}(v) \\ \sum_{c \in ch(v)} p(c) U_S(c) & \text{CHANCEnode}(v) \end{cases} \quad (4)$$

The *batch value of perfect information* (BVPI) for obtaining perfect information on a set S of nodes is defined as follows. Denote $U_S(\beta) = \max_i\{U_S(\beta_i)\}$, and let $p_{\alpha\beta S}(x, y)$

be the joint probability density function of $(U_S(\alpha), U_S(\beta))$ at (x, y) given the measurements at S. Then:

$$BVPI(S) = \int_{y>x} p_{\alpha\beta S}(x, y)(y - x)dxdy \quad (5)$$

Intuitively, $BVPI(S)$ is the gain due to a change of best move from α to some β_i, because observations at S revealed that the new utility of β_i is better than that of α. Note that if S is a set of leaves limited to a subtree under α then Eq. 5 reduces to Eq. 2. Likewise if S is limited to a subtree under β_i, in which case Eq. 5 reduces to Eq. 3.

In using $BVPI(S)$ (Eq. 5) below, we re-write it as:

$$\begin{aligned} BVPI(S) &= \int_{y>x} p_{\alpha S_\alpha}(x) p_{\beta S_\beta}(y)(y - x)dxdy \quad (6) \\ &= E[max(U_{S_\beta}(\beta) - U_{S_\alpha}(\alpha), 0)] \quad (7) \end{aligned}$$

where $p_{\beta S_\beta}$ is the density function of $U_{S_\beta}(\beta)$, which is the same as $U_S(\beta)$, due to subtree independence. The first equality also follows from the subtree independence, and the second from an algebraic manipulation of the terms.

To optimize (batch setting) information gathering using BVPI, one should find a set of nodes S with the highest net $BVPI(S)$. This is hard because: **(1)** We need to compute the BVPI for each such set. **(2)** There is an exponential number of potential subsets S of leaves of T.

3.1 COMPUTING THE BVPI

Theorem 1. *Computing $BVPI(S)$ for a given set of leaves S in expecti-mini-max trees is NP-hard.*

Proof (outline): by reduction from the Partition problem [Garey and Johnson, 1979] [SP12], defined as follows. Given a multi set S of integers $\{S_1, S_2, ..., S_n\}$ (w.l.o.g. $\sum_{i=1}^n S_i$ is even), is there an equal partition, i.e. a set of indices $I \subseteq [1, ..., n]$ such that $\sum_{i \in I} S_i = \sum_{i \notin I} S_i$?
The reduction uses the three-level expecti-minimax tree of Figure 3, with two-valued distributions at the leaves. The S_i in the figure are the same numbers as in the partition problem. By computing $BVPI(S)$, with S being the set of all children of the chance node, for 2 different values of u_2, we can decide the partition problem, as follows. Denote $\sigma = \sum_{i=1}^n \frac{S_i}{2n}$, the desired partition sum divided by n. Before any measurements are made, the expected utility of each uncertain leaf i is $\frac{S_i}{4}$, and thus the expected utility $\bar{U}(v)$ of the chance node is $\frac{\sigma}{2}$, which is less than u_1 and either value of u_2. So indeed MAX chooses α. If all uncertain leaf nodes are measured, then there is a non-zero probability that $u(v) > u_1$, but if we also have $u(v) > u_2$ then MIN will not pick the chance node, so the gain in such cases is limited by $u_2 - u_1$. Denote $BVPI(S)$ for the case where $u_2 = \sigma + \frac{1}{2n}$ by B_1, and for the case where

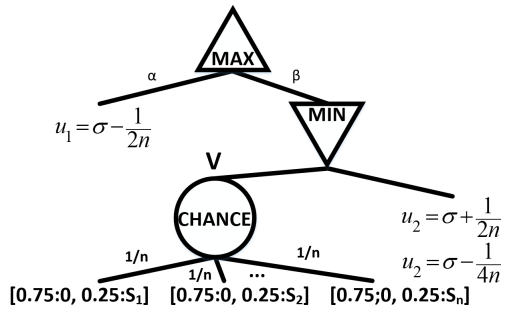

Figure 3: NP-hardness of computing BVPI: reduction

$u_2 = \sigma - \frac{1}{4n}$ by B_2. In the first case we have:

$$B_1 = P(U_S(v) = \sigma)(\sigma - u_1) + \frac{P(U_S(v) > \sigma)}{n}$$

Since the S_i are all integers, then σ and all possible values of $U_S(v)$ are integer multiples of $\frac{1}{n}$, so $P(U_S(v) \in [u_1, u_1 + \frac{1}{4n}]) = 0$. Thus, the BVPI in the second case is:

$$B_2 = \frac{P(U_S(v) > \sigma - \frac{1}{4n})}{4n} = \frac{P(U_S(v) \geq \sigma)}{4n}$$

If we can compute B_1 and B_2 in polynomial time, we can trivially solve for $P(U_S(v) = \sigma)$, which is non-zero just when the partition problem has a solution. □

Note that this reduction also implies NP-hardness of VPI (Eq. 3), as all the uncertain leaves are in the same subtree. However, if the utility of the leaves is bounded, standard *sampling* techniques can be used to approximate $BVPI(S)$. We show a *deterministic* approximation result.

Theorem 2. *Given a game tree T with finite 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, $\frac{1}{\epsilon}$, and utility span bound $U_{max} - U_{min}$.*

Proof (outline): We approximate $U_S(v)$ bottom up, in a manner similar to [Cohen *et al.*, 2015]. There, an approximate cumulative distribution (CDF) was computed for MAX-SUM trees, in a manner that bounded the Kolmogorov distance ($\max_x |F'(x) - F(x)|$) of the approximate CDF from the exact CDF. As the Kolmogorov distance is unhelpful in our case, we use a different version that provides the appropriate error bound for the expected values in Eq. 7.

Denote by $F_{U_S(v)}(x)$ the CDF of $U_S(v)$ at x. For MAX nodes we have: $F_{U_S(v)}(x) = \Pi_{c \in ch(v)} F_{U_S(c)}(x)$ due to independence. Likewise, for MIN nodes we have: $F_{U_S(v)}(x) = 1 - \Pi_{c \in ch(v)}(1 - F_{U_S(c)}(x))$. Chance nodes are weighted sums of the random variables, so we need to use convolution, which may grow the distributions' support exponentially. To overcome this problem, we bound

the support by an appropriately defined $n = f(U_{max} - U_{min}, |T|, \frac{1}{\epsilon})$ and apply a TRIM operator (see Algorithm 2) to the current $U_S(v)$, assumed to be a list of (value, probability) pairs sorted by increasing value.

Algorithm 2: TRIM operator

```
m ← |support(U_S(v))|;        U'_S(v) ← ∅;
head ← first(U_S(v));         tail ← rest(U_S(v));
while m > n and tail is non-empty do
    next ← first(tail);
    if value(next) - value(head) < (U_max − U_min)/n then
        prob(head) ← prob(head)+ prob(next);
        tail ← rest(tail);        m ← m-1;
    else
        U'_S(v) ← append(U'_S(v), head);
        head ← first(tail);       tail ← rest(tail);
return append(U'_S(v), tail) ;
```

By construction, $|\text{support}(U'_S(v))| \leq n$ after TRIM. The error introduced by TRIM is bounded by $\delta = \frac{U_{max} - U_{min}}{n}$, i.e.: $F_{U_S(v)}(x + \delta) \geq F_{U'_S(v)}(x) \geq F_{U_S(v)}(x)$ for all x.

We prove lemmas bounding combination errors: in MIN and MAX nodes errors are added, and in CHANCE nodes at worst equal to the maximum error in the children. The (low-order polynomial) function f can be chosen so that $F_{U_S(r)}(x + \epsilon) \geq F_{U'_S(r)}(x) \geq F_{U_S(r)}(x)$ for all x at the root r. These inequalities imply that the Wasserstein distance between $U_S(r)$ and $U'_S(r)$ is at most ϵ, thus $|E[U'_S(r)] - E[U_S(r)]| \leq \epsilon$. Since $|\text{support}(U'_S(v))| \leq n$, for all v, we can then compute the expectations in Eq. 7 in polynomial time. □

Note that we can also handle continuous distributions, by discretizing them into $\frac{U_{max} - U_{min}}{n}$ values, achieving similar approximation guarantees, assuming that we can efficiently compute their CDFs at any given point.

4 PRACTICAL BATCH-SELECTION

We now examine practical methods for choosing batches of nodes with a high net BVPI. Here, we treat this as a standalone problem on a partially developed tree T. In Section 5 we incorporate these methods into MCTS.

Optimizing the value of information was shown to be NP-hard even for one-level trees (see, e.g. [Reches *et al.*, 2013; Shperberg and Shimony, 2017]). Thus we are forced to introduce methods suggested by the theory, but which have no guarantees, in order to be able to meet the extremely difficult task of actually improving the runtime or quality of MCTS, which requires that we find node-sets with high BVPI in essentially negligible computation time.

4.1 HIGH-BVPI SETS: USING CONSPIRACIES

This scheme uses an idea based on the above described conspiracy numbers [McAllester, 1988]; it is possible to quickly detect node-sets involved in minimal conspiracies. We desire instead a probabilistic version that can quickly detect node-sets S that have a significant contribution to Eq. 6, and hence should have a high net $BVPI(S)$. Note that we do not need to find even a nearly optimal set, it is sufficient for our search application to find a reasonably good set, and not to return an empty set when sets with a high net BVPI are available.

Algorithm 3: C-VIBES Selection Scheme

```
function SelectNodes(root):
    foreach V ∈ 𝒱 do
        storeProbabilities(α, V)
        foreach Node c ∈ ch(root) − {α} do
            storeProbabilities(c, V)
    V, V', c ← values which optimize Equation 8
    S' ← ∅; OpenList.init(α); OpenList.insert(c)
    while OpenList not empty do
        v ← OpenList.pop()
        if v is a fringe node then
            S' ← S' ∪ {v}
        else
            Val ← { V   if v ∈ subtree of α
                  { V'  otherwise
            foreach c ∈ ch(v) do
                if φ[v, Val] ∉ {0, 1} and (CHANCEnode(v) or
                φ[v, Val] == φ[c, Val]) then
                    OpenList.insert(c)
    return S'
function storeProbabilities(v,Val):
    foreach c ∈ ch(v) do
        storeProbabilities(c, Val)
    φ[v, Val] ← { P̂(U_S(v) ≤ Val)   if v ∈ subtree of α
                { P̂(U_S(v) ≥ Val)   otherwise
```

We adapt the conspiracy scheme by defining a probabilistic variant of conspiracy numbers, and then by evaluating a modified version of Eq. 6 where we replace integration by maximization, using as S the entire set of leaf nodes, and $U_S(v)$ as in Eq. 4. In MIN-MAX trees, the probability that the value of node v will increase to at least Val if we measure nodes S is: $\uparrow \phi(v, Val) = P(U_S(v) \geq Val)$. Likewise the value v will decrease to be at most Val with probability: $\downarrow \phi(v, Val) = P(U_S(v) \leq Val)$. Now perform the following optimization (note the similarities with Eq. 6), and use its optimum as done within Algorithm 3.

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

The $\hat{P}$ in the algorithm are probabilities approximated as in Theorem 2, except that for CHANCE nodes, our BVPI approximation is still too slow for the desired real-time performance, so we use: $\hat{P}(U_S(v) \leq Val) \approx \sum_{c \in ch(v)} p(c) P(U_S(c) \leq Val)$ instead of convolution in our implementation of the conspiracy scheme. Example 3 below depicts one such node-set recovery instance.

Table 1: BVPI in Trees Generated by UCTO in the CTP (left) and in StarCaft (right)

CTP										StarCraft									
#	Tree Size	Tree Height	Max BF	Full Tree		Conspiracy		Greedy		#	Tree Size	Tree Height	Max BF	Full Tree		Conspiracy		Greedy	
				BVPI	T (**ms**)	BVPI	T (**ms**)	BVPI	T (**ms**)					BVPI	T (**ms**)	BVPI	T (**ms**)	BVPI	T (**ms**)
1	30	9	3	**359**	217	296	**0.28**	92	**0.28**	11	30	4	20	**283**	198	234	0.25	205	**0.24**
2	45	13	4	**453**	443	382	0.29	215	**0.28**	12	45	4	20	**356**	413	**356**	**0.25**	**356**	**0.25**
3	61	15	3	**126**	1868	103	**0.29**	19	**0.29**	13	60	6	20	**326**	1719	312	**0.28**	283	**0.28**
4	106	25	4	**527**	356872	480	**0.34**	319	**0.34**	14	100	11	20	**147**	299814	**147**	0.33	105	**0.31**
5	125	20	6	**812**	7129870	681	0.36	681	**0.35**	15	125	11	20	**63**	7088815	**63**	**0.36**	**63**	**0.36**
6	153	42	5	**219**	20676623	**219**	0.4	71	**0.38**	16	150	12	20	**96**	19938631	71	0.39	0	**0.38**
7	1018	33	6	T/O	T/O	**113**	0.78	0	**0.73**	17	1000	23	20	T/O	T/O	**33**	0.72	**33**	**0.70**
8	3077	29	6	T/O	T/O	**65**	1.89	13	**1.88**	18	3000	28	20	T/O	T/O	**116**	1.85	42	**1.82**
9	6820	37	5	T/O	T/O	**53**	4.2	0	**3.9**	19	7000	42	20	T/O	T/O	**25**	4.33	7	**4.01**
10	15321	69	7	T/O	T/O	**277**	8.5	12	**8**	20	15000	79	20	T/O	T/O	**47**	8.41	0	**7.96**

4.1.1 Batch Selection Algorithms

We evaluate three batch selection algorithms:

(1) Full tree (FT): Exhaustively compute net BVPI (using BVPI approximation algorithm from Theorem 2) for every subset S of T's leaves; pick S with the highest net BVPI.

(2) Greedy (G): Start S as an empty set. Estimate the net VPI for every leaf node not in S, and add the best to S. Repeat until no node has a positive net VPI.

(3) Conspiracy (C): The conspiracy-based scheme described above. As the optimization in Eq. 8 is still too slow to perform in real time for MCTS, our implementation optimizes over only a few possible values of V, V' in Eq. 8. In the experiments we used the value set:

$$\mathcal{V} = \{0.8\bar{U}(\alpha), 0.9\bar{U}(\alpha), 0.95\bar{U}(\alpha), \bar{U}(\alpha), 1.05\bar{U}(\alpha)\}$$

Example 3. *In figure 1, α is the current best move, since $\bar{U}(\alpha) = 12 > 9 = \bar{U}(\beta)$. The leaf node set $S = \{D, F, G\}$ is the only one to potentially change the best action from α to β, thus the only set with $BVPI(S) > 0$. FT estimates the BVPI exhaustively, and thus will correctly return S. Greedy suffers from myopic assumptions like MGSS*. Recall that in this example, the VPI of every individual node is 0, hence, Greedy will terminate without finding S and return an empty set. The Conspiracy scheme optimizes Equation 8 in order to find a batch to sample. The optimal solution is achieved with $V' = 11$ and $V = 9$. Using these values, $\downarrow \phi(\alpha, V = 9) = 0.5$, obtained by picking D, and $\uparrow \phi(\beta, V' = 11) = 0.5$ obtained by picking both F and G. Therefore, Conspiracy returns $\{D, F, G\}$ as desired. Note that the values $V' = 11$ and $V = 9$ are outside the range $\mathcal{V}$ used by our actual Conspiracy implementation for optimizing Equation 8. Despite that, using $V = 0.8\bar{U}(\alpha) = 9.6$ and $V' = 0.9\bar{U}(\alpha) = 10.8$, the implementation still finds and returns the correct node-set $S = \{D, F, G\}$.*

4.2 EXPERIMENTS ON GAME TREES

To get a realistic game tree, we used a snapshot of a partially developed game tree T' from a MCTS. T' is cut off so that its fringe nodes become the leaves of our tree T. Values previously returned by rollouts form an empirical distribution at each fringe node (now leaf) v; this distribution is assumed to be the actual distribution P_v of leaf node utilities. E.g. if we had 2 rollouts with value 10, and 6 rollouts with value 20, (from v) then we set $P_v = [0.25 : 10,\ 0.75 : 20]$. Measurement costs assumed are given by Equation 9 (Sec. 5). We obtained trees from the following 2 domains.

Domain 1: (stochastic) Canadian traveler problem [Papadimitriou and Yannakakis, 1991]; we are given a weighted graph $G = (V, E, w)$ where $w : E \to R^+$. Each edge $e \in E$, has a known probability $p(e)$ of being blocked. The agent starts at vertex $s \in V$, and must reach a vertex $t \in V$. Whether an edge is blocked becomes known upon reaching an incident vertex. The problem is to find a policy that minimizes the expected travel distance (sum of w) before reaching t (the utility here is minus the distance). The decision version of the stochastic CTP is PSPACE complete [Fried *et al.*, 2013]. [Eyerich *et al.*, 2010] present an effective UCT-based algorithm called UCTO which randomly searches the belief states in the given program instance, and generates EXPECTI-MAX trees.

Domain 2: StarCraft, a Real Time Strategy (RTS) game by Blizzard Entertainment, a popular AI competition and research platform. Our StarCraft experiments used code by [Justesen *et al.*, 2014] for playing StarCraft battles against an opposing team. Their code uses a UCT-based MCTS algorithm that generates MAX-MIN trees during the search.

Approximate net-BVPI and CPU time for 10 instances appear in Table 1(left) for CTP and in Table 1(right) for StarCraft. In both domains, FT timed out (T/O denotes timing out after 6 hours) in large instances. Greedy is the fastest, but the BVPI it achieved was only 64% of that of FT on average (over instances where FT did not time out). Conspiracy had the best balance between time and effectiveness. Its BVPI was 89% of that of FT on average, and was only slightly slower (2%) than Greedy. In fact, the conspiracy scheme was also the best in the MCTS below.

5 PLUGGING BVPI INTO MCTS

The next challenge is to use the theory of BVPI for selecting nodes on which rollouts will be performed.

5.1 SELECTING THE ROLLOUTS

We describe a number of BVPI-based selection schemes and evaluate them as well as other related schemes by plugging them into existing UCT based implementations (denoted as "the host MCTS algorithm") by changing only the selection phase. Each scheme has a different tradeoff between metareasoning overhead (time for choosing the nodes to sample), and the effectiveness of the resulting rollouts. The first scheme we implemented is the **Blinkered** scheme from [Hay *et al.*, 2012], which replaces the UCT criterion by a VOI bound in the first tree level, resorting to UCT at deeper levels.

We now introduce our BVPI-based schemes, called *Value of Information of a Batch Efficient Selection* (VIBES). The metareasoning overhead in VIBES is relatively high, thus too expensive to use to decide every single rollout. Therefore, after a node v is selected for sampling we perform N rollouts from v. This amortizes the metareasoning overhead over N rollouts. Additionally, some of the schemes below first select a batch S of fringe nodes to sample. This set can be of any size (up to the number of leaf nodes). In order not to over-commit a large number of rollouts in such cases, we choose K nodes from S with the highest individual net VPI, and an additional K randomly selected from S to a total of at most $2K$ nodes for rollouts. The additional K random samples are used to allow measurements on nodes with VPI = 0 inside the selected batch. Below we call this *the batch selection method*, $BSM(N, K)$. Values of $N = 3$, $K = 5$ proved to produce a good balance. That is, K was set to be roughly one quarter of the size of a typical set S observed in a few trial runs. Then N was set such that $2KN$ rollouts per decision delivered a metareasoning overhead of roughly 15% of the runtime.

(1) Full tree VIBES (FT-VIBES): Exhaustively check all possible sets of fringe nodes in the tree, to find a set S with the highest net $BVPI(S)$. Then use $BSM(N, K)$ on S.

(2) First level VIBES (FL-VIBES): Estimate the net BVPI for every subset of the root's children. Choose the subset S which maximizes net $BVPI(S)$. Within S, choose a child c with the maximal individual net VPI. At deeper levels, revert to selection as in the host MCTS algorithm (e.g., the UCT formula if host=UCTO).

(3) Recursive first level VIBES (RFL-VIBES): Use FL-VIBES to select node c at the first level. Then, recursively call FL-VIBES on c until reaching a fringe node.

(4) Blinkered VIBES (B-VIBES): Perform the blinkered algorithm [Hay *et al.*, 2012] until blinkered decides to halt. Then, resort to performing FT-VIBES.

(5) Conspiracy VIBES (C-VIBES):

Select the node-set S using the Conspiracy scheme (Algorithm 3). Then use $BSM(N, K)$ on S.

In all the above methods: (a) generate some random samples to gather statistics before applying VIBES (typically 1% of the sampling budget), (b) stop the entire sampling process if the sampling budget (number of rollouts per move, or a time limit per move) is exhausted or if the scheme did not find a set S with positive net BVPI.

We also tried greedy schemes. The *basic greedy* scheme (G) repeatedly chooses the node v with the greatest individual net VPI and performs rollouts from v. This halts when the net VPI of v is non-positive or the simulation budget has been reached. This method is thus essentially the same as the Meta-Greedy Single Step (MGSS*) method of [Russell and Wefald, 1991a] applied to MCTS.

Basic greedy is fast and performs well when the BVPI is submodular. Otherwise, it suffers from premature stopping, i.e. fails to detect sets of nodes with a high combined BVPI. In order to take advantage of the speed of the greedy scheme but avoid premature stopping, we can combine it with any of the above schemes (X), as follows. Run the greedy scheme. Once the greedy scheme decides to stop due to low VPI, revert to scheme X to decide on any additional samples as long as the budget allows. We denote these combined schemes by G-X (e.g, G-FT-VIBES).

Finally, we need to address the issue of fringe node distributions used in the BVPI, and the resulting distributions after potential additional rollouts. Ideally, use a Bayesian scheme for the distributions: have some prior, and compute distributions given past rollouts, and distributions over the conditional expectation of fringe node utilities given additional potential rollouts. This issue is beyond the scope of this paper, and even if somehow this is done, it would not be obvious how to do so in real time. Instead we performed the following. We assumed that the fringe node distributions P_v are the empirical utility distributions of past rollouts, which is the reason for the initial "statistics gathering" rollouts mentioned above. We then compute the BVPI values as if rollouts from a node result in a perfect measurement of the fringe node utility. As the latter assumption is obviously incorrect, we compensated by modifying the measurement cost of a node. (Note that our BVPI schemes are not sensitive to multiplying all costs and VOI values by a constant, so this compensation makes sense.) Although the latter is somewhat of a hack, it works in practice in a way that is not too sensitive to tunable parameters.

Thus, in the above schemes, as the measurement cost $C(v)$ in the computation of the net BVPI we used:

$$C(v) = \frac{C\sqrt{N(v)}}{B - \bar{U}(\alpha)} \tag{9}$$

with $N(v)$ as in Eq. 1, B the maximal utility bound on the solution. The rationale is: if the $N(v)$ (=number of previous rollouts) is a large number, it will require more future

Table 2: Average results for CTP. The standard deviation was at most 0.36%.

Settings		UCTO	Blink'd	FL-VIBES	RFL-VIBES	FT-VIBES	B-VIBES	C-VIBES	Greedy	G-FL-VIBES	G-RFL-VIBES	G-FT-VIBES	G-C-VIBES
n	p												
30	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
	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
	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
	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
	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
60	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
	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
	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
	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
	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
100	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
	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
	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
	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
	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

rollouts to change the overall average of all the rollout values by a significant amount (on the order of $B - \bar{U}(\alpha)$). C is empirically determined over a few instances. Results were not very sensitive to C, we used $C = 96$ in our experiments. A more disciplined treatment of the costs is a non-trivial issue beyond the scope of this paper, as the computation time is not on the same scale as the game utilities.

6 EMPIRICAL EVALUATION: MCTS

We now report results where the selection schemes were plugged into existing MCTS algorithms for two domains.

6.1 CANADIAN TRAVELER PROBLEM (CTP)

CTP experiments were on Delaunay graphs following [Bnaya *et al.*, 2009; Eyerich *et al.*, 2010]. Instance parameters were n (number of vertices), and p (probability for each edge to be potentially blocked). For each potentially blocked edge e, $p(e)$ is chosen uniformly from the range $[0, 1)$. Edge travel costs were random, uniform from $\{1, \ldots, 500\}$.

We compared the original UCTO code [Eyerich *et al.*, 2010], with the same code where the node-selection function (a UCT formula variant) was replaced by one of the above described methods. All the algorithms were evaluated as follows. 10 instances were generated for each pair of $n \in \{30, 60, 100\}$ and $p \in \{0.2, 0.4, 0.6, 0.8, 1\}$. We ran every instance 100 times using a time limit of 3 seconds to decide on each move.

Table 2 shows the average cost of the path traveled by the agent for the different algorithms. As can be seen, all the proposed schemes outperformed UCTO, except for FT-VIBES (see below). C-VIBES delivered a good rollout selection at relatively little overhead, thus the best overall performance: a major improvement of more than 15% in the **path-cost** of the resulting solutions over UCTO.

In order to gain additional insight, we chose a typical instance where computing the optimal policy by value iteration (VI) over the belief space was feasible. We examined the runtime and quality of the results vs. the optimal under two budget constraints. **(Const-1:)** a time limit of 3 sec/move (as in Table 2); results appear in Table 3. Here, FT-VIBES timed out after only a few samples. Conspiracy-VIBES was the best here (4.5% from optimal), as it produces good sampling decisions with only a modest overhead. **(Const-2:)** a rollout limit (but no time limit). Runtime vs. path-costs results appear in Figure 4 for a 10K rollout limit for all algorithms. Additional runs with different limits for UCTO and C-VIBES are also shown, labeled as "C-VIBES-3K" (limit 3,000 rollouts per move) etc. C-VIBES-3K already gets better path-cost than UCTO-45K and runs 10 times faster. Furthermore C-VIBES-20K, seems to have found the optimal policy for this instance in many of the runs.

It is also of interest that among all algorithms run with 10,000 samples, FT-VIBES was the best w.r.t. policy quality, as expected. So the BVPI scheme done exhaustively indeed resulted in the most efficient selection of nodes and samples if the meta-reasoning overhead is ignored. However, this scheme is seen here to be completely useless for MCTS due to its huge meta-reasoning overhead. Another interesting point is that for 10,000 samples the greedy scheme was already better than UCT, but only slightly. We conjecture that this is due to cases where the BVPI is not submodular. This is supported by the results for the greedy hybrids, which select further nodes after greedy decides that no nodes should be selected. All the greedy hybrids do better than greedy w.r.t. policy quality, usually with negligible additional overhead.

Table 3: 30 node graphs, p=0.2, time limit 3 sec/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

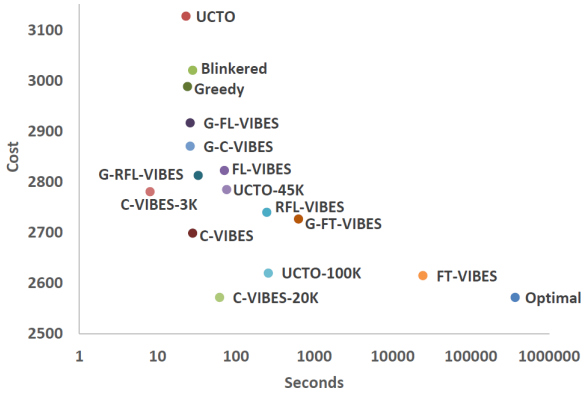

Figure 4: 30 node graphs, p=0.2, w. 10,000 rollouts/move

6.2 STARCRAFT BATTLES

Our experiments used JarCraft, an open-source java StarCraft combat simulator as well as the search algorithms from [Justesen *et al.*, 2014]. These algorithms use UCT on different action spaces as follows:

(1) UCTCD: a UCT variant which handles simultaneous and durative actions. Possible actions are sets of unit commands. [Churchill and Buro, 2013].

(2) Script-based UCTCD: an extension of UCTCD allowing both unit commands and scripts as possible actions [Justesen *et al.*, 2014]. Also presented there were **(3)**, **(4)**: **Cluster-based UCTCD**: two improvements of UCTCD that cluster units in order to decrease the number of possible actions.

We modified these UCT-based algorithms, replacing their node-selection by Blinkered, Greedy, and C-VIBES. The test scenario is based on [Justesen *et al.*, 2014]: each competing algorithm in each run controls $\frac{n}{2}$ Protoss Zealots (close combat unit) and $\frac{n}{2}$ Protoss Dragoons (ranged combat unit) vs. an opposing team of equal size controlled by another algorithm. The units are first lined up by type and then scattered randomly. For each army size n we chose the UCT variant with the best performance according to [Justesen *et al.*, 2014].

Representative results appear in Figure 5 where Blinkered, Greedy, and C-VIBES competed against UCT. C-VIBES was the best, averaging a 77% win rate against UCT (see Figure 5), though both Blinkered and Greedy were also consistently better than UCT. C-VIBES also defeated Greedy and Blinkered in head-to-head matches (not shown).

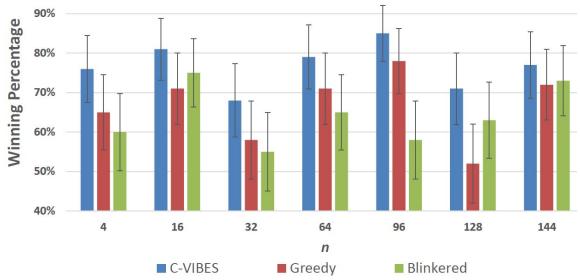

Figure 5: Win rates vs. UCT (100 games). Error bars are 95% confidence intervals.

7 CONCLUSION

Optimizing VOI for individual nodes has been shown to improve allocation of rollouts [Hay *et al.*, 2012; Feldman and Domshlak, 2013], as well as backups [Feldman and Domshlak, 2013] in MCTS. Estimating the VOI for individual nodes is too limiting. We suggested a method based on value of computation that considers large batches, and suggested several effective ways to approximately optimize them. While we confirmed that previously suggested VOI methods (MGSS* and blinkered) outperform UCT, our BVPI-based selection schemes plugged into MCTS implementations outperformed them all.

Although our measurements were assumed to be rollouts in a MCTS, the analysis in Section 3 and the batch selection methods in Section 4 may be applicable to other information-gathering operations; such as computing a static heuristic evaluation function at v, expanding v, or even real-world physical measurements (whenever the latter is meaningful).

Future improvements to our schemes are possible. First, a more disciplined way to estimate the cost of computations (e.g. by learning) would be beneficial. Second, a better defined distribution over fringe node utilities given the future rollouts is desired, such as through Bayesian updating, or estimation methods from [Feldman and Domshlak, 2013], which is essentially orthogonal to our paper, can be attempted.

Acknowledgments

Supported by ISF grant 417/13, and by the Frankel Center. We thank the authors of [Justesen *et al.*, 2014; Eyerich *et al.*, 2010] for providing their code.

References

[Bnaya *et al.*, 2009] Zahy Bnaya, Ariel Felner, and Solomon Eyal Shimony. Canadian traveler problem with remote sensing. In *Proc. of IJCAI*, pages 437–442, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[Browne *et al.*, 2012] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[Churchill and Buro, 2013] David Churchill and Michael Buro. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *2013 IEEE Conference on Computational Inteligence in Games (CIG), Niagara Falls, ON, Canada, August 11-13, 2013*, pages 1–8, 2013.

[Cohen *et al.*, 2015] Liat Cohen, Solomon Eyal Shimony, and Gera Weiss. Estimating the probability of meeting a deadline in hierarchical plans. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1551–1557. AAAI Press, 2015.

[Eyerich *et al.*, 2010] Patrick Eyerich, Thomas Keller, and Malte Helmert. High-quality policies for the Canadian traveler's problem. In *In Proc. AAAI 2010*, pages 51–58, 2010.

[Feldman and Domshlak, 2013] Zohar Feldman and Carmel Domshlak. Monte-Carlo planning: Theoretically fast convergence meets practical efficiency. In Ann Nicholson and Padhraic Smyth, editors, *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. AUAI Press, 2013.

[Fried *et al.*, 2013] Dror Fried, Solomon Eyal Shimony, Amit Benbassat, and Cenny Wenner. Complexity of Canadian traveler problem variants. *Theor. Comput. Sci.*, 487:1–16, 2013.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness*, page 190. W. H. Freeman and Co., 1979.

[Hay *et al.*, 2012] Nicholas Hay, Stuart J. Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. In Nando de Freitas and Kevin P. Murphy, editors, *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pages 346–355. AUAI Press, 2012.

[Justesen *et al.*, 2014] Niels Justesen, Balint Tillman, Julian Togelius, and Sebastian Risi. Script- and cluster-based UCT for StarCraft. In *2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26-29, 2014*, pages 1–8, 2014.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *ECML*, pages 282–293, 2006.

[Krause and Guestrin, 2009] Andreas Krause and Carlos Guestrin. Optimal value of information in graphical models. *J. Artif. Intell. Res. (JAIR)*, 35:557–591, 2009.

[Krause and Guestrin, 2011] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM TIST*, 2(4):32, 2011.

[McAllester, 1988] David Allen McAllester. Conspiracy numbers for min-max search. *Artificial Intelligence*, 35(3):287 – 310, 1988.

[Papachristoudis and Fisher III, 2012] Georgios Papachristoudis and John W. Fisher III. Theoretical guarantees on penalized information gathering. In *Statistical Signal Processing Workshop (SSP)*, pages 301–304, Aug 2012.

[Papadimitriou and Yannakakis, 1991] Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991.

[Reches *et al.*, 2013] Shulamit Reches, Ya'akov (Kobi) Gal, and Sarit Kraus. Efficiently gathering information in costly domains. *Decision Support Systems*, 55(1):326–335, 2013.

[Russell and Wefald, 1991a] Stuart J. Russell and Eric Wefald. *Do the right thing - studies in limited rationality*. MIT Press, 1991.

[Russell and Wefald, 1991b] Stuart J. Russell and Eric Wefald. Principles of metareasoning. *Artif. Intell.*, 49(1-3):361–395, 1991.

[Shperberg and Shimony, 2017] Shahaf S. Shperberg and Solomon Eyal Shimony. Some properties of batch value of information in the selection problem. *J. Artif. Intell. Res. (JAIR)*, 58:777–796, 2017.