Generalized Model for Rational Game Tree Search*†

Yan Radovilsky and Solomon E. Shimony
Comp. Sci. Dept., Ben Gurion University
P. O. Box 653, 84105 Beer-Sheva, Israel
{yanr,shimony}@cs.bgu.ac.il

Abstract – *Decision-theoretic meta-reasoning is a well known scheme for controlling search that has been shown to be advantageous in numerous domains, including real-time planning and acting, and game-tree search. Although in numerous adversarial games, such as chess, brute-force search currently emerges as the best contender, there is still scope for planning in some situations.*

In order to take advantage of both schemes, we merge the planning and exhaustive search schemes through meta-reasoning. Approximate value of information is used to decide which of the types of computation operator to apply, and where. This is done by generalizing the Best Play for Imperfect Player (BPIP) search control model of [1] to allow for planning steps, as well as game-tree search steps. A rudimentary system employing these ideas for chess was implemented, and preliminary empirical results are promising.

Keywords: Decision-theoretic control of search, game tree search, planning in games.

1 Introduction

Employing decision-theoretic meta-reasoning during search is the rational way to control search [3, 4]. This framework was applied in several research publications, such as [1, 2] towards game-tree search, in many cases with considerable success.

An alternate to position-based game tree search is to search through strategies [5]. Abstracting away from details of the specific positions results in applying planning techniques to games, an idea that was used, for example in the Belle chess playing program [5]. The rationale behind using planning rather than brute-force game-tree search is that humans use abstractions and planning, for the most part, rather than perform exhaustive game tree search. Potentially, abstractions can improve performance by avoiding irrelevancies and repeated consideration of (almost) equal positions. Yet applying these abstract planning (AP) techniques was only partially successful, in some domains. It is well known that, e.g. for chess playing, the best computer chess programs (Deep Blue and successors) essentially performed variants of brute-force game-tree search, rather than planning. The latter observation has, in the large, caused some loss of interest in planning techniques in games, with the exception being games where the branching factor is too large to use game-tree search.

And yet it seems intuitive that AP should have a role to play, in *addition* to game-tree search, rather than *instead* of game-tree search. The idea is to use a planner to help guide the search - the planner can quickly suggest moves that appear good long-term moves, rather than rely on deep search to reach the same conclusion. The candidates suggested by a planner can then become the foci for the game-tree search algorithm, resulting in deeper searches where (presumably) the relevant moves reside. However, it is not really obvious how such an integration can be performed - the following complicated questions need to be addressed along the way:

1. When and where do we apply planning, vs. standard search?

2. How do we apply results from one technique to advantage in the other technique?

Using a meta-reasoning framework, the answer to the first question is clear (though *not trivial*), and some ideas on how to answer the second question become evident. Basically, the technique to apply at each step is the one that, in expectation, provides the most valuable information on choice of move(s), after deducting the cost of applying the operation. Integrating results of the reasoning operators is more difficult, and in this paper we explore just one of its aspects - a planner can affect the evaluation of one or more nodes in the (partial) search tree. One reason why this scheme is non-trivial is that in order to make meta-reasoning reasonably efficient, several strong assumptions are made in the research literature. For example, the single-step assumption, the meta-greedy assumption, and the subtree-independence

*The research was partially supported by the Lynn and William Frankel Fund for Computer Science, and by the Paul Ivanier Center for Robotics and Production Management at BGU.

assumption are common [1]. But since the reason for doing planning in our framework is in order to affect several node evaluations in search tree, after which further search operations will be performed, clearly the above standard assumptions will be violated. This will require somewhat different methods, suggested in this paper.

As argued above, incorporating AP into a game-tree exploration process has clear advantages, but non-trivial. We treat the AP as a special case of the value function (VF) adaptation process. This ability can be very useful for more general operations that have a similar effect on precision of the local VF, such as deep position analysis. We start off with the Best Play for Imperfect Player (BPIP) model of [1], which is a model for game-tree search that picks a node to expand in the game tree based on an estimated value of information for expansion of the node. Our reason for starting off with this model is that the authors found a way to relax the single-step assumption, which we will also need to do. We then generalize their scheme, in order to incorporate planning steps.

The rest of the paper is organized as follows. Section 2, provides background on BPIP. Section 3 describes our extension to BPIP, and assumptions we make in order to make the model manageable. In Section 4 we discuss more specific assumptions made in our implementation of the model, and some details on a rudimentary domain-specific planner we use for the game of chess. Section 5 presents preliminary experimental results for the combined system.

2 Background

We begin with an overview of the basic BPIP model (refer to [1, 2] for details.

Definition 1 *Best play for imperfect players (BPIP) is a move choice policy in which the value of each node in the game tree is given as follows:*

$$V(\eta) = \begin{cases} \max_{m \in Ch(\eta)} -V(m) & \eta \text{ is a root} \\ E_\eta & \eta \text{ is a leaf} \\ \sum_{m \in Ch(\eta)} -Pr(m)V(m) & \text{otherwise} \end{cases} \quad (1)$$

where $Ch(\eta)$ are the immediate descendents (child nodes) of η. $Pr(m)$ is the probability that in BPIP starting from η, the player to move will make move m, and E_η is the expected desirability of the final score to be obtained in BPIP starting from η.

"Best play" means to pick the child of the root with the greatest (negated) value, as your move.

Definition 2 *Depth-free independent staircase approximation (DFISA) is a concrete probabilistic model for BPIP, which assumes the following:*

1. *The evaluation function and the expected desirability E_η fully describe the probability distribution P_η of opinion changes, where $P_\eta(x) = y$ means that upon deeper search of η, our opinion about the expected desirability of η will be x, with probability y. The term "staircase" is due to the assumed shape of the Cumulative Distribution Function (CDF).*
2. *The probability of various opinion changes at a given node is independent of whether we expand that node one level or more than one level (depth-free approximation).*
3. *The opinion-change distributions are independent at different leaves.*

Definition 3 *The expected utility with respect to a move i is:*

$$Q^i = \int_0^\infty P(\text{Best is better than move } i \text{ by } q)\ q\ dq \quad (2)$$

We assume that moves are ordered by their current expected value (starting from the best move). Thus, Q^1 reflects the expected utility of knowing the exact values of all of the leaves in the game (partially expanded) tree.

Definition 4 (Q step size - (QSS)) *The quality step size for node x is:*

$$A_L(x) = \sum_{j=1}^{n} p_j |Q^1 - Q_L^1(j)| \quad (3)$$

where j runs over n spikes (x_j, p_j) of the leaf L distribution, and $Q_L^1(j)$ refers to the value of Q^1, when L's distribution is replaced by a spike at x_j.

QSS was chosen by Baum and Smith to be the main criterion for the leaf expansion (LE) relevance. The basic algorithm (without the special speed-up tricks, which they also implemented) appears below:

1. Start with a search tree S consisting of the root and its m children.
2. While $Q^1 >$ ExpansionTimeCost() do:
   (a) Select leaf L with greatest QSS.
   (b) Expand leaf L and update all values in S.
3. Return the child of the root with the greatest expected desirability.

According to Baum&Smith [1, 2], good results were obtained vs. standard alpha-beta searchers, and thus their scheme provides a good starting point for our work.

3 Our Extensions

Although, as stated above, BPIP DFISA is a good framework to start from, it is limited in certain aspects that need to be extended. We begin by discussing these limitations, and continue to the extensions introduced in our scheme.

3.1 Limitations of QSS

BPIP DFISA cannot be integrated with AP, unless appropriately generalized. Integrating VF adaptation into a game-tree search model requires the latter to support explorative operations that may have the following properties:

1. Operations have variable (non-uniform) time cost (TC).
2. Operations affect the desirability distribution of its future successors, as well as the desirability distribution of the currently existing leaf node.

QSS fully ignores the TC of the operation, which is unrealistic, when there are several different types of operation (for example, planning may well take more time, and thus be more costly, than standard node expansion). Therefore the first property forces us to modify the relevance criterion to reflect the variability of TC.

QSS also cannot reflect "future effects", if the LE is assumed to immediately cause the distribution over the local value to collapse into a single spike. Under such conditions, operations like AP, which mainly affect the local VF (and through that - the effectiveness of future expansions of this leaf's sub-tree) will never be recognized as more relevant than a simple expansion of the same leaf. However, direct consideration of all possible "future effects" requires significant sophistication of the search model by dealing with "distributions over distributions" [1] or search in a space of "computational sequences" [3]. Unfortunately both extensions seem to be intractable. We try to avoid the "complexity explosion" by modifying the "depth-free" assumption and by allowing some operations to affect the future expansion time of the relevant nodes.

3.2 Planning and evaluation model

In our model the time cost of an operation depends on the type and the application point of the operation. We allow two types of operation: an adaption (planning) step at a node x, which we denote by $a(x)$, and a full expansion step at a node x, which we denote by $e(x)$. By a full expansion step we mean performing a sequence of leaf expansions, all applied to descendants of x, until the exact value of node x is achieved. In this way we relax the depth-free assumption used in DFISA. Note, however, that selecting an operation $e(x)$ to perform does not mean that the algorithm will *actually* expand the node to get exact values, in fact it will only generate the immediate descendants of x and then decide about the next step. We assume knowledge of expected time to complete an operation s (denoted by $ET(s)$), and that the time cost function TC is a known function of this expected time. The cost of a computational operator is thus TC($ET(s)$), which we denote by $C(s)$.

As frequently used in the research literature we use the notation object.op to denote the object after applying operation op. For example, $T.a(x)$ denotes the tree T after applying an adaptation operation at node x, and $ET(e(x.a(y)))$ to denote the expected time for performing full expansion at node x after an adaptation step was performed at node y. We also use notation $T.r_j(x)$ to denote the search tree after replacing the node x distribution by a single value of its j-th spike (x_j, p_j).

In our model for adaptation operation, we assume that applying it at a node decreases the expected time to explore "similar" nodes, where the effect is monotonic in the similarity. Let $dist(y, x)$ be a distance metric between nodes y and x. A typical such measure is a distance between feature vectors of the nodes. The similarity between nodes is a reciprocal of the distance. We thus allow the adaptation step to affect the expected time of future operations, but assume for simplicity that the effect of adaptation operations do not accumulate, i.e. that only the *nearest* (w.r.t. $dist$) adaptation operation can affect the expected time to expand a node, $ET(e(x))$. We also assume a model for expected expansion time after applying an adaption operation, $ET(e(x.a(y)))$, discussed in the next section. The expected time gain for a node is defined as:

$$ETG(y, x) = ET(e(x)) - ET(e(x.a(y))) \tag{4}$$

Of special interest is the expected time gain for all the leaves:

$$ETG_L(y) = \sum_{x \in L} ETG(y, x) \tag{5}$$

We denote by V the value of information for various items of knowledge. In particular, for a set of nodes A we denote by $V(A)$ the value knowing the exact values of all nodes in A. Also, $V(A.r_j(x))$ is the gain in value obtained by knowing the exact values for set A after we have already obtained the knowledge that node x has value x_j. Denoting $L = Leaves(T)$, we have $V(L) = Q^1$ as defined in previous section. The term $V(L.r_j(x))$ denotes the value of information for finding out the exact values for all leaves (hence for the entire tree) given that we have already obtained the exact value x_j for node x.

Let us denote by $V_Q(x)$ the value of information as used in BPIP DFISA, that is:

$$V_Q(x) = QSS(x) = A_L(x) \tag{6}$$

In order to allow for the effect of adaptation, we separate the value of information V into two components. The first, V_I is the *immediate* value (assuming that no further reasoning steps are performed), that is,

$$V_I(x) = \sum_{(x_j, p_j) \in DD(x)} p_j V(L.r_j(x)) - V(L) \tag{7}$$

where $DD(x)$ is the desirability distribution of the position in node x. The second element, V_F, is the future value of information, which we define as $V_F(x) = V_Q(x) - V_I(x)$. Using a discounting factor γ, we define the full utility of knowing the exact value of x as $V(x) = V_I(x) + \gamma V_F(x)$.

The set of worthwhile candidates for exploration is called the *potential set* P. The potential set is the set of leaves x such that $V(e(x)) > C(e(x))$. Note that the potential set may depend on operations performed, for example $P(T.e(x)) = P - \{x\}$. That is, in the tree resulting after x is explored (and gets its final value, as per our assumption), there is no further value in exploring x (or its descendants), and x is thus not in the potential set. However, for an adaptation operation, we assume that:

$$P(T.a(y)) = \{x | x \in L \wedge (V(e(x)) > C(e(x.a(y))))\} \quad (8)$$

We also define the value of a potential set, which is the value of exploring all nodes in the set: $V_P = \sum_{x \in P} V(e(x))$. The value of a potential after a computation s is given by:

$$V_P(T.s()) = V_F(s) + \sum_{x \in P(T.s())} V(x) \quad (9)$$

Note that the above value is not easy to compute, since the operation s will affect the costs and the potential set. Likewise, we define the cost for expanding all nodes in the potential node set:

$$C_P = \sum_{x \in P} C(e(x)) \quad (10)$$

and also need to find the cost of the potential node set after an operation s:

$$C_P(T.s()) = \sum_{x \in P(T.s())} C(e(x.s())) \quad (11)$$

We know the model of potential changes in expected time expressed in terms of ETG, the expected time gain. $x.s()$ reflects the fact that the operation s can affect ET through features of some nodes (such as distance to adaptation point).

Finally, we can define the utility of operations as the value of information gained, deducting the time costs. Thus, the utility of a expanding a potential set is: $U_P = V_P - C_P$. We also separate the utility into immediate and future utility: $U_I(s) = V_I(s) - C(s)$ and $U_F(s) = U_P(T.s()) - U_P$. The total utility of operation s is the weighted sum of the utilities, $U(s) = U_I(s) + \gamma U_F(s)$, where γ is a discounting factor. The utility of the operations we need are approximated as:

$$\begin{aligned} U(e(x)) &= U_I(e(x)) + \gamma U_F(e(x)) \\ &= V_I(x) - C(x) + \gamma(U_P(T.e(x)) - U_P) \\ &= V_I(x) - C(x) + \gamma(V_F(x) - V(x) + C(x)) \\ &= (1-\gamma)(V(x) - C(x)) \end{aligned} \quad (12)$$

$$\begin{aligned} U(a(y)) &= U_I(a(y)) + \gamma U_F(a(y)) \\ &= -C(a(y)) + \gamma(U_P(T.a(y)) - U_P) \\ &\approx \gamma TC(ETG_L(y)) - TC(ET(a(y))) \end{aligned} \quad (13)$$

In developing our equations, we have used the following assumptions:

Input: search tree T initialized with a root and its children.
Output: explored search tree T.

```
ExploreSearchTree(T)
    repeat forever
        S ← {All possible operations on T};
        foreach s ∈ S
            Evaluate R(s);
        op-best ← argmax{R(s)|s ∈ S};
        if (R(op-best) ≤ 0)
            return T;
        Apply op-best to T;
```

Figure 1: Search Algorithm Outline

1. Linear independence: the value of a set equals the sum of values of its nodes.

2. Large potential set $P \approx L$.

3.3 Outline of the algorithm

The original ordering over steps was based on QSS, which assumed uniform TC, and cannot be applied in the non-uniform case. We use $R(s) = \frac{U(s)}{ET(s)}$ instead, which takes into account the whole computational effort needed to choose and perform the next operation. Our stop condition is: no further expansions are done if the relevance R of the most relevant operation becomes negative. See Figure 1 for an outline of our meta-reasoning algorithm.

4 Implementation Details

The general evaluation scheme discussed in the previous section has several unspecified terms. In addition to the time cost function, we need to specify the form of the distributions in the nodes, and various other parameters. In this chapter, we discuss some of these details, and the assumptions behind them, as implemented in our system.

The implemented system applies our scheme to the domain of chess. Nevertheless, most of the system is domain-independent and can be extended to other zero-sum games (with a little additional effort needed to specify appropriate domain definitions).

For convenience, all time values in our system are expressed in uniform metric of expansion time units (ETU), where $1ETU$ is equal to time consumed by one basic leaf expansion. Although in real systems this value can depend on many factors (such as depth of the expanded node), we assume for this purpose the existence of such a uniform measure, which can be pre-evaluated by averaging over a large set of such cases.

To provide an abstract model of our domain we defined a set of easily-computable binary features S, such as a location

of a piece in a given square or a control of a piece over a given square (see example in Table 1).

Table 1: Example of features

ID	Description
X0025	Moving Side has a pawn in 'b5'
X1775	Waiting Side king has a control over 'g7'

We denote by S_p the space of board positions and by S_f - the Abstract Feature Space. Stochastic operators from a set Tdescribe the expected transition in a state corresponding to applying a move and receiving its adversarial response. The main reason for using such a stochastic model is our attempt to capture the inevitable uncertainty involved in a possible response of the opposite side. We describe these operators in a compact form of Bayes nets.

Over the feature set S, we defined a linear basis function set B, where every function is represented in the form of a short conjunction of features (or their negations). We include in this set all the preconditions and some common contexts of the operators contained in T.

Finally, a reward function is defined, which maps feature vectors to real numbers according to commonly used in chess a static material evaluator, which gives to each piece on the board its equivalence in pawn units (Table 2).

Table 2: Static piece values (in pawn units)

Piece	Value
Queen	9
Rook	4.5
Bishop	3
Knight	3
Pawn	1

Our value function (VF) $V : S_f \rightarrow R$ is based on the linear basis B. It gives a scalar value to a given state, indicating the desirability of the position corresponding to this state. For example our initial position described in the next section has static value 0.6 (due to more forwarded pawns).

The implemented system involves 4 main modules: Detector, Evaluator, Adapter and Tree-Searcher. The domain specific Detector is used to evaluate features for a given position. The Evaluator supplies the value of a given state by applying the VF. In addition to a scalar value $M(x)$ our Evaluator also supplies upper and lower bounds on given state value ($Up(x)$ and $Lw(x)$, respectively). From these values we construct a triple-spike formed distribution:

$$\begin{aligned} P(x) &= \{(\frac{(1-\alpha)(Up(x)-M(x))}{Up(x)-Lw(x)}, Lw(x)), \\ &\quad (\alpha, M(x)), \\ &\quad (\frac{(1-\alpha)(M(x)-Lw(x))}{Up(x)-Lw(x)}, Up(x))\} \end{aligned} \quad (14)$$

where α is determines the weight of the central component of the distribution (higher if we believe that the evaluator can find the correct value with high probability).

Another linear function, which maps the given state to appropriate original expansion time (OET), is also supplied by the Evaluator. $OET(x)$ is the expected time required to explore node x for its exact value using the original VF (before any adaptation). For $ET(a(x))$ we used a constant value equal to 10 time units. Approximation of expected time for an expansion after adaptation uses:

$$\begin{aligned} ET(e(x.a(y))) &= min(ET(e(x)), \\ &\quad (1 - ARF(y,x))OET(x)) \end{aligned} \quad (15)$$

where $ARF(y,x)$ is a part of time reduced due to adaptation of VF in node y (adaptation reduce factor). We expect from this value to be proportional to similarity of two involved positions, $sim(y,x)$.

$$ARF(y,x)) = sim(y,x)SRF(x) \quad (16)$$

where $SRF(x)$ is a part of time reduced due to adaptation of VF in node x (self reduce factor). This parameter may also depend on the involved position, but we just used a constant value 0.5, as a rough approximation.

All mappings implemented by the Evaluator are represented in a form of weighted sum over functions in set B. These weights can be adapted using learning, but initially this was done manually.

The Adapter receives the initial state (corresponding to a starting position) and an initial VF. It uses our abstract transition model to adapt the VF to the initial state by iterating stochastic trials and updating the current state value using look-ahead-1 q-value maximization. Figure 2 outlines our adaptation algorithm.

The Tree-Searcher contains the current tree configuration and is used to choose the most relevant node, to expand the chosen leaf node by its immediate descendants, and to propagate distributions and other useful information along the tree.

5 Preliminary Results

We performed an experiment using a pawn ending position, shown in Figure 3, with white to move and win. The correct solution is: 1.b6 axb6 2.c6 bxc6 3.a6 and then a6-a7-a8Q or 1... cxb6 2.a6 bxa6 3.c6 and then c6-c7-c8Q.

Starting from this position, Negamax requires nine-ply search to "see" the promotion to queen and performs about

Input: initial Value Function V
initial state S, number of trials n
Output: adapted Value Function V.
AdaptVF(V, S, n)
policy ← GeneratePolicy(V);
repeat n times {
$S_c \leftarrow S$; step ← 0;
repeat
step ← step + 1;
policy.LearnState(S_c);
op ← $policy.ChooseNextOp(S_c)$;
Apply op to S_c;
until (op = STOP_OP) or
(step = MAX_ADAPT_STEPS)
}
return V;

Figure 2: Adaptation Algorithm Outline

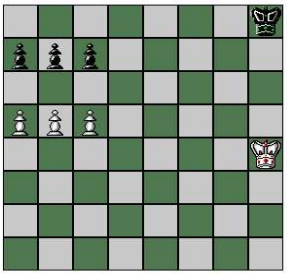

Figure 3: Ending position used in experiment

400.000 leaf expansions to obtain the correct solution using alpha-beta pruning. Obviously, this position is rather hard for standard search methods.

We implemented relevance based search, as follows: the QSS relevance, based on BPIP DFISA, TRQSS (our method, using $R(s)$, but without using the Adapter), and ATRQSS (our method including the use of the Adapter). The results are shown in table 3, the Exp column contains the actual number of expanded nodes, TU being the number of consumed time units. ATRQSS performed one adaptation operation at the root node during the tree exploration, in addition to the node expansions.

Table 3: Results for ending chess position

Method	Exp	TU
QSS	89	89
TRQSS	66	66
ATRQSS	51	61

The results show the advantage of taking into account the variable time needed for exploration, as well as the added value for planning within the scope of game-tree search.

We have also experimented on some other ending positions, and the results were mixed, as follows. Although QSS does improve the search, and TRQSS usually improves upon QSS, adaptation only very rarely kicks in, and in some cases even expends resources to no advantage. While that is to be expected, the result depend heavily on the quality of the initial VF, discounting factors and other parameters, which were introduced in an ad-hoc manner. In principle, these parameters should be learned and adapted by the system, an issue that is well beyond the scope of this paper.

6 Conclusion

Aiming at taking advantage of both planning and standard game-tree search, this paper is an initial attempt to merge these paradigms by using decision-theoretic meta-reasoning. Approximate value of information was used to decide which of the types of computation operator to apply, and where. Towards that end, we adapted and generalized the Best Play for Imperfect Player (BPIP) search control model of [1] to allow for planning steps, as well as game-tree search steps.

A rudimentary system employing these ideas within a chess playing program was implemented. Preliminary empirical results show that the scheme is promising, and improves upon both standard game-tree search and its relevance-based control versions, such as QSS within the BPIP model. However, this research is non-trivial and there is a long way to go in order to achieve the full goals mentioned in the introduction. Topics remaining for future research range from adding planning modules and improving the meta-reasoning model of planning operations, to making the system more robust by a more disciplined selection of the many parameters. The issue of learning distributions over execution times of operators, and of error distributions is also an important desirable feature that at present is unexplored.

References

[1] E. Baum, W. Smith. Best Play for Imperfect Players and game tree search. Part I, Theory. Technical report NEC Research Institute, Princeton, NJ, 1995.

[2] E. Baum, W. Smith. A Bayesian Approach to Relevance in Game Playing. NEC Research Institute, Princeton, NJ, 1997.

[3] S. Russell, E. Wefald. On optimal game-tree search using rational meta-reasoning. In [IJCAI-89] pages 334-340.

[4] S. Russell. Fine-grained decision-theoretic search control. In [UAI-90] pages 436-442.

[5] J. Schaeffer. Long-range planning in computer chess. ACM Annual Conference, pages 170-179, 1983.