Algorithm Selection in Optimization and Application to Angry Birds

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

Solomon Eyal Shimony
CS Department
Ben Gurion U., UMass Lowell
shimony@cs.bgu.ac.il

Avinoam Yehezkel
CS Department
Ben Gurion University
yehezavi@post.bgu.ac.il

Abstract

Consider the MaxScore algorithm selection problem: given some optimization problem instances, a set of algorithms that solve them, and a time limit, what is the optimal policy for selecting (algorithm, instance) runs so as to maximize the sum of solution qualities for all problem instances?

We analyze the computational complexity of restrictions of MaxScore (NP-hard), and provide a dynamic programming approximation algorithm. This algorithm, as well as new greedy algorithms, are evaluated empirically on data from agent runs on Angry Birds problem instances. Results show a significant improvement over a meta-agent greedy scheme from related work. Hopefully this improvement will carry over to an Angry Birds meta-agent in the actual competition.[1]

1 Introduction

Algorithm selection is of significant interest to researchers in AI, and other fields where more than one algorithm is available to solve problems under computational resource constraints [Rice, 1976; Huberman *et al.*, 1997; Xu *et al.*, 2008]. This paper examines a variant of algorithm selection ("MaxScore") where one needs to solve a *set* of optimization problems, with computational resource (assumed here to be time) limitation being over the entire set.

Our motivation for the MaxScore setting is combining multiple programs that compete in the AI Angry Birds (Copyright Rovio Entertainment) competition, on which we also base the empirical results of this paper. In the AIbirds competition, each agent program (or human) is presented with N previously unseen game levels (problem instances). The agents can select a level to play, where at each level the agent is presented with a screen-shot representing a physical simulation. The agent may play any level as many times as desired, until its overall allocated time (typically 10 or 30 minutes) expires. Level score is the maximum achieved in all attempts, with total score being the sum of level scores (typically 4 or 8 levels in past competitions).

[1]Supported by: ISF grant 417/13, BGU Frankel Center. Meta-agent implemented by: Lior Schachter, Dor Bareket, Ori Zviran.

Of the numerous AI techniques used in AI birds agent programs, to-date none have achieved near-human performance; each program has strengths and weaknesses in different areas of the game. Rather than attempting to create a better AI for Angry Birds, our goal here is one of meta-reasoning: use a portfolio of existing programs to play better, an idea suggested originally by [Stephenson and Renz, 2017] with promising initial results. The focus of this paper is on how to do this combination optimally given the available information, in a decision-theoretic sense, rather than on the aspect of learning to fit the program to the problem instance.

Informally (see Section 2 for the formal definition) we are given a set of problem instances (levels), to each of which we can apply any of a set of given algorithms (agents). Each such application uses up an unknown amount of time, and results in a score for the level that can be observed after each algorithm run terminates. Our meta-reasoning problem is to find a policy for selecting which agent to apply to which level, at any point in time, such that the sum of scores for all the levels at timeout is maximized. Note that in order to make this policy optimization well defined, one must specify some prior knowledge about scores and runtimes. In this paper we assume that these are specified by random variables with known distribution models.

2 Formal Problem Statement

A MaxScore problem is a 4-tuple (I, A, T, P) where I is a set of problem instances to be optimized (game levels in Angry Birds), A is a set of algorithms (or agent programs), T is a time limit, and P is a known distribution model over problems in I, agents in A, that describes the (non-negative) score $S(a, l, i)$ achieved by agent $a \in A$ and the (positive valued) runtime $T(a, l, i)$ of a when applied to problem instance $l \in I$ during decision-making round i of the problem-solving task (or game play). Distribution P (also known as a *performance profile* [Zilberstein and Russell, 1996]) can be defined in various ways, we consider cases where P consists of deterministic functions, or of distributions with independent variables, or distribution with dependent variables (some of which are unobservable).

A policy π is a mapping from process histories to actions. The process is to select, at each round $i \geq 1$, an agent program $a(i)$ to apply to a problem instance (level) $l(i)$, given the past observations, according to π. The results $S(a(i), l(i), i)$

and $T(a(i), l(i), i)$ are observed after the selection at round i. Then i is incremented as we go to the next round. The process stops when we reach the time limit, i.e. at the first k such that:

$$\sum_{i=1}^{k} T(a(i), l(i), i) > T$$

The score of the process is the sum of maximal achieved scores for each problem instance, i.e.

$$S = \sum_{l \in I} \max_{i=1}^{k-1} S(a(i), l(i), i)\delta(l, l(i))$$

where $\delta(i, j)$ is the Kronecker delta. The problem is: find a policy (mapping from process history, or alternately belief state and round number, to (agent, problem instance) pairs) that maximizes the expected value of S. This is the *sequential* (*adaptive*) decision making version of the problem.

We also consider, for computational complexity analysis, simpler *linear* settings of MaxScore, where the decision on which agents to run on which problem instances (and in which order) is decided only once, in advance. A policy in this setting is simply a sequence of (agent, problem instance) pairs. The linear setting is the same as the *batch* setting (also called *non-adaptive*) commonly used in the research literature [Shperberg and Shimony, 2017], except that if the runtimes are not both known and deterministic, one must specify the ordering so as to have a well-defined policy (some of the agents may not get to run at all, due to runtimes uncertainty).

2.1 Performance Models

Assuming that the distribution model is Markov, the MaxScore problem is a POMDP with states defined by the current maximum scores vector cR and the play time elapsed. As the number of rounds is not known in advance, we define the problem as an indefinite horizon POMDP with terminal states being those where the sum of runtimes exceeds T. The transition probabilities in this POMDP are trivially (and deterministically) defined given the score and runtime achieved in the current round (which in turn are defined by distribution P). The reward function is 0 except for transitions into terminal states. In general, POMDPs are intractable (PSPACE complete even if the belief space is finite). The actual complexity of MaxScore depends on the setting (sequential vs. linear/batch), on the performance profiles distribution model P, and on the size of sets I and A.

A major issue is the performance profile (distribution model) P. Typically, exactly what scores and runtimes to expect is unknown, except by running the programs on the problem instances, which is too late to make the needed decision. However, we can run the programs on similar instances, collect statistics and learn a prediction model given instance features. Related work involved learning to predict the expected score of an agent in an unseen level [Stephenson and Renz, 2017]. However, as argued below, such information is insufficient for optimal choice: one may need to predict the whole score distribution (or equivalently, the expected improvement over each possible current score).

If the agent programs are effectively memoryless, i.e. attempt to solve the level from scratch each time they encounter it, then the order of the observed scores and runtimes is irrelevant. This behavior is reasonable for Angry Birds, as the game is effectively stochastic. Additionally (unlike search problems in most search domains), even if an agent knows the optimal play, it must still waste the time to execute it, which takes essentially as much time as for an agent that needs to plan the play. We thus consider the agent scores and runtimes for a problem instance as i.i.d samples drawn from the distributions $P_S(a, l)$ and $P_T(a, l)$, respectively. If these distributions are known, the MaxScore problem is in fact an MDP, analyzed below.

3 Analysis: Known IID Case

We examine the computational complexity of some settings of the MaxScore problem. We begin with the fully deterministic case (scores and runtimes known in advance), and proceed to the independent case.

3.1 Complexity: Restricted Versions

We show that the MaxScore problem is NP hard even in the following extremely restricted cases:

1. Independent score distributions, deterministic runtimes, and only a single problem instance ($|I| = 1$).
2. Deterministic scores and runtimes, with $|A| = 1$ (but with $|I|$ unbounded).

We begin with the latter instance, as proving NP-hardness here is immediate through a straightforward mapping from the Knapsack problem. Simply map Knapsack item values to scores, item weights to runtimes, and the weight limit to T. Note that as the scores and runtimes are deterministic, in this case there is no difference between a linear setting, a batch setting, and a sequential setting of the problem.

With only one problem instance, we need to be more careful, but still get (see appendix for proof):

Theorem 1. *The linear setting of the MaxScore problem with independent score distributions, deterministic runtimes, and $|I| = 1$, is NP-hard.*

We believe that the complexity of *sequential* setting with the same restrictions is at least as hard as the linear setting, but have not proved it. Also note that the *linear* setting MaxScore problem with $|I| = 1$ is non-trivial even if we further restrict it to *unit runtimes*. For example, using a natural greedy scheme that picks the agent with the best expected score can be suboptimal. Consider having agents A, B, C, with time limit T=2. Suppose A always scores 100, B scores 101 with probability 0.99 and 0 otherwise, and C scores 200 with probability 0.001, and 99 with probability 0.999. A greedy scheme would first pick A, as it has the certain value 100, higher than the expected scores of B and C. In fact the optimal policy is to pick B and C (expected score just over 101, whereas anything containing A achieves less than 101). The computational complexity of this setting of MaxScore is, as far as we know, an open problem.

3.2 Approximation Algorithms

Using dynamic programming schemes it is possible to achieve a pseudo polynomial algorithm for the case of

$$OPT(rT, cR) = \max_{\substack{a \in A \\ l \in I}} \left(\sum_{\substack{r \in \\ sp(P_S(a,l))}} \sum_{\substack{t \in \\ sp(P_T(a,l)) \\ \wedge t \leq rT}} OPT(rT - t, R') \times P_T(a,l)[t] \times P_S(a,l)[r] + \sum_{\substack{t \in \\ sp(P_T(a,l)) \\ \wedge t > rT}} \sum_{1}^{m} cR_i \times P_T(a,l)[t] \right)$$

Figure 1: Optimal Solution to the MaxScore problem

(known distribution) independent scores and runtimes, and $|I|$ bounded by a constant, using the following scheme. (The assumption that $|I|$ is bounded by a constant is reasonable for e.g. AIbirds competitions, where $|I|$ is 4 or 8.) Additionally, we are assuming that score items (e.g. score for killing a pig in Angry Birds) and runtimes are integer valued, and that the time span and score items have a unary representation in the input. The following dynamic programming value determination scheme (a variant of the Bellman equation) computes the optimal policy, and has a time complexity linear in the time span and the maximum score, and exponential in $|I|$.

Let $OPT(rT, cR)$ be the optimal solution value to the MaxScore problem with rT remaining time, and current maximum score vector $cR = \langle cR_1, \ldots, cR_m \rangle$. The value determination recursive equation for $OPT(rT, cR)$ appears in Figure 1, where $R' = \langle cR_1, \ldots, \max\{cR_l, r\}, \ldots, cR_m \rangle$, and $sp(D)$ is the support of distribution D. The value of $OPT(T, \langle 0, \ldots, 0 \rangle)$ is that of the optimal policy at the initial state.

If the score distributions are continuous, or have too many values, we can round them into bins, achieving a $(1 - \varepsilon)$-approximation to the optimal policy. Likewise discretizing the runtime distributions is possible, but here near-optimality is not guaranteed. Although the dynamic programming approximation scheme can be computed in pseudo-polynomial time, it is still too computationally demanding to be practical. We would thus like to use a greedy scheme in practice, and the one that comes to mind immediately is to use the agent that has the best expected score, as essentially done in [Stephenson and Renz, 2017]. A slightly better scheme is to take into account the runtime, and use the ratio of expected score over expected runtime. However, it is easy to show that these schemes are far from optimal (as verified by the empirical results).

For example, suppose we have only one problem instance and two agents. We have already achieved a score of 10000, and have time for exactly one more run. The first agent always scores 10000. The second agent scores 100000 with probability 0.05, and otherwise fails and scores 0, thus its expected score is 5000. The above greedy schemes would select the first agent and always get 10000, while the optimal policy would obviously select the second agent, to possibly end up with 100000 (with expected final score 10500). An improved greedy scheme instead looks at the *expected improvement* to the score over the current score, i.e. the value:

$$E[S(a,l,i) - \max_{a \in A} \max_{j=1}^{i-1} S(a,l,j)] \tag{1}$$

rather than just the expected score $E[S(a,l,i)]$. (Consider $S(a,l,j)$ to be 0 if agent a was not selected in round j to be run on problem instance l.) Although the improved greedy scheme is suboptimal even for unit runtimes, in practice it does well (Section 4).

4 Experiments: Known IID

Quality/runtime tradeoffs for the independent model were examined for score and runtime distributions based on runs of AIbirds algorithms on original Angry Birds game levels. Each algorithm was run 10 times on each level to obtain the empirical distributions, which were then treated as if they were the true distributions. Levels that caused issues with the agent's vision module were filtered out.

We applied the meta-agent to the following four existing agents: AngryBER, ihsev, Eagle's Wing, planA, which competed in past AIBirds competitions. All tests were conducted on Windows 10 using a machine with Intel(R) Core(TM) i7-4700HQ 2.40GHz processor and 12 GB RAM. The evaluation process was performed using different numbers of levels and time budgets as described below:

```
for 1 to 50 do
    for Every number of levels and time budget T configuration in {2,3,4} × {200,400,600,800,1000} do
        Draw random levels uniformly from the level pool.
        for 1 to 10000 do
            while Time budget was not exceeded do
                Compute policy and choose a move using collected statistics as true distribution.
                Execute the selected move (agent and level) by drawing score and time according to the statistics.
```

We also evaluated some of the algorithms on the settings used in the AIBirds competition, solving 8 levels with $T = 1800$ seconds.

The following optimization algorithms where compared:

1. **Dynamic Programing (optimal)**: compute the recurrence relation in Figure 1, using memoization of the results from all recursive calls.
2. **Binned Dynamic Programing(X,Y)**: same as the optimal solution, with scores rounded up to the next multiple of X , and times rounded to the next multiple of Y. We used this algorithmic scheme with $X \in \{1, 1000, 10000\}$ and $Y \in \{1, 10, 25\}$.
3. **Score Greedy**: choose the agent and level that maximize the expected score.
4. **Rate Greedy**: choose the agent and level that maximize the expected score divided by the expected time given that the time is less than or equal to the remaining time, multiplied by the probability that the time is less or equal to the remaining time.

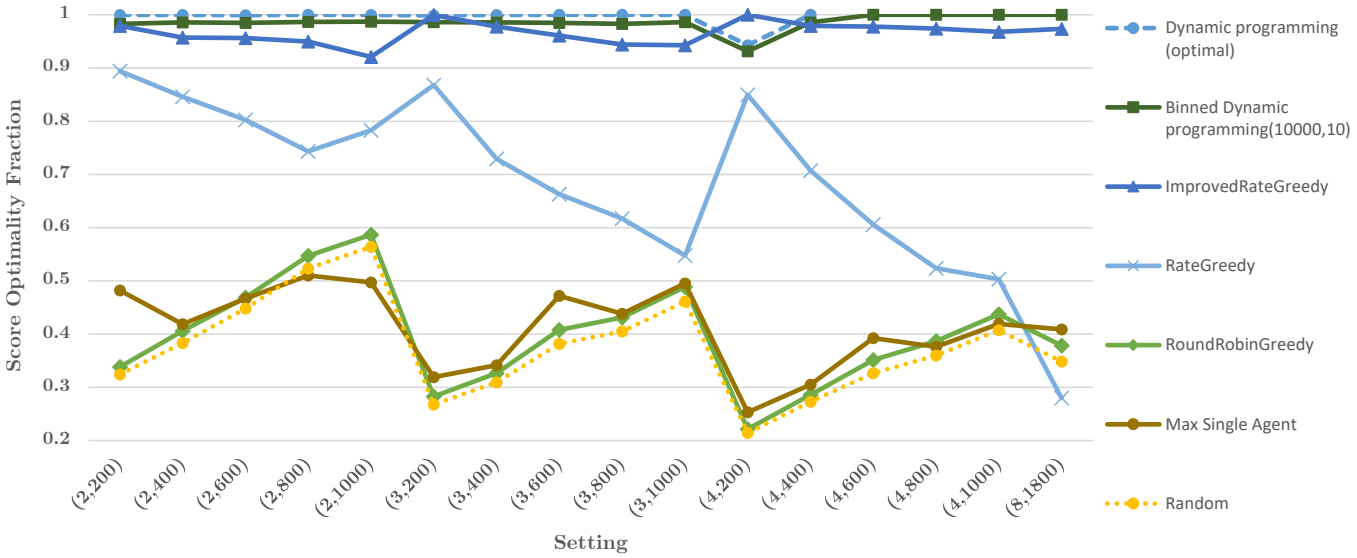

Figure 2: Optimization Algorithms Score Evaluation

5. **Improved Score/Rate Greedy**: same as the score/rate greedy except for considering expected score/rate improvement instead of expected score.
6. **Round Robin Score Greedy**: the algorithm from [Stephenson and Renz, 2017], which selects a level using round-robin, and chooses the agent that maximizes the expected score for that level, preferring agents not selected in previous attempts.
7. **Play Single Agent**(A): select a level using round-robin, always with agent A. This scheme was evaluated for each of the 4 possible agents.
8. **Random**: draw a combination of level and agent uniformly.

The results are shown in Figure 2. The scores in the plot are normalized to the highest score for each setting. For clarity, we show only a subset of the algorithms. For the Play Single Agent policies, we show only the maximum value among them. The score greedy and improved score greedy were dominated by the rate greedy and improved rate greedy respectively, thus the former two are not shown. Finally, we showed the Binned Dynamic Programing(10000,10) as a sole representative of its category, since it achieved the best balance between runtime and score. The optimal policy did not always result in the best score, as the process is stochastic and thus exhibits measurement noise. The results indicate that the optimal policy is indeed the best in terms of scores. However, the binned version and the improved rate greedy achieved near-optimal results. When considering runtime and space usage, the optimal solution could not solve instances greater than 4 levels with $T = 400$ time budget. The binned version was able to solve all instances, with a maximal overhead of 13.7 seconds and a maximal memory usage of 52 MB across all instances with 4 levels or less. However, it required an average of 335 seconds and 270 MB of memory to handle the full competition setting (8 levels with 1800 seconds time budget). All the other algorithms ran in negligible time (several milliseconds) and memory. This makes the improved rate greedy the best algorithm in terms of balance between score and resources. Note that in the experiments we did not include the optimization runtime in the total available runtime T; but in a competition it must be. Also, the simulation runtime of the agents averaged roughly 90 seconds, so 1800 seconds consists of about 20 rounds (agent runs).

5 Unknown Distributions

A major point of competitions like AIbirds (as well as other competitions, such as in planning, SAT-solving, etc.) is that they are done with *previously unseen* problem instances, so the score and runtime distributions are unknown. The latter issue then becomes a learning problem, which can be modeled by treating the agent performance quality as hidden random variables, with some assumed prior distributions based on observing similar problem instances. We adopt a naive learning scheme (described briefly below for independent scores and runtimes for simplicity, as implemented for the AIbirds meta-agent).

5.1 Unknown IID Score and Runtime

In our naive learning scheme, we are assuming that agent performance profiles of a previously unseen problem instance are (almost) equal to their performance profile on some problem instance(s) for which performance statistics were already collected. Hence, we are essentially taking a case-based reasoning (CBR) approach to predicting the performance profile for an unknown instance. However, we do not assume knowledge of *which* of the previously seen performance statistics fits the current instance.

Therefore, if we need to predict an agent performance $Q(a, l, i)$ for round i of the current problem instance l (which we model as the score and runtime iid $P_S(a, l)$ and $P_T(a, l)$) it makes sense to condition on the (unknown) agent performance profile $Q(a, l)$ for the current problem instance. Es-

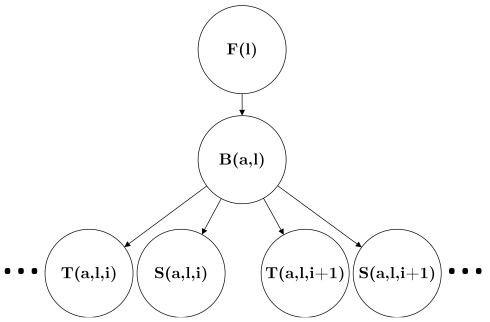

Figure 3: Dependency model (Bayes net fragment)

sentially, what we need is a mapping from features to agent performance (i.e. *distributions* over score and runtime distributions), as we do in this paper.

The mapping we adopt here is simply a smoothed version of the performance profiles of the K most similar instances. (Smoothing is done in this paper by assigning scores and runtimes into bins.) That is, for every problem instance l in the training set, and every agent program a, collect and store the score and runtime distributions estimate as $Q(a, l)$ indexed by the feature values (vector $F(l)$) for problem instance l.

When a new problem instance l is encountered online, compute its feature vector values $F(l)$, and find the K most similar instances $l_1, ..., l_K$ according to some appropriately defined similarity measure $s(l_i, l)$. Now we assume that the agent performance for instance l has a distribution over performance profiles, and that it is equal to the (smoothed version) of its performance profile for some instance l_i, with probability proportional to $s(l_i, l)$. That is, denote by $B(a, l)$ a K-valued random variable, with integer values denoting the respective $Q(a, l_i)$ profile. Then we have: $P(B(a, l) = i) = \frac{s(l_i,l)}{\sum_{j=1}^{K} s(l_j,l)}$.

The performance profiles describe both score and runtime distributions, and additionally we assumed that these are drawn i.i.d. given the value of $B(a, l)$. The distribution model topology is summarized in Figure 3, for each problem instance l (shown for one agent program). We have an observable feature vector variable $F(l)$. Belief updating for this conditionally i.i.d. model is straightforward, as this is a naive Bayes model.

Obtaining a reasonable $s(l, l')$ is a learning problem, which was done here naively by normalizing all feature values to $[0, 1]$, and taking the inverse of the Euclidean distance as the similarity. The result is a low-quality performance predictor.

Improving the prediction quality is a learning problem beyond the scope of this paper. In order to methodically examine robustness of the meta-reasoning to the prediction error, we introduce a prediction cheating factor p_l. For each problem instance l, use the naive learning scheme defined above, but modify the distribution of $B(a, l)$ to allow the true (possibly binned) $Q(a, l)$ as an additional possible performance profile (numbered 0). That is,

$$P(B(a, l) = i) = \frac{s(l_i, l)}{\sum_{j=1}^{K} s(l_j, l)} \cdot (1 - p_l) \qquad (2)$$

for $1 \leq i \leq K$ and $P(B(a, l) = 0) = p_l$. The cheating factor p_l can be varied between 1 (that is, knowing the correct distribution as in Section 4, and 0 (pure low-quality learned model as above).

5.2 Experiments: Unknown IID

We represented each level's score normalized by $maxScore_l$, an upper bound on achievable score in each level, that can be computed using the features. We used the following subset of features, described in [Stephenson and Renz, 2017; Tziortziotis *et al.*, 2016]: #Blocks, targetWidth, targetHeight, closestObjDist, farthestObjDist, density, #Objects, #iceObjects, #woodObjects, #stoneObjects, #Pigs,#helmetPigs, #noHelmetPigs, #Birds, #RedBirds, #YellowBirds, #BlueBirds, #BlackBirds, and #WhiteBirds.

We tested the optimization algorithms using the same process as in Section 4, where the algorithms had to rely on the predicted distribution based on the naive learning scheme, with a "cheating factor" $p_l \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. For each p_l value, we incorporated the resulting distribution in the following algorithms: (1) the improved rate greedy defined in Section 4, using the distribution of distributions without performing belief updating (denoted as IRG); (2) a binned version of the improved rate greedy with Bayesian belief updating, using a zero value bin and 10 additional bins uniform in $(i \cdot maxScore_l, i + 0.1 \cdot maxScore_l], 0 \leq i \leq 9$ (denoted as BIRG). We used $K = 30$ in the learning process for both algorithms. We also tested algorithm (1) with other K values using $p_l = 0$.

The results are shown in Table 1. Algorithm (1) using $p_l = 0, k = 30$ does not show any improvement over choosing agent and level at random and the round robin greedy algorithm (denoted as RRG). Choosing $k = 1$ improved the average performance, but still achieves results of moderate quality. Nonetheless, even a slight improvement in the distribution prediction ($p_l = 0.2$) improves the scores by a factor of 2, and an additional improvement ($p_l = 0.4$) provides a solution with 96% of an algorithm using the true distributions. The binned version of the algorithm is more robust to the low quality in the predicted distributions, and shows a more promising result given $p_l = 0$; and better than the max single agent. However, the coarse binning gives up accuracy and is thus less able to take advantage of a possible improvement of the distribution prediction quality.

6 Agent Implementation Issues

In the conditionally i.i.d. model, since $B(a, l)$ is unobservable, but its current distribution (thus belief state) changes given new observations of $T(a, l, i)$ and $S(a, l, i)$, we now have a POMDP that we cannot hope solve optimally, especially in real time. Instead, we can solve an MDP where the $T(a, l, i)$ and $S(a, l, i)$ are assumed to be i.i.d. as before, but based on the current belief state of $B(a, l)$.

That is, we can do the belief updating given the new observed scores and runtimes, but in the policy computation act as if future updates are not observed. Then one can recompute the MDP policy after each observation and belief update. However, the MDP solution was also quite computationally intensive, and re-computation makes it even more so.

Table 1: Optimality Fraction Results of the Learning Process with Different p_l Values

Algorithm	**Number Of Levels**															**Optimality Fraction (Average)**
	2					**3**					**4**					
	200	400	600	800	1000	200	400	600	800	1000	200	400	600	800	1000	
IRG - $p_l = 0$	0.38	0.47	0.47	0.48	0.58	0.28	0.43	0.39	0.49	0.40	0.22	0.19	0.30	0.26	0.37	0.37
IRG - $p_l = 0.2$	0.82	0.80	0.78	0.79	0.77	0.79	0.76	0.80	0.76	0.75	0.79	0.73	0.75	0.68	0.71	0.76
IRG - $p_l = 0.4$	0.98	0.94	0.97	0.95	0.94	0.98	0.96	0.96	0.95	0.94	0.96	0.97	0.94	0.95	0.94	0.95
IRG - $p_l = 0.6$	0.98	0.98	0.99	0.98	0.97	0.98	0.98	0.99	0.98	0.98	0.98	0.99	0.98	0.98	0.98	0.98
IRG - $p_l = 0.8$	0.98	0.99	0.99	0.99	0.99	0.98	0.99	0.99	1.00	0.99	0.98	0.99	0.99	0.99	0.99	0.99
IRG - $p_l = 1$(true)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
BIRG - $p_l = 0$	0.71	0.64	0.70	0.63	0.64	0.62	0.57	0.59	0.51	0.53	0.64	0.51	0.46	0.44	0.46	0.55
BIRG - $p_l = 0.2$	0.88	0.78	0.76	0.73	0.70	0.85	0.68	0.60	0.51	0.55	0.86	0.59	0.48	0.49	0.46	0.61
BIRG - $p_l = 0.4$	0.89	0.81	0.79	0.71	0.71	0.85	0.66	0.59	0.52	0.56	0.86	0.59	0.51	0.48	0.47	0.62
BIRG- $p_l = 0.6$	0.91	0.82	0.79	0.72	0.71	0.87	0.65	0.60	0.52	0.57	0.86	0.58	0.50	0.49	0.46	0.62
BIRG - $p_l = 0.8$	0.90	0.82	0.79	0.73	0.70	0.88	0.67	0.61	0.52	0.56	0.89	0.60	0.50	0.50	0.47	0.63
BIRG - $p_l = 1$(true)	0.92	0.82	0.79	0.72	0.71	0.91	0.68	0.61	0.53	0.56	0.89	0.61	0.51	0.49	0.47	0.63
IRG - $k = 1$	0.60	0.55	0.50	0.73	0.47	0.40	0.51	0.62	0.53	0.47	0.35	0.39	0.41	0.46	0.43	0.49
IRG - $k = 5$	0.40	0.63	0.43	0.62	0.52	0.27	0.39	0.52	0.45	0.43	0.23	0.31	0.28	0.28	0.37	0.40
IRG - $k = 10$	0.38	0.52	0.49	0.60	0.63	0.34	0.41	0.56	0.54	0.51	0.27	0.30	0.38	0.36	0.45	0.45
IRG - $k = 20$	0.29	0.49	0.48	0.51	0.59	0.26	0.39	0.41	0.46	0.44	0.20	0.23	0.33	0.28	0.37	0.38
Random	0.32	0.45	0.44	0.62	0.55	0.29	0.34	0.43	0.45	0.47	0.25	0.28	0.36	0.37	0.41	0.41
RRG - $p_l = 1$(true)	0.30	0.49	0.48	0.68	0.61	0.30	0.40	0.49	0.50	0.52	0.24	0.33	0.43	0.42	0.41	0.45

As the improved greedy scheme performed almost as well as the MDP solution w.r.t. optimality, we no longer considered using the MDP solution, for practical reasons.

Our meta-agent implementation starts by collecting information on all levels using the provided vision module. Based on this, the meta-agent constructs an objects-tree for each level, extracts features from the objects-trees, and predicts a performance profile for each level and agent pair. Then, the meta-agent applies the improved greedy scheme to select a pair of agent and a level to play. The meta-agent sends the selection to the server and observes the results of the run. The observations are used for belief updating. The select-and-play process repeats until the time limit is reached.

Experiments with the actual meta-agent take much longer than those based solely on the distributions, due to the need to average numerous runs and level combinations, compounded with the inherent 30 minutes runtime per run. In preliminary results, running the agent on past competition levels (between 2014 and 2016), a total of 72 levels (8 at a time), the improved-greedy based meta-agent achieved a total score of $3,974,150$, compared to PlanA $(3,280,570)$, ihsev $(2,891,520)$, AngryBER $(2,479,010)$ and Eagle's Wing $(2,907,890)$. Note that the above 4 agents were the ones actually used by the meta-agent, and all of them contributed to the total score of the meta-agent. The hyper-agent of [Stephenson and Renz, 2017] achieved an impressive total score of $3,822,660$; However, it uses 8 agents as oppose to our 4.

7 Discussion

In this paper we defined the MaxScore optimization problem, analyzed its computational complexity (NP-hard even under extreme restrictions), and suggested approximation algorithms for known independent distributions. In practice, based on empirical evaluation on AI birds, it turns out that a greedy algorithm based on expected improvement is near-optimal. Despite the latter having no theoretical guarantees, it currently seems to be the only viable alternative for real-time computation. Applying these results to unknown distributions requires learning performance profiles given problem instance features. A naive learning scheme applicable to the AIbirds application was proposed. This results in significant error in the predicted distributions, which degrades the meta-reasoning results. Nevertheless, the greedy algorithm is still the better option, especially if the distribution model is updated using scores and runtimes observed during the run.

The MaxScore problem is closely related to *algorithm selection*, as originally defined by Rice in 1976 [Rice, 1976]. *Algorithm portfolios* [Gomes and Selman, 2001; Huberman *et al.*, 1997] are a natural and popular extension of the idea of algorithm selection. Such techniques are based on minimizing risk in economics. This approach defines a collection of algorithms (a portfolio) and establish a resource allocation to the algorithms in the portfolio in order to solve a given problem instance (instead of choosing a single algorithm for a given problem instance). This field has been studied extensively in the last decades, including works on different computational settings (parallel, sequential or in-between), many applications with outstanding results [Xu *et al.*, 2008; Hoos *et al.*, 2014; Kadioglu *et al.*, 2010; Kadioglu *et al.*, 2011] and even meta-level techniques for choosing a selector [Lindauer *et al.*, 2015]. Most common settings of algorithm portfolios focus on finding a solution to a single given problem instance. Our setting generalizes the meta-level decision problem solved in algorithm portfolios to choosing which problem instance to work on, as well as selecting algorithms to use at any given time.

A significant part of the research on algorithm portfolios and multi-armed bandits focuses on learning issues. E.g. in [Kotthoff, 2016], the focus is on analyzing problem features and applying different varieties of machine learning techniques in order to find scheduling policies for the portfolios. In this paper we achieved good results despite using a rather naive learning scheme to obtain a mapping from features to score and runtime distributions.

References

[Garey and Johnson, 1979] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Gomes and Selman, 2001] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.

[Hoos *et al.*, 2014] Holger Hoos, Marius Thomas Lindauer, and Torsten Schaub. claspfolio 2: Advances in algorithm selection for answer set programming. *TPLP*, 14(4-5):569–585, 2014.

[Huberman *et al.*, 1997] Bernardo A. Huberman, Rajan M. Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[Kadioglu *et al.*, 2010] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. Isac –instance-specific algorithm configuration. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 751–756, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[Kadioglu *et al.*, 2011] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *CP (LNCS6876)*, pages 454–469, 01 2011.

[Kotthoff, 2016] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In Christian Bessiere, Luc De Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi, editors, *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*, pages 149–190. Springer, 2016.

[Lindauer *et al.*, 2015] Marius Thomas Lindauer, Holger H. Hoos, Frank Hutter, and Torsten Schaub. Autofolio: An automatically configured algorithm selector. *J. Artif. Intell. Res.*, 53:745–778, 2015.

[Rice, 1976] John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65 – 118, 1976.

[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.

[Stephenson and Renz, 2017] Matthew Stephenson and Jochen Renz. Creating a hyper-agent for solving angry birds levels. In Brian Magerko and Jonathan P. Rowe, editors, *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17), October 5-9, 2017, Snowbird, Little Cottonwood Canyon, Utah, USA.*, pages 234–240. AAAI Press, 2017.

[Tziortziotis *et al.*, 2016] Nikolaos Tziortziotis, Georgios Papagiannis, and Konstantinos Blekas. A bayesian ensemble regression framework on the angry birds game. *IEEE Trans. Comput. Intellig. and AI in Games*, 8(2):104–115, 2016.

[Xu *et al.*, 2008] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *J. Artif. Intell. Res.*, 32:565–606, 2008.

[Zilberstein and Russell, 1996] Shlomo Zilberstein and Stuart J. Russell. Optimal composition of real-time systems. *Artif. Intell.*, 82(1-2):181–213, 1996.

Appendix: Proof of Theorem

For convenience, Theorem 1 is re-stated here:

Theorem 1. *The linear setting of the MaxScore problem with independent score distributions, deterministic runtimes, and $|I| = 1$, is NP-hard.*

Proof: by reduction from the optimization version of knapsack ([Garey and Johnson, 1979], problem number [MP9]), re-stated below. Given a set of items $\mathcal{S} = \{s_1, ...s_n\}$, each with a positive integer weight w_i and a positive integer value v_i, a weight limit W, find a sub-multiset S of $\mathcal{S}$ with a maximal total value, subject to: total weight of S at most W.

In the reduction, each agent represents an item in the Knapsack problem, where $P_T(a_i, l) = [1 : w_i]$ and $P_S(a_i, l) = [\varepsilon : v_i, 1 - \varepsilon : 0]$. As this is a simple one-to-one mapping, we abuse the notation and treat the agents as if they are actually the respective elements from $\mathcal{S}$ in the Knapsack problem. In the MaxScore problem, let:

$$T = W, \;\; H = \max_{s_i \in \mathcal{S}} v_i, \;\; M = \frac{W}{\min_{s_i \in \mathcal{S}} w_i}, \;\; \varepsilon = \frac{1}{M^2 H + 1}$$

Suppose that S is a candidate solution to the MaxScore problem, where $m = |S| \leq n$. Assume w.l.o.g. that $S = \{s_1, s_2, ..., s_m\}$ and that the items are sorted in non-descending order of values v_i. Denote by $P(S)$ expected value from selecting the items in the sequence S as a policy. Then:

$$P(S) = \sum_{i=1}^{m} v_i \varepsilon (1 - \varepsilon)^{m-i} \leq \sum_{i=1}^{m} v_i \varepsilon$$

On the other hand, we have:

$$\begin{aligned} P(S) &= \sum_{i=1}^{m} v_i \varepsilon (1 - \varepsilon)^{m-i} \geq \sum_{i=1}^{m} v_i \varepsilon (1 - \varepsilon)^{m} \\ &\geq \sum_{i=1}^{m} v_i \varepsilon (1 - \varepsilon)^{M} \end{aligned}$$

From Bernoulli's inequality, we have:

$$(1 - \varepsilon)^M \geq 1 - M\varepsilon = 1 - \frac{M}{M^2 H + 1} > 1 - \frac{1}{MH}$$

Therefore:

$$\begin{aligned} P(S) &> \sum_{i=1}^{m} v_i \varepsilon (1 - \frac{1}{MH}) = \sum_{i=1}^{m} v_i \varepsilon - \frac{\sum_{i=1}^{m} v_i \varepsilon}{MH} \\ &\geq \sum_{i=1}^{m} v_i \varepsilon - \varepsilon = \varepsilon (\sum_{i=1}^{m} v_i - 1) \end{aligned}$$

Now let S be an *optimal* solution to the MaxScore problem. Since S satisfies the time constraint, we have $\sum_{i=1}^{m} w_i \leq T = W$, so S satisfies the weight constraint in the Knapsack problem and is thus a solution therein. Assume in contradiction that there exists a legal solution S' to Knapsack s.t. $\sum_{s_i \in S'} v_i > \sum_{s_i \in S} v_i$. Since the values of the items in knapsack are integers, we know that $\sum_{s_i \in S'} v_i \geq (\sum_{s_i \in S} v_i) + 1$. Thus, as $|S'| \leq \frac{1}{\varepsilon}$:

$$P(S') > \varepsilon (\sum_{s_i \in S'} v_i - 1) \geq \varepsilon (\sum_{s_i \in S} v_i) \geq P(S)$$

As S' satisfies the timing budget in the MaxScore problem, it is a solution better than S, a contradiction. So S is also an optimal solution to the Knapsack problem. □