The K Best-Paths Approach to Approximate Dynamic Programming with Application to Portfolio Optimization

Nicolas Chapados and Yoshua Bengio

Université de Montréal, Dept. IRO, P.O. Box 6128
Montréal, Québec, H3C 3J7, Canada
{chapados,bengioy}@iro.umontreal.ca,
http://www.iro.umontreal.ca/∼{chapados,bengioy}

Abstract. We describe a general method to transform a non-markovian sequential decision problem into a supervised learning problem using a K-best-paths algorithm. We consider an application in financial portfolio management where we can train a controller to directly optimize a Sharpe Ratio (or other risk-averse non-additive) utility function. We illustrate the approach by demonstrating experimental results using a kernel-based controller architecture that would not normally be considered in traditional reinforcement learning or approximate dynamic programming.

1 Introduction

Dynamic programming is a general computational technique for solving sequential optimization problems that can be expressed in terms of an additive cost function [1, 5]. However, it suffers from the so-called *curse of dimensionality*, wherein the computational cost of a solution grows exponentially with the problem dimension (size of the state, action and disturbance spaces). In recent years, many approximation algorithms—notably under the names *reinforcement learning* (RL) or *neurodynamic programming* (NDP)—have been proposed for tackling large-scale problems, in particular by making use of simulation and function approximation methods [6, 22, 20]

Most of these methods remain within the confines of traditional dynamic programming, which assumes that the function to be optimized can be separated as a sum of individual cost-per-time-step terms and, for finite-horizon problems, a terminal cost. Unfortunately, for more complex utility functions, which may *depend on the trajectory of visited states*, dynamic programming does not provide ready solutions.

In finance, it has long been known that the problem of optimal portfolio construction can be expressed as a stochastic optimal control problem, which can be solved by dynamic programming [14, 16, 12]. Still, such formulations assume that the investor is governed by additive utility functions. In practice, this is far from being the case: many risk averse investors care as much about portfolio trajectory as they care about abstract higher moments of a conditional return distribution.

This explains the popularity of performance measures used by practitioners and professional fund managers, such as the Sharpe Ratio [17, 18], Information Ratio [9], Sortino Ratio [21] and Calmar Ratio. A common theme among these utility functions is that they depend on the entire sequence of returns (or statistics of the sequence); they cannot conveniently separated into a form amenable to solution by dynamic programming.

One might argue that dynamic programming should be abandoned altogether, and one ought instead to revert to general nonlinear programming algorithms [4] to attempt optimizing under such utilities. This is the approach followed, in a certain sense, by Bengio's direct optimization of a financial training criterion [2], Moody's direct reinforcement algorithm [13] and Chapados and Bengio's direct maximization of expected returns under a value-at-risk constraint [7]. However, these methods are found lacking in two respects: (i) they still rely on, either time-separable utilities (such as the quadratic utility), or on approximations of trajectory-dependent utilities that enable time-separability, (ii) they fundamentally rely on stochastic gradient descent optimization, and as such can be particularly sensitive to local minima.

This paper investigates a different avenue for portfolio optimization under general utility functions. It relies on formulating portfolio optimization on historical data as a deterministic shortest path problem, where we extract not only the single best path, but the K best paths, yielding, after some transformations, a training set to train a supervised learning algorithm to act as a controller. This controller can directly be used in a portfolio management task.

The paper is organized as follows: first, we introduce the overall approach (section 2); next we investigate in more detail the K best paths algorithm that we used (section 3); we then summarize some experimental portfolio optimization results (sections 4 and 5); and conclude.

2 Problem Formulation

We consider a discrete-time system in an observable state $x_t \in R^N$ at time t, and which must take an action $u_t \in R^M$ at every time step. The system evolves according to a state-transition equation $x_{t+1} = f_t(x_t, u_t, w_t)$, where w_t is a random disturbance. At each time-step, the system experiences a random reward $g_t(x_t, u_t, w_t)$. Our objective is to maximize an expected utility of the sequence of received rewards over a finite horizon $t = 0, \ldots, T$),

$$J_0^*(x_0) = \max_{u_1,\ldots,u_{T-1}} \mathop{\mathrm{E}}_{w_1,\ldots,w_{T-1}} \left[U(g_0, g_1, \ldots, g_T) | \, x_0 \right] . \tag{1}$$

Obviously, if $U(g_0, g_1, \ldots, g_T)$ can be written as $\sum_t g_t$, the finite-horizon problem is solved by writing the *value function* $J_t(x_t)$ in terms of Bellman's recursion,

$$J_T^*(x_T) = g_T(x_T) \tag{2}$$

$$J_t^*(x_t) = \max_{u_t} \mathop{\mathrm{E}}_{w_t} \left[g_t(x_t, u_t, w_t) + J_{t+1}^*(f_t(x_t, u_t, w_t)) \right] . \tag{3}$$

From the value function, the optimal action u_t^* at time t is obtained as that reaching the maximum in the equation above.

2.1 Solving for a General Utility

Our objective is to devise an effective algorithm to obtain optimal actions in the case of a general utility function. Although no recursion such as Bellman's can readily be written in the general case, a key insight lies in the simple observation that, given a **realized trajectory** of rewards, most utility functions (at least those of interest, for instance, in finance) can be computed quickly, in time $O(T)$. Hence, if we are given K such trajectories, we can find the best one under a *general utility function* U in time $O(K(T + \log K))$.

A second observation is that given this sequence of actions, we have obtained what amounts to a set of $\langle \text{state}_t, \text{action}_t \rangle$ pairs at each time-step within the trajectory. We can make use of these as a *training set for a supervised learning algorithm*. In other words, we can bypass completely the step of estimating a value function under the desired utility function, and instead directly train a controller (also called an *actor* in reinforcement learning [22]) to make decisions.

The two preceding observations can be combined into the following algorithm for solving eq. (1):

1. **Generate** a large number of candidate trajectories;
2. **Rescore** (sort) the trajectories under the desired utility function U;
3. Use the best rescored trajectory to **construct a dataset** of $\langle \text{state}, \text{action} \rangle$ pairs; carry out steps 1–3 until the dataset is large enough.
4. Using the dataset from steps 1–3, **train a supervised learning algorithm** to output the action label given the input state.

As is common practice in reinforcement learning [6], this algorithm estimates the expectation in eq. (1) with a sample average over a large number of trajectories. Furthermore, as we shall see below, we can dispense with a generative model of trajectories by using historical data.

2.2 Generating Good Trajectories

It remains the question of *generating good trajectories* in the first place. This is where a K best paths algorithm is involved: under an "easier" (i.e. additive) utility function and a large historical time period (which will become the training set), we use the K best paths algorithm to generate the candidate trajectories of step (1) above. Obviously, both the "easier" and desired utility functions, henceforth respectively called the *source* and *target* utilities, must be correlated, so that searching for good solutions under one function has a high likelihood of yielding good solutions under the other. We discuss this point more fully below. Figure 1 (left part) illustrates schematically the complete algorithm.

2.3 Known Uses

This algorithm is certainly not the first one to make use of a K best paths algorithm: they have been used extensively in speech recognition and natural language processing (e.g. [15]). However, in these contexts, the rescored action

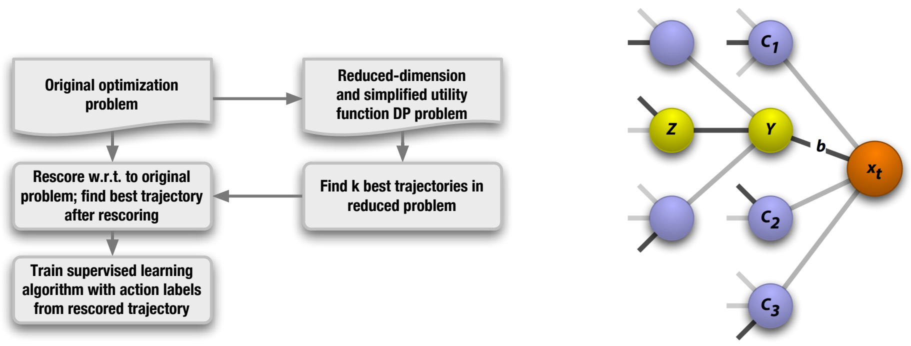
Original optimization problem
Reduced-dimension and simplified utility function DP problem
Rescore w.r.t. to original problem; find best trajectory after rescoring
Find k best trajectories in reduced problem
Train supervised learning algorithm with action labels from rescored trajectory
C1
Z
Y
b
xt
C2
C3

Fig. 1. (Left) Summary of the proposed algorithm for finding good trajectories under a non-additive utility function. **(Right)** Intuition behind the recursive relationship underlying the REA K-best-paths algorithm; see text for details.

labels found by the K best paths are either discarded (speech) or not used beyond proposing alternative hypotheses (NLP). In particular, no use is made of the rescored trajectories for training a controller.

Recent publications in the reinforcement learning literature have explored the idea of converting a RL problem into a supervised learning problem [11]. However, all of the proposed approaches so far have focused on staying within an additive utility function framework and assume the presence of a generative model to construct trajectory histories.

3 Enumerating the K Best Paths

We rely on a very time- and memory-efficient implementation of the Recursive Enumeration Algorithm (REA) of Jiménez and Marzal [10]. This algorithm can be made very effective by implicitly constructing a path from its *differences* with a previous path. It builds upon a generalization of Bellman's recursion of eq.(3) to a statement of optimality of higher-order paths in terms of lower-order ones. Although the precise algorithm statement is not repeated for space reasons, an intuition into the algorithm's working can be obtained from Figure 1 (right part):

- Suppose that the best path to a vertex x_t ends with $\cdots - Z - Y - x_t$.
- According to the REA recursion, the **second best path** up to x_t is given by the best of:
  1. Either the **first best path** up to the immediate predecessors of x_t, namely the *candidate vertices* $\{C_1, C_2, C_3\}$, followed by a transition to x_t.
  2. Or the **second best path** up to Y, followed by the transition b to x_t. The second best path to Y is found by applying the algorithm recursively.

4 Application: Portfolio Optimization

The portfolio optimization setting that we consider is a multi-period, multi-asset problem with transaction costs. We assume that the assets (e.g. stocks, futures) are sufficiently liquid that market impacts can be neglected. We invest in a universe of M asset, and the state x_t at time t is given by

$$x_t = (n_{t,1}, \dots, n_{t,M}, p_{t,1}, \dots, p_{t,M})\,,$$

where $n_{t,i} \in \mathbf{Z}$ is the number of shares of asset i held, and $p_{t,i} \in \mathbf{R}_+$ is the price of asset i at time t. We can only hold an integral number of shares and short (negative) positions are allowed. The possible actions are $u_t \in \mathbf{Z}^M$ which are interpreted as buying or selling the number $u_{t,i}$ of shares asset i. To limit the search space, both $n_{t,i}$ and $u_{t,i}$ may be restricted to a small integer.

The cost function $g_t(x_t, u_t)$ at time t is the \$ amount required to carry out u_t (i.e. establish the desired position), accounting for transaction costs. The source utility function U over all time steps is defined simply as the sum of negative individual costs,[1]

$$U(g_0, \dots, g_{T-1}) = -\sum_{t=0}^{T-1} g_t.$$

Moreover, we impose the constraints that both the initial and final portfolios be empty (i.e. they cannot hold any shares of any asset). With those constraints in place, maximizing U over a time horizon $t = 0, \dots, T$ is equivalent to finding a strategy that *maximizes the terminal wealth* of the investor over the horizon. We call this source utility function the "terminal wealth" utility.

Note that with this utility function, we never need to explicitly represent the cash amount on hand (i.e. it is not part of the state variables) since we use the **value function itself** (*viz.* $J_t^*(x_t)$ in eq. (3)) to stand for the cash currently on hand. This formulation has the advantage that we never need to discretize the cash currently being held, which allows minute price variations and small transaction costs to be handled without loss of precision.

4.1 Target Utilities

Denote by $\upsilon_t = n_t' p_t$ the *portfolio value* at time t, and by

$$\rho_t = \frac{\upsilon_t - \upsilon_{t-1}}{\upsilon_{t-1}}$$

the *portfolio relative return* between time-steps $t-1$ and t.

In the experiments below, we consider two target utility functions:

1. **Average Return Per Time-Step**:

$$\bar{\rho}_T = \frac{1}{T}\rho_t\,.$$

[1] This function would incorporate a discounting factor if the horizon was very long; we assume that this is not the case.

2. **Sharpe Ratio**:
$$SR_T = \frac{\bar{\rho}_T - r_f}{\hat{\sigma}_T},$$
where r_f is an average *government risk-free rate* over the horizon, and $\hat{\sigma}_T$ is the sample standard deviation of returns
$$\hat{\sigma}_T = \frac{1}{T-1}\sum_{t=1}^{T}(\rho_t - \bar{\rho}_T)^2 .$$
The Sharpe Ratio is one of the most widely-used risk-corrected performance measures used by portfolio managers.

4.2 Choosing a Good *K*

It remains to answer the question of choosing an appropriate value of K for a particular problem. To give an indication as to how one might proceed, Figure 2 shows various utility functions as a function of the index of the K-th best path, when extracting 2.5×10^6 paths from a historical price sample.[2] We clearly observe that, beyond a certain point, the quality of the rescored trajectories stops increasing. We investigate when one should stop extracting further trajectories.

Assume that, given a random trajectory i, a source utility function U and a target utility function V, the utility values of the trajectory follow a joint probability distribution $p(u, v)$, where $u = U(i)$ and $v = V(i)$. This is illustrated in Figure 3 (left part). Assume further that we are interested in sampling trajectories that have at least an unconditional target utility of α or better, namely $v \geq \alpha$. Given an observed value of the source utility u, the probability that the target utility be greater than this level is given by
$$P(v \geq \alpha | u) = \frac{1}{\eta(u)} \int_{\alpha}^{\infty} p(u, \tilde{v})\, d\tilde{v}\,,$$
where $\eta(u) = \int_{-\infty}^{\infty} p(u, \tilde{v})\, d\tilde{v}$ is a normalization factor. For each trajectory i, call this probability p_i^{α}. For K trajectories, the probability that *at least one* exceeds α under V can sometimes be computed analytically and is upper-bounded by $\sum_{k=1}^{K} p_k^{\alpha}$. Hence, assuming an estimator of the joint distribution $p(u, v)$, we can compute the number K that would yield a desired confidence of exceeding the target utility threshold α.

The right part of Figure 3 illustrates this idea. It shows a kernel density estimate [23] of the joint distribution between U (terminal wealth utility) and V (Sharpe Ratio utility), along with the regression line between the two (dashed red line), for the same sample path history as reported in Figure 2. Even though the

[2] Four-asset problem (futures on BritishPound, Sugar, Silver, HeatingOil), allow from -3 to $+3$ shares of each asset in the portfolio; maximum variation of $+1$ or -1 share per time-step; proportional transaction costs of 0.5%, trajectory length = 30 time-steps).

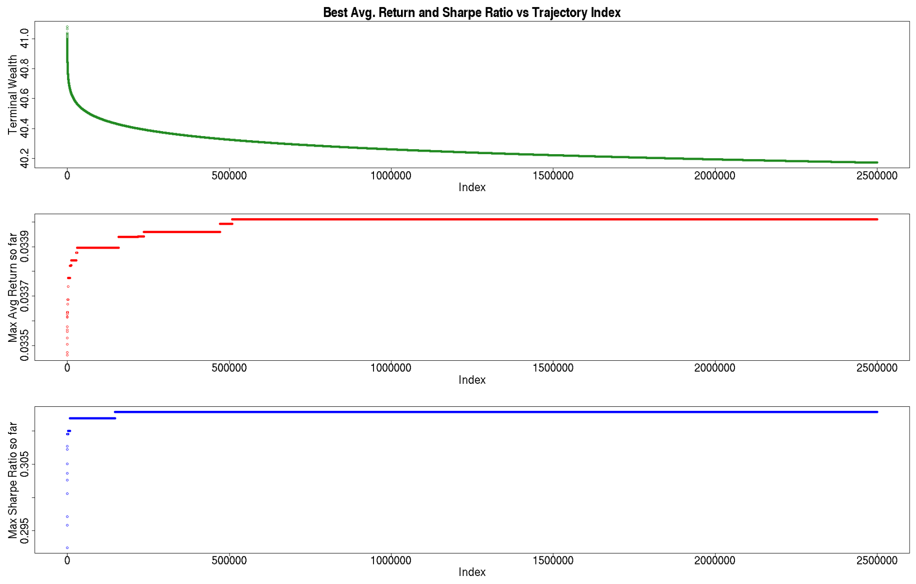

Fig. 2. Utility as a function of the extracted path index (up to 2.5×10^6 extracted paths). **(Top)** The source utility function (terminal wealth), which decreases monotonically by virtue of being extracted in that order by the K-best-paths algorithm. **(Middle)** First target utility: average return per time-step (running maximum value). **(Bottom)** Second target utility: Sharpe Ratio (running maximum value).

slope of the regression line appears small, it is highly statistically significant (t-statistic greater than 200 yielding a p-value of zero-slope null hypothesis smaller than 10^{-16}.) Here is the complete correlation structure between the "terminal wealth" source utility, and both the "average return per time-step" and "Sharpe Ratio" target utilities:

	Terminal Wealth	Avg. Return	Sharpe Ratio
Terminal Wealth	1.00	0.36	0.13
Avg. Return	0.36	1.00	0.45
Sharpe Ratio	0.13	0.45	1.00

5 Experimental Results

We conclude by presenting results on a real-world portfolio management problem. We consider a four-asset problem on commodity futures (feeder cattle, cotton, corn, silver). Since individual commodity futures contracts expire at specific dates, we construct, for each commodity, a *continuous return series* by considering the return series of the contract closest to expiry, and *rolling over* to the next contract at the beginning of the contract expiration month. This return series is converted back into a price series.

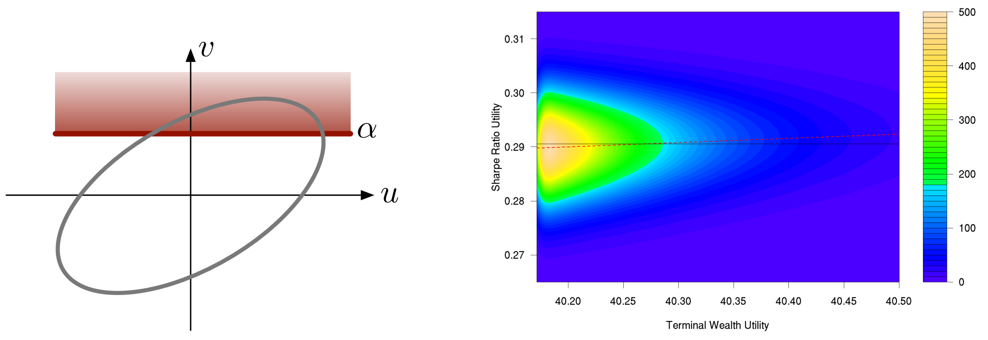

Fig. 3. (Left) Exploiting the correlation between the source and target utility functions: we want the number K of extracted paths to be large enough to sample "good target utility" region (shaded) well enough. **(Right)** Kernel density estimate of the relationship between the terminal wealth (source utility) and the Sharpe Ratio (target utility); even though the dashed red regression line has a small positive slope, it is extremely statistically significant.

Instead of a single train–test split, the simulation is run in the context of *sequential validation* [3]. In essence, this procedure uses data up to t to train, then tests on the single point $t+1$ (and produces a single out-of-sample performance result), then adds $t+1$ to the training set, tests on $t+2$, and so forth, until the end of the data. An initial training set of 1008 points (four years of daily trading data) was used.

From a methodology standpoint, we proceeded in two steps: (i) computing a controller training set with the targets being the "optimal" action under the Sharpe Ratio utility, (ii) running a simulation of a controller trained with that training set, and comparing against a naïve controller. We describe each in turn.

5.1 Constructing the Training Set

The construction of the training set follows the outline set forth in Figure 1 (left part). To run the K-best-paths algorithm, we allow from -1 to $+1$ shares of each asset in portfolio, maximum variation in each asset of -1 to $+1$ shares per time-step, and proportional transaction costs of 0.5%.

We use, as targets in the training set, the "optimal" action under the Sharpe Ratio utility function, obtained after rescoring on 1.0^6 paths. Since extracting trajectories spanning several thousands time-steps is rather memory intensive, we reverted to a slightly suboptimal local-trajectory solution:

– We split the initial asset price history (spanning approximately 1500 days) into overlapping 30-day windows. The overlap between consecutive windows is 22 days.

– We solve the Sharpe Ratio optimization problem independently within each 30-day window.
– To account for boundary effects, we drop the seven first and last actions within each window.
– We concatenate the remaining actions across windows.

We thus obtain the sequence of target actions across a long horizon.[3]

For the input part of the training set, we used:

– The current portfolio state (4 elements);
– The asset returns at horizons of length $h \in \{1, 22, 252, 378\}$ days (16 elements)

5.2 Controller Architecture

Given the relatively small size of the training set, we could afford to use *kernel ridge regression* (KRR) as the controller architecture [19]. Any (preferably non-linear) regression algorithm, including neural networks, can be brought to bear. For an input vector $\mathbf{x}$, the forecast given by the KRR estimator has a particularly simple form,

$$f(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i)(M + \lambda\, I)^{-1}\mathbf{y}$$

where $k(\mathbf{x}, \mathbf{x}_i)$ is the vector of kernel evaluations of the test vector $\mathbf{x}$ against all elements of the training set $\mathbf{x}_i$, M is the Gram matrix on the training set, and $\mathbf{y}$ is the matrix of targets (in this case, the optimal actions under the Sharpe Ratio utility, found in the previous section). We made use of a standard Gaussian kernel, with $\sigma = 3.0$ and fixed $\lambda = 10.0$ (both found by cross-validation on a non-overlapping time period).

5.3 Results

For validation purposes, we compared against two benchmarks models:

– The same KRR controller architecture, but with the targets replaced by the 10-day ahead 22-day asset returns, followed by a sign$(\cdot)$ operation. Hence, this controller takes a long position if it believes that an asset will experience a positive return over the short-term, and symmetrically takes a short position if it believes otherwise. For this controller, the current portfolio state is not included within the inputs, since it cannot be established at training time, yielding a total of 16 inputs instead of 20.
– The same targets as previously, but with a linear forecasting model (estimated by ordinary least squares) instead of KRR.

[3] It is obvious that this approach will not capture long-range dependencies between actions (more than 30 days in this case). However, for the specific case of the Sharpe Ratio, the impact of an action is not usually felt at a long distance, and thus this approach was found to work well in practice. Needless to say, it will have to be adjusted to other more complex utility functions.

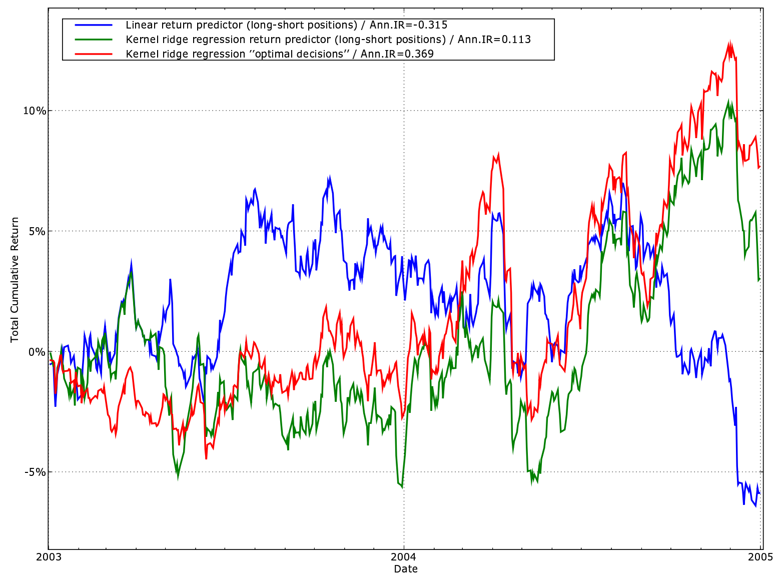

Fig. 4. Out-of-sample financial simulation results for the 2003–2004 period, comparing a controller trained with the proposed algorithm (red; Information Ratio = 0.369) against two benchmark models (resp. IR=−0.315 for linear controller (blue) and IR=0.113 for KRR controller (green)).

Performance results over the out-of-sample period 2003–2004 (inclusive) appear in Figure 4. Further performance statistics appear in Table 1. We observe that, at constant annual volatility (around 10%), the KRR model trained with the proposed methodology (Sharpe Ratio utility function) outperforms the two benchmarks.

6 Discussion and Future Work

The current results, although demonstrating the value of the proposed algorithm, probably raise more questions than they answer. In particular, we did not consider the impact on rescoring performance of the choice of source utility function. Moreover, we so far ignored the major efficiency gains that can be achieved by an intelligent pruning of the search graph, either in the form of beam searching or action enumeration heuristics.

Another question that future work should investigate is that with the methodology proposed here, the supervised learning algorithm optimizes the controller with respect to a regression (or classification) criterion which can disagree with

Table 1. Financial performance statistics for the out-of-sample simulation results on the 2003–2004 period.

	Benchmark Linear Model	**Benchmark KRR Model**	**Sharpe Ratio "Optimal" Targets; KRR Model**
Average monthly relative return	-0.14%	0.20%	0.43%
Monthly return standard deviation	2.68%	2.89%	3.07%
Worst monthly return	-5.80%	-6.21%	-7.14%
Best monthly return	5.15%	6.80%	6.73%
Annual Information Ratio†	-0.31	0.11	0.37
Average daily net exposure	-5.3%	10.8%	28.3%
Average portfolio effective leverage	74.7%	73.4%	59.6%
Average monthly portfolio turnover	499%	175%	80%
Average daily hit ratio	48.1%	50.6%	51.6%

† with respect to U.S. 3-month T-Bill.

the target utility when the target training set trajectory is not perfectly reproduced. In order to achieve good generalization, because of unpredictability in the data and because of the finite sample size, the trained controller will most likely not reach the supervised learning targets corresponding to the selected trajectory. However, among all the controllers that do not reach these targets and that are reachable by the learning algorithm, we will choose one that minimizes an ordinary regression or classification criterion, rather than one that maximizes our financial utility. Ideally, we would like to find a compromise between finding a "simple" controller (from a low-capacity class) and finding a controller which yields high empirical utility. One possible way to achieve such a trade-off in our context would be to consider the use of a *weighted training criterion* (e.g. similar to ones used to train boosted weak learners in Adaboost [8]) that penalizes regression or classification errors more or less according to how much the target utility would decrease by taking the corresponding "wrong" decisions, different from the target decision.

References

1. R. E. Bellman. *Dynamic Programming.* Princeton University Press, NJ, 1957.
2. Yoshua Bengio. Using a financial training criterion rather than a prediction criterion. *International Journal of Neural Systems*, 8(4):433–443, 1997.
3. Yoshua Bengio and Nicolas Chapados. Extensions to metric based model selection. *Journal of Machine Learning Research*, 3(7–8):1209–1228, October–November 2003.
4. Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, second edition, 2000.
5. Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, second edition, 2001.
6. Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

7. Nicolas Chapados and Yoshua Bengio. Cost functions and model combination for VaR-based asset allocation using neural networks. *IEEE Transactions on Neural Networks*, 12:890–906, Juillet 2001.
8. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of Thirteenth International Conference*, pages 148–156, USA, 1996. ACM.
9. Richard C. Grinold and Ronald N. Kahn. *Active Portfolio Management*. McGraw Hill, 2000.
10. Víctor Manuel Jiménez Pelayo and Andrés Marzal Varó. Computing the K shortest paths: a new algorithm and an experimental comparison. In *Proc. 3rd Worksh. Algorithm Engineering*, July 1999.
11. John Langford and Bianca Zadrozny. Relating reinforcement learning performance to classification performance. In *22nd International Conference on Machine Learning (ICML 2005)*, Bonn, Germany, August 2005.
12. Robert C. Merton. Lifetime portfolio selection under uncertainty: The continuous-time case. *The Review of Economics and Statistics*, 51(3):247–257, 1969.
13. John Moody and Matthew Saffel. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889, 2001.
14. Jan Mossin. Optimal multiperiod portfolio policies. *The Journal of Business*, 41(2):215–229, 1968.
15. L. Rabiner and B.H. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.
16. Paul A. Samuelson. Lifetime portfolio selection by dynamic stochastic programming. *The Review of Economics and Statistics*, 51(3):239–246, 1969.
17. W. F. Sharpe. Mutual fund performance. *Journal of Business*, pages 119–138, January 1966.
18. W. F. Sharpe. The sharpe ratio. *The Journal of Portfolio Management*, 21(1):49–58, 1994.
19. John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
20. Jennie Si, Andrew G. Barto, Warren B. Powell, and Don Wunsch, editors. *Handbook of Learning and Approximate Dynamic Programming*. IEEE Press Series on Computational Intelligence) (Hardcover. Wiley–IEEE Press, 2004.
21. F. Sortino and L. Price. Performance measurement in a downside risk framework. *The Journal of Investing*, pages 59–65, Fall 1994.
22. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
23. M.P. Wand and M.C. Jones. *Kernel Smoothing*. Chapman and Hall, London, 1995.