HAL
archives-ouvertes.fr

Non linear programming for stochastic dynamic programming

Olivier Teytaud, Sylvain Gelly

▶ **To cite this version:**

Olivier Teytaud, Sylvain Gelly. Non linear programming for stochastic dynamic programming. Icinco 2007, 2007, Angers, France. inria-00173202

HAL Id: inria-00173202
https://hal.inria.fr/inria-00173202

Submitted on 19 Sep 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

NONLINEAR PROGRAMMING IN APPROXIMATE DYNAMIC PROGRAMMING: BANG-BANG SOLUTIONS, STOCK-MANAGEMENT AND UNSMOOTH PENALTIES

Olivier Teytaud, Sylvain Gelly
TAO (Inria, Univ. Paris-Sud, UMR CNRS-8623)
olivier.teytaud@inria.fr, gelly@lri.fr

Keywords: Evolutionary computation and control, Optimization algorithms.

Abstract: Many stochastic dynamic programming tasks in continuous action-spaces are tackled through discretization. We here avoid discretization; then, approximate dynamic programming (ADP) involves (i) many learning tasks, performed here by Support Vector Machines, for Bellman-function-regression (ii) many non-linear-optimization tasks for action-selection, for which we compare many algorithms. We include discretizations of the domain as particular non-linear-programming-tools in our experiments, so that by the way we compare optimization approaches and discretization methods. We conclude that robustness is strongly required in the non-linear-optimizations in ADP, and experimental results show that (i) discretization is sometimes inefficient, but some specific discretization is very efficient for "bang-bang" problems (ii) simple evolutionary tools outperform quasi-random in a stable manner (iii) gradient-based techniques are much less stable (iv) for most high-dimensional "less unsmooth" problems Covariance-Matrix-Adaptation is first ranked.

1 NON-LINEAR OPTIMIZATION IN STOCHASTIC DYNAMIC PROGRAMMING (SDP)

Some of the most traditional fields of stochastic dynamic programming, e.g. energy stock-management, which have a strong economic impact, have not been studied thoroughly in the reinforcement learning or approximate-dynamic-programming (ADP) community. This is damageable to reinforcement learning as it has been pointed out that there are not yet many industrial realizations of reinforcement learning. Energy stock-management leads to continuous problems that are usually handled by traditional linear approaches in which (i) convex value-functions are approximated by linear cuts (leading to piecewise linear approximations (PWLA)) (ii) decisions are solutions of a linear-problem. However, this approach does not work in large dimension, due to the curse of dimensionality which strongly affects PWLA. These problems should be handled by other learning tools. However, in this case, the action-selection, minimizing the expected cost-to-go, can't be anymore done using linear-programming, as the Bellman function is no more a convex PWLA.

The action selection is therefore a nonlinear programming problem. There are not a lot of works dealing with continuous actions, and they often do not study the non-linear optimization step involved in action selection. In this paper, we focus on this part: we compare many non-linear optimization-tools, and we also compare these tools to discretization techniques to quantify the importance of the action-selection step.

We here roughly introduce stochastic dynamic programming. The interested reader is referred to (Bertsekas and Tsitsiklis, 1996) for more details.

Consider a dynamical system that stochastically evolves in time depending upon your decisions. Assume that time is discrete and has finitely many time steps. Assume that the total cost of your decisions is the sum of instantaneous costs. Precisely:

$$\begin{aligned} cost &= c_1 + c_2 + \cdots + c_T \\ c_i &= c(x_i, d_i), \quad x_i = f(x_{i-1}, d_{i-1}, \omega_i) \\ d_{i-1} &= strategy(x_{i-1}, \omega_i) \end{aligned}$$

where x_i is the state at time step i, the ω_i are a ran-

dom process, *cost* is to be minimized, and *strategy* is the decision function that has to be optimized. We are interested in a control problem: the element to be optimized is a function.

Stochastic dynamic programming, a tool to solve this control problem, is based on Bellman's optimality principle that can be informally stated as follows:

"Take the decision at time step t such that the sum "cost at time step t due to your decision" plus "expected cost from time step $t+1$ to ∞" is minimal."

Bellman's optimality principle states that this strategy is optimal. Unfortunately, it can only be applied if the expected cost from time step $t+1$ to ∞ can be guessed, depending on the current state of the system and the decision. Bellman's optimality principle reduces the control problem to the computation of this function. If x_t can be computed from x_{t-1} and d_{t-1} (i.e., if f is known) then the control problem is reduced to the computation of a function

$$V(t,x_t) = E[c(x_t,d_t) + c(x_{t+1},d_{t+1}) + \cdots + c(x_T,d_T)]$$

Note that this function depends on the strategy (we omit for short dependencies on the random process). We consider this expectation for any optimal strategy (even if many strategies are optimal, V is uniquely determined as it is the same for any optimal strategy).

Stochastic dynamic programming is the computation of V backwards in time, thanks to the following equation:

$$\begin{aligned} V(t,x_t) &= \inf_{d_t} c(x_t,d_t) + EV(t+1,x_{t+1}) \\ &\text{or equivalently} \\ V(t,x_t) &= \inf_{d_t} c(x_t,d_t) + EV(t+1,f(x_t,d_t)) \quad (1) \end{aligned}$$

For each t, $V(t,x_t)$ is computed for many values of x_t, and then a learning algorithm (here by support vector machines) is applied for building $x \mapsto V(t,x)$ from these examples. Thanks to Bellman's optimality principle, the computation of V is sufficient to define an optimal strategy. This is a well known, robust solution, applied in many areas including power supply management. A general introduction, including learning, is (Bertsekas, 1995; Bertsekas and Tsitsiklis, 1996). Combined with learning, it can lead to positive results in spite of large dimensions. Many developments, including RTDP and the field of reinforcement learning, can be found in (Sutton and Barto, 1998).

Equation 1 is used many many times during a run of dynamic programming. For T time steps, if N points are required for efficiently approximating each V_t, then there are $T \times N$ optimizations. Furthermore, the derivative of the function to optimize is not always available, due to the fact that complex simulators are sometimes involved in the transition f. Convexity sometimes holds, but sometimes not. Binary variables are sometimes involved, e.g. in power plants management. This suggests that evolutionary algorithms are a possible tool.

1.1 Robustness in non-linear optimization

Robustness is one of the main issue in non-linear optimization and has various meanings.

1. A first meaning is the following: robust optimization is the search of x such that in the neighborhood of x the fitness is good, and not only at x. In particular, (DeJong, 1992) has introduced the idea that evolutionary algorithms are not function-optimizers, but rather tools for finding wide areas of good fitness.

2. A second meaning is that robust optimization is the avoidance of local minima. It is known that iterative deterministic methods are often more subject to local minima than evolutionary methods; however, various forms of restarts (relaunch the optimization from a different initial point) can also be efficient for avoiding local minima.

3. A third possible meaning is the robustness with respect to fitness noise. Various models of noise and conclusions can be found in (Jin and Branke, 2005; Sendhoff et al., 2004; Tsutsui, 1999; Fitzpatrick and Grefenstette, 1988; Beyer et al., 2004).

4. A fourth possible meaning is the robustness with respect to unsmooth fitness functions, even in cases in which there's no local minima. Evolutionary algorithms are usually rank-based (the next iterate point depends only on the fitnesses rank of previously visited points), therefore do not depend on increasing transformations of the fitness-function. It is known that they have optimality properties w.r.t this kind of transformations (Gelly et al., 2006). For example, $\sqrt{||x||}$ (or some C^∞ functions close to this one) lead to a very bad behavior of standard Newton-based methods like BFGS (Broyden., 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) whereas a rank-based evolutionary algorithm behaves the same for $||x||^2$ and $\sqrt{||x||}$.

5. The fifth possible meaning is the robustness with respect to the non-deterministic choices made by the algorithm. Even algorithms that are considered as deterministic often have a random part[1]: the choice of the initial point. Population-based methods are more robust in this sense, even if they use more randomness for the initial step (full random initial population

[1] Or, if not random, a deterministic but arbitrary part, such as the initial point or the initial step-size.

compared to only one initial point): a bad initialization which would lead to a disaster is much more unlikely.

The first sense of robustness given above, i.e. avoiding too narrow areas of good fitness, fully applies here. Consider for example a robot navigating in an environment in order to find a target. The robot has to avoid obstacles. The strict optimization of the cost-to-go leads to choices just tangent to obstacles. As at each step the learning is far from perfect, then being tangent to obstacles leads to hit the obstacles in 50 % of cases. We see that some local averaging of the fitness is suitable.

The second sense, robustness in front of non-convexity, of course also holds here. Convex and non-convex problems both exist. The law of increasing marginal costs implies the convexity of many stock management problems, but approximations of V are usually not convex, even if V is theoretically convex. Almost all problems of robotics are not convex.

The third sense, fitness (or gradient) noise, also applies. The fitness functions are based on learning from finitely many examples. Furthermore, the gradient, when it can be computed, can be pointless even if the learning is somewhat successfull; even if $\hat{f}$ approximates f in the sense that $||f - \hat{f}||_p$ is small, $\nabla \hat{f}$ can be very far from ∇f.

The fourth sense is also important. Strongly discontinuous fitnesses can exist: obstacle avoidance is a binary reward, as well as target reaching. Also, a production-unit can be switched on or not, depending on the difference between demand and stock-management, and that leads to large binary-costs.

The fifth sense is perhaps the most important. SDP can lead to thousands of optimizations, similar to each other. Being able of solving very precisely 95 % of families of optimization problems is not the goal; here it's better to solve 95 % of any family of optimization problems, possibly in a suboptimal manner. We do think that this requirement is a main explanation of results below.

Many papers have been devoted to ADP, but comparisons are usually far from being extensive. Many papers present an application of one algorithm to one problem, but do not compare two techniques. Problems are often adapted to the algorithm, and therefore comparing results is difficult. Also, the optimization part is often neglected; sometimes not discussed, and sometimes simplified to a discretization.

In this paper, we compare experimentally many non-linear optimization-tools. The list of methods used in the comparison is given in 2. Experiments are presented in section 3. Section 4 concludes.

2 ALGORITHMS USED IN THE COMPARISON

We include in the comparison standard tools from mathematical programming, but also evolutionary algorithms and some discretization techniques. Evolutionary algorithms can work in continuous domains (Bäck et al., 1991; Bäck et al., 1993; Beyer, 2001); moreover, they are compatible with mixed-integer programming (e.g. (Bäck and Schütz, 1995)). However, as there are not so many algorithms that could naturally work on mixed-integer problems and in order to have a clear comparison with existing methods, we restrict our attention to the continuous framework. We can then easily compare the method with tools from derivative free optimization (Conn et al., 1997), and limited-BFGS with finite differences (Zhu et al., 1994; Byrd et al., 1995). We also considered some very naive algorithms that are possibly interesting thanks to the particular requirement of robustness within a moderate number of iterates: random search and some quasi-random improvements. The discretization techniques are techniques that test a predefined set of actions, and choose the best one. As detailed below, we will use dispersion-based samplings or discrepancy-based samplings.

We now provide details about the methods integrated in the experiments. For the sake of neutrality and objectivity, none of these source codes has been implemented for this work: they are all existing codes that have been integrated to our platform, except the baseline algorithms.

- random search: randomly draw N points in the domain of the decisions ; compute their fitness ; consider the minimum fitness.
- quasi-random search: idem, with low discrepancy sequences instead of random sequences (Niederreiter, 1992). Low discrepancy sequences are a wide area of research (Niederreiter, 1992; Owen, 2003), with clear improvements on Monte-Carlo methods, in particular for integration but also for learning (Cervellera and Muselli, 2003), optimization (Niederreiter, 1992; Auger et al., 2005), path planning (Tuffin, 1996). Many recent works are concentrated on high dimension (Sloan and Woźniakowski, 1998; Wasilkowski and Wozniakowski, 1997), with in particular successes when the "true" dimensionality of the underlying distribution or domain is smaller than the apparent one (Hickernell, 1998), or with scrambling-techniques (L'Ecuyer and Lemieux, 2002).
- Low-dispersion optimization is similar, but uses low-dispersion sequences (Niederreiter, 1992;

Lindemann and LaValle, 2003; LaValle et al., 2004) instead of random i.i.d sequences ; low-dispersion is related to low-discrepancy, but easier to optimize. A dispersion-criterion is

$$Dispersion(P) = \sup_{x \in D} \inf_{p \in P} d(x,p) \quad (2)$$

where d is the euclidean distance. It is related to the following (easier to optimize) criterion (to be maximized and not minimized):

$$Dispersion_2(P) = \inf_{(x_1,x_2) \in D^2} d(x_1,x_2) \quad (3)$$

we use eq. 3 in the sequel of this paper. We optimize dispersion in a greedy manner: each point x_n is optimal for the dispersion of $x_1,\ldots,x_n$ conditionally to $x_1,\ldots,x_{n-1}$; i.e. $x_1 = (0.5,0.5,\ldots,0.5)$, x_2 is such that $Dispersion_2(\{x_1,x_2\})$ is maximal, and x_n is such that $Dispersion_2(\{x_1,\ldots,x_{n-1},x_n\})$ is minimal. This sequence has the advantage of being much faster to compute than the non-greedy one, and that one does not need a priori knowledge of the number of points. Of course, it is not optimal for eq. 3 or eq. 2.

- Equation 3 pushes points on the frontier, what is not the case in equation 2 ; therefore, we also considered low-dispersion sequences "far-from-frontier", where equation 3 is replaced by:

$$Dispersion_3(P) = \inf_{(x_1,x_2) \in D^2} d(x_1, \{x_2\} \cup D') \quad (4)$$

  As for $Dispersion_2$, we indeed used the greedy and incremental counterpart of eq. 4.

- CMA-ES (EO and openBeagle implementation): an evolution strategy with adaptive covariance matrix (Hansen and Ostermeier, 1996; Keijzer et al., 2001; Gagné, 2005).

- The Hooke & Jeeves (HJ) algorithm (Hooke and Jeeves, 1961; Kaupe, 1963; Wright, 1995), available at `http://www.ici.ro/camo/unconstr/hooke.htm`: a geometric local method implemented in C by M.G. Johnson.

- a genetic algorithm (GA), from the *sgLibrary* (`http://opendp.sourceforge.net`). It implements a very simple genetic algorithm where the mutation is an isotropic Gaussian of standard deviation $\frac{\sigma}{\sqrt[d]{n}}$ with n the number of individuals in the population and d the dimension of space. The crossover between two individuals x and y gives birth to two individuals $\frac{1}{3}x+\frac{2}{3}y$ and $\frac{2}{3}x+\frac{1}{3}y$. Let $\lambda_1,\lambda_2,\lambda_3,\lambda_4$ be such that $\lambda_1+\lambda_2+\lambda_3+\lambda_4 = 1$; we define S_1 the set of the $\lambda_1.n$ best individuals, S_2 the $\lambda_2.n$ best individuals among the others. At each generation, the new offspring is (i) a copy of S_1 (ii) $n\lambda_2$ cross-overs between individuals from S_1 and individuals from S_2 (iii) $n\lambda_3$ mutated copies of individuals from S_1 (iv) $n\lambda_4$ individuals randomly drawn uniformly in the domain. The parameters are $\sigma = 0.08, \lambda_1 = 1/10, \lambda_2 = 2/10, \lambda_3 = 3/10, \lambda_4 = 4/10$; the population size is the square-root of the number of fitness-evaluations allowed. These parameters are standard ones from the library. We also use a "no memory" (GANM) version, that provides as solution the best point in the final offspring, instead of the best visited point. This is made in order to avoid choosing a point from a narrow area of good fitness.

- limited-BFGS with finite differences, thanks to the LBFGSB library (Zhu et al., 1994; Byrd et al., 1995). Roughly speaking, LBFGS uses an approximated Hessian in order to approximate Newton-steps without the huge computational and space cost associated to the use of a full Hessian.

In our experiments with restart, any optimization that stops due to machine precision is restarted from a new random (independent, uniform) point.

For algorithms based on an initial population, the initial population is chosen randomly (uniformly, independently) in the domain. For algorithms based on an initial point, the initial point is the middle of the domain. For algorithms requiring step sizes, the step size is the distance from the middle to the frontier of the domain (for each direction). Other parameters were chosen by the authors with equal work for each method on a separate benchmark, and then plugged in our dynamic programming tool. The detailed parametrization is available in `http://opendp.sourceforge.net`, with the command-line generating tables of results.

Some other algorithms have been tested and rejected due to their huge computational cost: the DFO-algorithm from Coin (Conn et al., 1997),`http://www.coin-or.org/`; Cma-ES from Beagle (Hansen and Ostermeier, 1996; Gagné, 2005) is similar to Cma-ES from EO(Keijzer et al., 2001) and has also been removed.

3 EXPERIMENTS

3.1 Experimental settings

The characteristics of the problems are summarized in table 1; problems are scalable and experiments are performed with dimension (i) the baseline dimension in table 1 (ii) twice this dimensionality (iii) three

times (iv) four times. Both the state space dimension and the action space are multiplied. Results are presented in tables below. The detailed experimental setup is as follows: the learning of the function value is performed by SVM with Laplacian-kernel (SVMTorch,(Collobert and Bengio, 2001)), with hyperparameters heuristically chosen; each optimizer is allowed to use a given number of points (specified in tables of results); 300 points for learning are sampled in a quasi-random manner for each time step, non-linear optimizers are limited to 100 function-evaluations. Each result is averaged among 66 runs. We can summarize results below as follows. Experiments are performed with:

- 2 algorithms for gradient-based methods (LBFGS and LBFGS with restart),
- 3 algorithms for evolutionary algorithms (EO-CMA, GeneticAlgorithm, GeneticAlgorithm-NoMemory),
- 4 algorithms for best-of-a-predefined sample (Low-Dispersion, Low-Dispersion "fff", Random, Quasi-Random),
- 2 algorithms for pattern-search methods (Hooke&Jeeves, Hooke&Jeeves with restart)

3.2 Results

Results varies from one benchmark to another. We have a wide variety of benchmarks, and no clear superiority of one algorithm onto others arises. E.g., CMA is the best algorithm in some cases, and the worst one in some others. One can consider that it would be better to have a clear superiority of one and only one algorithm, and therefore a clear conclusion. Yet, it is better to have plenty of benchmarks, and as a by-product of our experiments, we claim that conclusions extracted from one or two benchmarks, as done in some papers, are unstable, in particular when the benchmark has been adapted to the question under study. The significance of each comparison (for one particular benchmark) can be quantified and in most cases we have sufficiently many experiments to make results significant. But, this significance is for each benchmark independently; in spite of the fact that we have chosen a large set of benchmarks, coming from robotics or industry, we can not conclude that the results could be extended to other benchmarks. However, some (relatively) stable conclusions are:

- For best-of-a-predefined-set-of-points:
  - Quasi-random search is better than random search in 17/20 experiments with very good overall significance and close to random in the 3 remaining experiments.
  - But low-dispersion, that is biased in the sense that it "fills" the frontier, is better in 10 on 20 benchmarks only; this is problem-dependent, in the sense that in the "away" or "arm" problem, involving nearly bang-bang solutions (i.e. best actions are often close to the boundary for each action-variable) the Low-dispersion-approach is often the best. LD is the best with strong significance for many *-problems (in which bang-bang solutions are reasonnable).
  - And low-dispersion-fff, that is less biased, outperforms random for 14 on 20 experiments (but is far less impressive for bang-bang-problems).
- For order-2 techniques[2]: LBFGSB outperforms quasi-random-optimization for 9/20 experiments; Restart-LBFGSB outperforms quasi-random optimization for 10/20 experiments. We suggest that this is due to (i) the limited number of points (ii) the non-convex nature of our problems (iii) the cost of estimating a gradient by finite-differences that are not in favor of such a method. Only comparison-based tools were efficient. CMA is a particular tool in the sense that it estimates a covariance (which is directly related to the Hessian), but without computing gradients; a drawback is that CMA is much more expensive (much more computation-time per iterate) than other methods (except BFGS sometimes). However it is sometimes very efficient, as being a good compromise between a precise information (the covariance related to the Hessian) and fast gathering of information (no gradient computation). In particular, CMA was the best algorithm for all stock-management problems (involving precise choices of actions) as soon as the dimension is ≥ 8, with in most cases strong statistical significance.
- The pattern-search method (the Hooke&Jeeves algorithm with Restart) outperforms quasi-random for 10 experiments on 20.
- For the evolutionary-algorithms:
  - EoCMA outperforms Quasi-Random in 5/20 experiments. These 5 experiments are all stock-management in high-dimension, and are often very significant.
  - GeneticAlgorithm outperforms Quasi-Random in 14/20 experiments and Random in 17/20 experiments (with significance in most cases). This algorithm is probably the most stable one in our experiments. GeneticAlgorithmNoMemory outperforms Quasi-Random

[2] We include CMA in order-2 techniques in the sense that it uses a covariance matrix which is strongly related to the Hessian.

in 14/20 experiments and Random in 15/20 experiments.

Due to length-constraints, the detailed results, for each method and with confidence intervals, are reported to `http://www.lri.fr/~teytaud/sefordplong.pdf`. We summarize the results in table 2.

4 CONCLUSION

We presented an experimental comparison of non linear optimization algorithms in the context of ADP. The comparison involves evolutionary algorithms, (quasi-)random search, discretizations, and pattern-search-optimization. ADP has strong robustness requirements, thus the use of evolutionary algorithms, known for their robustness properties, is relevant. These experiments are made in a neutral way; we did not work more on a particular algorithm than another. Of course, perhaps some algorithms require more work to become efficient on the problem. The reader can download our source code, modify the conditions, check the parametrization, and experiment himself. Therefore, our source code is freely available at http://opendp.sourceforge.net for further experiments. A Pascal-NoE challenge (`www.pascal-network.org/Challenges/`) will be launched soon so that anyone can propose his own algorithms.

Our main claims are:

- **High-dimensional stock-management**. CMA-ES is an efficient evolution-strategy when dimension increases and for "less-unsmooth" problems. It is less robust than the GA, but appears as a very good compromise for the important case of high-dimensional stock-management problems. We do believe that CMA-ES, which is very famous in evolution strategies, is indeed a very good candidate for non-linear optimization as involved in high-dimensional-stock-management where there is enough smoothness for covariance-matrix-adaptation. LBFGS is not satisfactory: in ADP, convexity or derivability are not reliable assumptions, as explained in section 1.1, even if the law of increasing marginal cost applies. Experiments have been performed with dimension ranging from 4 to 16, without heuristic dimension reduction or problem-rewriting in smaller dimension, and results are statistically clearly significant. However, we point out that CMA-ES has a huge computational cost. The algorithms are compared above in the case of a given number of calls to the fitness; this is only a good criterion when the computational cost is mainly the fitness-evaluations. For very-fast fitness-evaluations, CMA-ES might be prohibitively too expensive.
- **Robustness requirement in highly unsmooth problems.** Evolutionary techniques are the only ones that outperform quasi-random-optimization in a stable manner even in the case of very unsmooth penalty-functions (see **-problems in the Table 2). The GA is not always the best optimizer, but in most cases it is at least better than random; we do believe that the well-known robustness of evolutionary algorithms, for the five meanings of robustness pointed out in section 1.1, are fully relevant for ADP.
- **A natural tool for generating bang-bang-efficient controlers.** In some cases (typically bang-bang problems) the LD-discretization introducing a bias towards the frontiers are (unsurprisingly) the best ones, but for other problems LD leads to the worst results of all techniques tested. This is not a trivial result, as this points out LD as a natural way of generating nearly bang-bang solutions, which depending on the number of function-evaluations allowed, samples the middle of the action space, and then the corners, and then covers the whole action space (what is probably a good "anytime" behavior). A posteriori, LD appears as a natural candidate for such problems, but this was not so obvious a priori.

REFERENCES

Auger, A., Jebalia, M., and Teytaud, O. (2005). Xse: quasi-random mutations for evolution strategies. In *Proceedings of EA'2005*, pages 12–21.

Bäck, T., Hoffmeister, F., and Schwefel, H.-P. (1991). A survey of evolution strategies. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the 4^{th} International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann.

Bäck, T., Rudolph, G., and Schwefel, H.-P. (1993). Evolutionary programming and evolution strategies: Similarities and differences. In Fogel, D. B. and Atmar, W., editors, *Proceedings of the 2^{nd} Annual Conference on Evolutionary Programming*, pages 11–22. Evolutionary Programming Society.

Bäck, T. and Schütz, M. (1995). Evolution strategies for mixed-integer optimization of optical multilayer systems. In McDonnell, J. R., Reynolds, R. G., and Fogel, D. B., editors, *Proceedings of the 4^{th} Annual Conference on Evolutionary Programming*. MIT Press.

Bertsekas, D. (1995). *Dynamic Programming and Optimal Control, vols I and II*. Athena Scientific.

Bertsekas, D. and Tsitsiklis, J. (1996). Neuro-dynamic programming, athena scientific.

Beyer, H.-G. (2001). *The Theory of Evolutions Strategies*. Springer, Heidelberg.

Beyer, H.-G., Olhofer, M., and Sendhoff, B. (2004). On the impact of systematic noise on the evolutionary optimization performance - a sphere model analysis, genetic programming and evolvable machines, vol. 5, no. 4, pp. 327 360.

Broyden., C. G. (1970). The convergence of a class of double-rank minimization algorithms 2, the new algorithm. j. of the inst. for math. and applications, 6:222-231.

Byrd, R., Lu, P., Nocedal, J., and C.Zhu (1995). A limited memory algorithm for bound constrained optimization. *SIAM J. Scientific Computing, vol.16, no.5*.

Cervellera, C. and Muselli, M. (2003). A deterministic learning approach based on discrepancy. In *Proceedings of WIRN'03, pp53-60*.

Collobert, R. and Bengio, S. (2001). Svmtorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160.

Conn, A., Scheinberg, K., and Toint, L. (1997). Recent progress in unconstrained nonlinear optimization without derivatives.

DeJong, K. A. (1992). Are genetic algorithms function optimizers ? In Manner, R. and Manderick, B., editors, *Proceedings of the 2nd Conference on Parallel Problems Solving from Nature*, pages 3–13. North Holland.

Fitzpatrick, J. and Grefenstette, J. (1988). Genetic algorithms in noisy environments, in machine learning: Special issue on genetic algorithms, p. langley, ed. dordrecht: Kluwer academic publishers, vol. 3, pp. 101 120.

Fletcher, R. (1970). A new approach to variable-metric algorithms. computer journal, 13:317-322.

Gagné, C. (2005). Openbeagle 3.1.0-alpha.

Gelly, S., Ruette, S., and Teytaud, O. (2006). Comparison-based algorithms: worst-case optimality, optimality w.r.t a bayesian prior, the intraclass-variance minimization in eda, and implementations with billiards. In *PPSN-BTP workshop.*

Goldfarb, D. (1970). A family of variable-metric algorithms derived by variational means. mathematics of computation, 24:23-26.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaption. In *Proc. of the IEEE Conference on Evolutionary Computation (CEC 1996)*, pages 312–317. IEEE Press.

Hickernell, F. J. (1998). A generalized discrepancy and quadrature error bound. *Mathematics of Computation*, 67(221):299–322.

Hooke, R. and Jeeves, T. A. (1961). Direct search solution of numerical and statistical problems. *Journal of the ACM, Vol. 8, pp. 212-229.*

Jin, Y. and Branke, J. (2005). Evolutionary optimization in uncertain environments. a survey, ieee transactions on evolutionary computation, vol. 9, no. 3, pp. 303 317.

Kaupe, A. F. (1963). Algorithm 178: direct search. *Commun. ACM*, 6(6):313–314.

Keijzer, M., Merelo, J. J., Romero, G., and Schoenauer, M. (2001). Evolving objects: A general purpose evolutionary computation library. In *Artificial Evolution*, pages 231–244.

LaValle, S. M., Branicky, M. S., and Lindemann, S. R. (2004). On the relationship between classical grid search and probabilistic roadmaps. *I. J. Robotic Res.*, 23(7-8):673–692.

L'Ecuyer, P. and Lemieux, C. (2002). Recent advances in randomized quasi-monte carlo methods. pages 419–474.

Lindemann, S. R. and LaValle, S. M. (2003). Incremental low-discrepancy lattice methods for motion planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2920–2927.

Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM.

Owen, A. (2003). *Quasi-Monte Carlo Sampling, A Chapter on QMC for a SIGGRAPH 2003 course*.

Sendhoff, B., Beyer, H.-G., and Olhofer, M. (2004). The influence of stochastic quality functions on evolutionary search, in recent advances in simulated evolution and learning, ser. advances in natural computation, k. tan, m. lim, x. yao, and l. wang, eds. world scientific, pp 152-172.

Shanno, D. F. (1970.). Conditioning of quasi-newton methods for function minimization. mathematics of computation, 24:647-656.

Sloan, I. and Woźniakowski, H. (1998). When are quasi-Monte Carlo algorithms efficient for high dimensional integrals? *Journal of Complexity*, 14(1):1–33.

Sutton, R. and Barto, A. (1998). *Reinforcement learning: An introduction.* MIT Press., Cambridge, MA.

Tsutsui, S. (1999). A comparative study on the effects of adding perturbations to phenotypic parameters in genetic algorithms with a robust solution searching scheme, in proceedings of the 1999 ieee system, man, and cybernetics conference smc 99, vol. 3. ieee, pp. 585 591.

Tuffin, B. (1996). On the use of low discrepancy sequences in monte carlo methods. In *Technical Report 1060, I.R.I.S.A.*

Wasilkowski, G. and Wozniakowski, H. (1997). The exponent of discrepancy is at most 1.4778. *Math. Comp*, 66:1125–1132.

Wright, M. (1995). Direct search methods: Once scorned, now respectable. *Numerical Analysis (D. F. Griffiths and G. A. Watson, eds.), Pitman Research Notes in Mathematics*, pages 191–208. http://citeseer.ist.psu.edu/wright95direct.html.

Zhu, C., Byrd, R., P.Lu, and Nocedal, J. (1994). L-BFGS-B: a limited memory FORTRAN code for solving bound constrained optimization problems. *Technical Report, EECS Department, Northwestern University.*

Table 1: Summary of the characteristics of the benchmarks. The stock management problems theoretically lead to convex Bellman-functions, but their learnt counterparts are not convex. The "arm" and "away" problem deal with robot-hand-control; these two problems can be handled approximately (but not exactly) by bang-bang solutions. Walls and Multi-Agent problems are motion-control problems with hard penalties when hitting boundaries; the loss functions are very unsmooth.

Name	Nb of time steps	State space dimension (basic case)	Nb scenarios	Action space dimension (basic case)
Stock Management	30	4	9	4
Stock Management V2	30	4	9	4
Fast obstacle avoidance	20	2	0	1
Arm	30	3	50	3
Walls	20	2	0	1
Multi-agent	20	8	0	4
Away	40	2	2	2

Table 2: Experimental results. All stds are available at `http://www.lri.fr/~teytaud/sefordplong.pdf`. For the "best algorithm" column, **bold** indicates 5% significance for the comparison with all other algorithms and *italic* indicates 5% significance for the comparison with all but one other algorithms. **y** holds for 10%-significance. Detailed results in `http://www.lri.fr/~teytaud/sefordplong.pdf` show that many comparisons are significant for larger families of algorithms, e.g. if we group GA and GANM, or if we compare algorithms pairwise. Problems with a star are problems for which bang-bang solutions are intuitively appealing; LD, which over-samples the frontiers, is a natural candidate for such problems. Problems with two stars are problems for which strongly discontinuous penalties can occur; the first meaning of robustness discussed in section 1.1 is fully relevant for these problems. Conclusions: 1. GA outperforms random and often QR. 2. For *-problems with nearly bang-bang solutions, LD is significantly better than random and QR in all but one case, and it is the best in 7 on 8 problems. It's also in some cases the worst of all the tested techniques, and it outperforms random less often than QR or GA. LD therefore appears as a natural efficient tool for generating nearly bang-bang solutions. 3. In **-problems, GA and GANM are often the two best tools, with strong statistical significance; their robustness for various meanings cited in section 1.1 make them robust solutions for solving non-convex and very unsmooth problems with ADP. 4. Stock management problems (the two first problems) are very efficiently solved by CMA-ES, which is a good compromise between robustness and high-dimensional-efficiency, as soon as dimensionality increases.

Problem	Dim.	Best algo.	QR beats random	GA beats random ; QR	LBFGSBrestart beats random;QR	LD beats random;QR
Stock	4	**LDfff**	y	y;n	y ; n	y ; n
and	8	**EoCMA**	**y**	n;n	n ; n	n ; n
Demand	12	*EoCMA*	**y**	n;n	n ; n	n ; n
	16	**EoCMA**	n	n;n	n ; n	n ; n
Stock and	4	**LD**	**y**	**y**;y	**y**; **y**	**y**; **y**
Demand2	8	EoCMA	n	y;**y**	n ; **y**	y ; **y**
Avoidance	1	**HJ**	y	y;n	n ; n	**y**; y
Walls**	1	GA	y	y;y	y ; y	y ; y
Multi-agent**	4	**GA**	n	**y**;**y**	n ;n	n ; n
	8	**GANM**	y	**y**;**y**	n ;n	n ; n
	12	**LDfff**	**y**	**y**;**y**	n ;n	n ; n
	16	*GANM*	**y**	**y**;**y**	n ;n	y ; n
Arm*	3	LD	y	**y**;y	y ; y	**y**; **y**
	6	HJ	**y**	**y**;y	y ; y	**y**; **y**
	9	LD	**y**	**y**;n	y ; y	**y**; **y**
	12	**LD**	**y**	**y**;**y**	y ; y	**y**; **y**
Away*	2	LD	y	y;y	y ; n	**y**; y
	4	LD	y	**y**;y	y ; y	**y**; **y**
	6	LD	y	y;y	y ; y	**y**; **y**
	8	**LD**	y	y;y	y ; y	**y**; **y**
Total			17/20	17/20 ; 14/20	11/20 ; 10/20	14/20 ; 12/20