A Vector Quantization Approach to Scenario Generation for Stochastic NMPC

C. Goodwin, Graham; Østergaard, Jan; E. Quevedo, Daniel; Feuer, Arie

Published in:
Lecture Notes in Computer Science

DOI (link to publication from Publisher):
10.1007/978-3-642-01094-1_19

Publication date:
2009

Document Version
Early version, also known as pre-print

A Vector Quantization Approach to Scenario Generation for Stochastic NMPC

Graham C. Goodwin[†], Jan Østergaard[‡], and Daniel E. Quevedo[⋆]

The University of Newcastle,
School of Electrical Engineering and Computer Science,
New South Wales, Australia
[†] graham.goodwin@newcastle.edu.au
[‡] jan.ostergaard@newcastle.edu.au
[⋆] dquevedo@ieee.org

Keywords : Scenario generation, closed loop control, stochastic nonlinear model predictive control, vector quantization

Abstract : This paper describes a novel technique for scenario generation aimed at closed loop stochastic nonlinear model predictive control. The key ingredient in the algorithm is the use of vector quantization methods. We also show how one can impose a tree structure on the resulting scenarios. Finally, we briefly describe how the scenarios can be used in large scale stochastic nonlinear model predictive control problems and we illustrate by a specific problem related to optimal mine planning.

1 Introduction

The motivation for the research described in the current paper arises from large scale optimization problems having a temporal component. A specific example of such a problem is open-cut mine planning. In this example, the goal is to determine the value of an asset by carrying out an optimization of possible future actions over a suitable planning horizon (typically 20 years for a mine). Such problems can be converted into nonlinear model predictive control problems by appropriate choice of variables. An important feature of such problems is that they contain a large number of inputs and states. Indeed, a simplified version of the mine optimization problem involves tens of thousands of state variables. Hence, even after the application of spatial and temporal aggregation, it typically takes many hours to carry out the required optimization on a high speed computer. Another important feature of such problems is that there are usually variables whose future values cannot be accurately predicted. For example, in the case of mining, one does not know the future price that the ore will bring. Hence it is desirable to treat such problems in a stochastic setting. Alas, the issue of computational complexity now becomes critical. Given that the problem may take several hours to solve for one particular realization of the random variables, then one is necessarily restricted to use a handful (say 100) "scenarios" to represent the possible future values of the uncertain processes when carrying out closed loop optimization. With the above as background, the current paper addresses the problem of scenario generation for complex stochastic optimization problems.

The specific motivation for the current work arises from the problem of open-cut mine planning. This problem typically has a planning horizon of the order of 20 years.

A plan based on nominal values for the uncertain processes can take many hours on a high speed computer. Interestingly, this speed of solution is achieved only after several simplifying steps have been made [].

As can be readily appreciated, the values of certain key variables are not precisely known into the future. In such cases, it makes sense to use a stochastic problem formulation. Indeed, it could be argued that such a formulation should, in principle, be used for all model predictive control (MPC) problems since, in practice, future disturbances will always have a non-predictable element. However, for many problems, taking a formal stochastic approach is unwarranted since the gains achieved from such an approach can be quite small. Hence, the usual MPC paradigm used in industrial control is the so called "receding horizon" approach. Here one typically uses open loop optimization to determine the control sequence over some horizon and then one applies the first control action. At the next time instant, one measures (or estimates) the state and then recomputes the control over a future control horizon and the first control step is again implemented. This strategy is very well known to the control community and has been extremely successful in practice []. This kind of strategy "reacts" to disturbances when they occur (since the input is recalculated based on the measured state). However, no explicit account is taken of the fact that, in the future, we will have more information about the uncertain states than we do at the present time. Control policies which implement the latter policy are usually termed "closed loop".

There exists a substantial literature[1] on "closed loop" optimization in the stochastic programming literature []. There has also been some interest in the topic in recent control literature. For example, Muñoz de la Peña, Bemporad and Alamo [1] consider closed loop policies based on the vertices of an assumed set.

Our particular interest in the current paper resides in cases where the state dimension is very large and the underlying system is highly nonlinear. Clearly, in such a problem one needs to be extremely careful with stochastic optimization since the associated computations can easily become intractable. The end result of this line of reasoning is that one can, at best, deal with a "handful" (say several hundred) possible realizations of the uncertain elements in the problem. This, in turn, raises the issue of how one should choose this "handful" of realizations (which we term "scenarios") so they give representative "coverage" of the likely outcomes. To illustrate the difficulty of this problem, we note that if we utilize an optimization horizon of 20 steps and we consider just 10 values for the uncertain variables at each step then this gives 10^{20} realizations of the uncertain process. Since in non-convex stochastic optimization the computational time grows linearly with the number of scenarios, then if it takes several hours to deal with one realization, then clearly 10^{20} realizations is completely impossible. (It would take 10^{17} years!).

The topic of scenario generation has been addressed in the stochastic programming literature []. Indeed, there exists a substantial literature devoted to this topic. One obvious recommendation made in the literature is that one should ideally design the scenarios taking into account the "true problem". However, this is not sensible in the context of complex problems since it is computational intractable to compare different scenario patterns via a Monte-Carlo study. Indeed, if this were possible then one could simply use a Monte-Carlo study to carry out the intended design.

Our strategy will be to divide the problem into two stages. In the first stage, we will carry out scenario design based on a simple measure of scenario performance. In this stage, we rely upon the fact that the number of uncertain variables is typically small

[1] The policies are sometimes said to be "with recourse" in the stochastic programming literature.

(say 2 or 3 in the case of the mining problem). Then, in a second stage, we will utilize the scenarios on the "true problem". This "divide-and-conquer" strategy is aimed at making the overall problem computationally tractable.

The novel contribution in the current paper is to link the problem of scenario generation to code book design in vector quantization. This link allows us to develop a new strategy for scenario generation. We also explain several embellishments of the basic scheme including how to enforce a tree structure on the scenarios. The latter is used in the context of closed loop stochastic control.

The layout of the remainder of the paper is as follows: In section 2, we give a brief overview of the mine planning problem so as to place the subsequent work in a practical context. In section 3 we briefly review different stochastic optimization strategies. Section 4 contains the key result of the paper, namely, the scenario generation algorithm. In section 5 we briefly return to the mine planning problem and conclusions are given in section 6.

2 Motivational Problem

Before describing the scenario generation strategy, we will first set the work in a practical context by briefly describing the optimal mine planning problem.

The key idea is as follows: given geological data based on preliminary exploration, determine where and when to dig. The optimization problem can be cast as a mixed integer linear programming (MILP) problem. A host of constraints need to be satisfied e.g. mining capacity in each year, slope constraints on the walls of the mine, precedent constraints on the order in which material is removed, processing plant constraints etc.

If one adopts the, so called, block model approach, then one divides the mine into blocks say 100×100 on the surface and 10 vertically. This gives 10^5 blocks. Over a 15 year horizon, this gives 10^{75} decisions on when to remove a block. Interestingly, 10^{75} is approximately the number of atoms in the known universe, so clearly some simplifications are necessary.

The basic problem can be given a nice interpretation in the NMPC framework. To see how this can be done, we divide the surface into rectangular blocks $\{j = 1, \ldots, M; k = 1, \ldots, M\}$ as shown in Fig. 1.

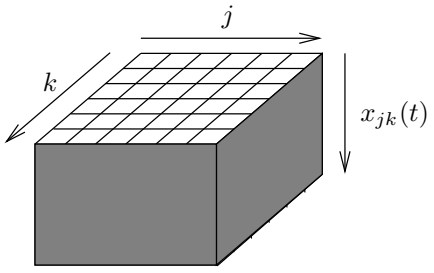


Figure 1: Block model of a mine.

Let $x_{jk}(t)$ denote the depth at location j, k at time t, $u_{jk}(t) \in \{0, 1\}$ denote the decision to mine (1) or not to mine (0) at time t. Then a simple state space model is

$$x_{jk}(t) = x_{jk}(t-1) + bu_{jk}(t). \tag{1}$$

The various constraints take the form

$$\sum_{jk} h^l_{jk} x_{jk} \leq b_l. \tag{2}$$

The cost function can be expressed as

$$J = \sum_{t=1}^{N} d_t c_t \sum_{j,k} V_{j,k}(x_{jk}(t)) u_{j,k} \tag{3}$$

where $V_{jk}(x_{jk}(t))$ denotes the amount of ore at depth $x_{jk}(t)$ in location (j,k), d_t denotes a discount factor and c_t denotes the value of ore at time t.

3 Stochastic Optimization Strategies

The simplified description of the mine planning problem given above implicitly assumes that the value of the ore is known. However, future values of this variable are certainly not exactly known. Several strategies can be adopted to deal with this uncertainty as described below:

3.1 Open Loop Policies

Here one carries out the design based on some nominal trajectory (say the expected value) for the uncertain variables. Then one applies the strategy irrespective of what actually happens. This may sound rather strange to the control community but, in mine planning, certain decisions (e.g. how large to make the processing plant) cannot easily be changed in the light of updated information.

3.2 Receding Horizon (or Reactive) Policies

Here one bases the original design on some nominal trajectory for the uncertain variables. However, one only implements the first stage. One then redoes the optimization when new information is obtained (i.e. one "reacts" to incoming data). This idea is central to model predictive control and will be very familiar to the control community.

3.3 Closed Loop Policies

These policies take account of the fact that, in the future, we will have additional information not available now. Closed loop policies typically lead to function optimization problems in which one designs a mapping from the future information state to the control. We give a brief overview of the dynamic programming (DP) approach to these policies.

Let I_k denote the information available to the controller at time k, that is

$$I_k = (y_0, \ldots, y_k, u_0, \ldots, u_{k-1}). \tag{4}$$

The required control policy $\pi(\mu_0, \ldots, \mu_{N-1})$ maps I_k into the control space c_k. A key feature is the non-anticipatory constraint i.e. decisions can only be based on the information that has been revealed so far.

The cost function takes the form:

$$J_\pi = \mathop{\mathbb{E}}_{z_0,\{\omega_k\},\{\nu_k\}} \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(I_k), \omega_k) \right\} \tag{5}$$

where the state evolution satisfy

$$x_{k+1} = f_k(x_k, \mu_k(I_k), \omega_k). \tag{6}$$

The available measurements are

$$y_0 = h_0(x_0, \nu_0) \tag{7}$$

$$y_k = h_k(x_k, \mu_{k-1}(I_{k-1}), \nu_k) \tag{8}$$

where $\{\omega_k\}, \{\nu_k\}$ are i.i.d. sequences (typically Gaussian distributed).

The associated DP equations are

$$J_{N-1}(I_{N-1}) = \min_{u_{N-1}\in\mathcal{U}_{N-1}} \left\{ \mathop{\mathbb{E}}_{x_{N-1},\omega_{N-1}} \Big[g_N(f_{N-1}(x_{N-1}, u_{N-1}, \omega_{N-1})) + g_{N-1}(x_{N-1}, u_{N-1}, \omega_{N-1}) | I_{N-1}, \mu_{N-1} \Big] \right\} \tag{9}$$

and for $k = 0, \ldots, N-2$

$$J_k(I_k) = \min_{u_k\in\mathcal{U}_k} \left\{ \mathop{\mathbb{E}}_{x_k,\omega_k,y_{k+1}} \Big[g_k(x_k, u_k, \omega_k) + J_{k+1}(I_k, y_{k+1}, u_k) | I_k, \mu_k \Big] \right\}. \tag{10}$$

3.4 A Simple Example

To illustrate closed loop planning, we consider the simple two-stage stochastic decision problem in Fig. 2. We see in this figure that there is only one random variable, ω, which takes one of two values, ω_1, ω_2 with equal probability. There are two stages in the problem and two decisions for the control at each stage. Thus, at stage 1, u_0 can be chosen as a or b and at stage 2, u_1 can be chosen as a or b. The final rewards (cost function) are shown on the right of the diagram.

Optimal open loop and reactive policies do not use the fact that the state will be known after stage 1 has been completed. Thus open loop and reactive policies both lead to the same return of \$2. This can be seen from the following simple argument:

Say we apply $u_0 = a$; then whatever we do next gives $\pm\$50,000$ with equal probability. Hence, the expected return is \$0. However, $u_0 = b, u_1 = a$ returns \$1 and $u_0 = b, u_1 = b$ returns \$2. Thus, in conclusion, the best open loop strategy is $u_0 = b, u_1 = b$ yielding \$2 (we would get the same answer with a reactive policy).

For the closed loop case, we add the extra information that we will know where we have reached at the end of stage 1. The obvious closed loop policy is; $u_0 = a$, then $u_1 = a$ if $\omega = \omega_1$ which implies a return of $\$50,000$ and $u_0 = a$, then $u_1 = b$ if $\omega = \omega_2$ which implies a return of $\$50,000$.

We see from the above that a closed loop strategy can give significant benefits compared with open loop or reactive. At a heuristic level the closed loop policy keeps "all options open" and avoids being "painted into a corner" by the first move.

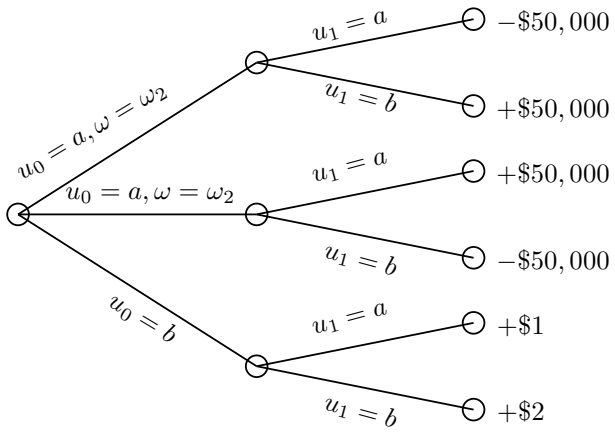


Figure 2: Simple example.

3.5 Computational Issues

The computational burden associated with the design of closed loop policies grows linearly with the number of alternatives considered for the uncertain variables. Hence, it is usually essential to restrict the cardinality of the set of alternatives for the uncertain variables. The issue of how to choose the representative set of alternatives is addressed in the next section.

4 Scenario Generation

The goal of scenario generation is to come up with a (relatively small) set of representatives trajectories for a stochastic process. A specific example is that of ore prices as described in section 2.

A "brute force" method is to use Monte Carlo type methods to simulate a set of trajectories using different "seeds" for the underlying innovation process. This is known to perform well for a large number of scenarios []. However, in the case where the cardinality of the scenario set is severely restricted, then it is prudent to exercise some care in the scenario selection.

In the next subsection we describe the key novel contribution of this paper, namely, linking the problem of scenario generation to vector quantization.

4.1 Vector Quantization

Let us first briefly review some important properties of vector quantization (VQ). For a thorough introduction to VQ we refer the reader to [2, 3].

An L-dimensional vector quantizer Q_L is a (nonlinear and non-invertible) map:

$$Q_L : \mathbb{R}^L \to \mathcal{C} \tag{11}$$

where $\mathcal{C}$ is a discrete set of M distinct elements given by

$$\mathcal{C} \triangleq \{c^i \in \mathbb{R}^L : i = 1, \ldots, M\}. \tag{12}$$

The set $\mathcal{C}$ is also known as a codebook and the element $c^i \in \mathcal{C}$ is usually referred to as the ith codeword or ith reconstruction point.

It is convenient to decompose Q_L into a cascade of two functions, e.g. $Q_L(\cdot) = \beta(\alpha(\cdot))$ where α is referred to as the encoder and β is the decoder. The encoder is a many-to-one function which maps points in $\mathbb{R}^L$ to indices, i.e.

$$\alpha : \mathbb{R}^L \to \mathcal{I} \tag{13}$$

where $\mathcal{I}$ is an index set defined as

$$\mathcal{I} \triangleq \{i \in \mathbb{N} : i = 1, \ldots, M\}. \tag{14}$$

We then define $\alpha(x) \triangleq i$ if and only if $x \in S^i$ where $S^i \in \mathcal{S}$ and where

$$\mathcal{S} \triangleq \{S^i \subset \mathbb{R}^L : i = 1, \ldots, M\}. \tag{15}$$

We generally require that $\mathcal{S}$ "cover" $\mathbb{R}^L$, i.e. $\mathbb{R}^L \subseteq \mathcal{S}$ and moreover that any pair of subsets $(S^i, S^j), i \neq j$, do not overlap except possibly at their boundaries.

The decoder is given by

$$\beta : \mathcal{I} \to \mathcal{C} \tag{16}$$

where $\mathcal{I}$ is given by (14) and $\mathcal{C}$ by (12). With this, we establish the following chain of equivalence:

$$Q_L : x \mapsto c^i \Leftrightarrow x \in S^i \Leftrightarrow \alpha(x) = i, \beta(i) = c^i \quad \text{so that} \quad Q_L(x) = \beta(\alpha(x)). \tag{17}$$

Given a distortion measure (or cost function) $\rho : \mathbb{R}^L \times \mathcal{C} \to \mathbb{R}^+$ we define a Voronoi cell V^i :

$$V^i \triangleq \{x \in \mathbb{R}^L : \rho(x, c^i) \leq \rho(x, c^j), \quad j = 1, \ldots, M\}, \quad i = 1, \ldots, M. \tag{18}$$

It follows that if Q_L is a *nearest neighbor* quantizer, then $S^i = V^i$ and this is in fact an optimal encoder for the given decoder, i.e. for the given set of codewords $\mathcal{C}$ [2].

Let $\phi_X(x)$ denote the probability density function for the random variable X. Then, an optimal quantizer is one that, for a given M, minimizes the expected cost J where

$$J = \mathbb{E}\rho(X, Q_L(X)) \tag{19}$$

$$= \sum_{i=1}^{M} \int_{x \in S^i} \phi_X(x) \rho(x, c^i)\, dx \tag{20}$$

$$= \sum_{i=1}^{M} P(X \in S^i) \mathbb{E}[\rho(x, c^i) | X \in S^i]. \tag{21}$$

In simple cases, the codeword c^i only appears in one of the terms of the sum in (21). It then follows that the optimal codeword, given the set S^i, is the generalized centroid of S^i. Specifically, given $S^i \in \mathcal{S}$

$$\hat{c}^i = \arg \min_{c^i \in \mathbb{R}^L} \mathbb{E}\left[\rho(X, c^i) | X \in S^i\right] \tag{22}$$

$$= \arg \min_{c^i \in \mathbb{R}^L} \frac{\int_{x \in S_i} \phi_X(x) \rho(x, c)\, dx}{\int_{x \in S_i} \phi_X(x)\, dx}. \tag{23}$$

In other words, given the encoder, or equivalently, given the set $\mathcal{S}$, the optimal decoder is defined by the set of reconstructions points $\mathcal{C} = \{\hat{c}^i : i = 1, \ldots, M\}$, where $\hat{c}^i$ is given by (22).

An optimal quantizer is therefore a nearest neighbor quantizer having centroids as codewords [2]. For example, if ρ is the squared error distortion measure, i.e. $\rho(x, x') = \|x - x'\|^2 = \sum_{n=0}^{L-1} |x_n - x'_n|^2$ then it is easy to show that

$$\hat{c}^i = \arg \min_{c^i \in \mathbb{R}^L} \mathbb{E}\left[\rho(X, c^i) | X \in S^i\right] \tag{24}$$

$$= \mathbb{E}\left[X | X \in S^i\right] \tag{25}$$

$$= \frac{\int_{x \in S_i} \phi_X(x) x \, dx}{\int_{x \in S_i} \phi_X(x) \, dx}. \tag{26}$$

Furthermore, if X is stationary and ergodic, then one can approximate the centroid by the sample average obtained simply by drawing a large number of points from S^i and taking their average [2].

Unfortunately, it is generally hard to design a jointly optimal encoder and decoder pair $(\alpha(\cdot), \beta(\cdot))$. However, there exist iterative design algorithms which yield locally optimal quantizers. One such algorithm is Lloyd's algorithm, which was originally defined for the scalar case [6, 7] and later extended to the vector case [5].

Lloyd's algorithm (and its extension to the vector case) is basically a cyclic minimizer that alternates between two stages; given an optimal encoder α, find the optimal decoder β and given an optimal decoder find an optimal encoder. More specifically, we first construct a random set of codewords $\mathcal{C}$. Then we repeatedly apply the following two steps:

1. Given a set of centroids $\mathcal{C} = \{c^i\}_{i=1}^M$, find the Voronoi cells $\mathcal{S} = \{S^i\}_{i=1}^M$ by use of (18).
2. Given a set of decision cells $\mathcal{S} = \{S^i\}_{i=0}^{M-1}$ find the centroids $\mathcal{C} = \{c^i\}_{i=1}^M$ by use of (22).

This approach guarantees convergence to a (local) minimum [4].

4.2 Scenario Generation by Vector Quantization Techniques

We will now establish a connection between the extended Lloyd's VQ design algorithm and scenario generation in stochastic MPC.

Let $z_k \in \mathbb{R}^L$ be a state vector that satisfies the Markovian recursion given by

$$z_{k+1} = f(z_k, \omega_k) \tag{27}$$

where $\omega_k \in \mathbb{R}^L$ is an arbitrary distributed random vector process.

In the special case where $\omega_k \in \{w_k(0), \omega_k(1)\}$, i.e. the disturbance, can take on only two distinct values at every time instant k, then the evolving state sequence describes a binary tree as shown in Fig. 3. The root of the tree describes the initial state z_0 at time $k = 0$. Then, at time $k = 1$, the next state, i.e. z_1 will take on the value $z_1(0)$ or $z_1(1)$ depending upon whether the event $\omega_0(0)$ or $\omega_0(1)$ happens. At time $k = 2$, if the previous state was $z_1(0)$, the current state will be either $z_2(0)$ or $z_2(1)$. Similarly, if the previous state was $z_1(1)$ then the current state will be either $z_2(2)$ or $z_2(3)$ depending on the actual realization of the uncertain disturbance ω_k. Thus, four

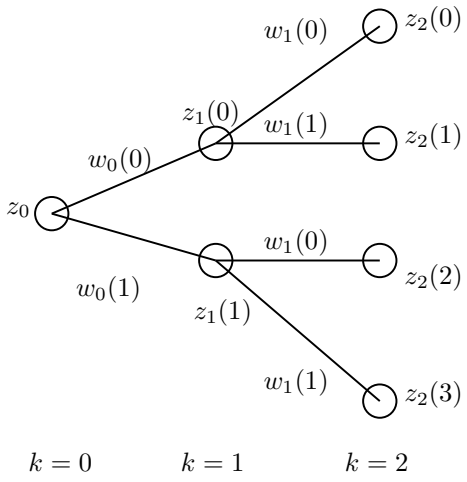


Figure 3: Binary tree.

different state trajectories are possible and at time $k = 0$ it is not known in advance which one of them will eventually happen. The only information that is available at time $k = 0$ is the statistics, i.e. the probability of each of the trajectories. Notice that we can describe each trajectory, i.e. each path in the tree, by a sequence of disturbances. Specifically, the ith path can be described by the sequence $\omega^i = (\omega_0(i), \omega_1(i))$, where $i \in \{0, \ldots, 3\}$. In the general case where we have $N + 1$ stages in the tree and M distinct end nodes $\{z_N(i)\}, i = 1, \ldots, M$, which (in the case of the binary tree) corresponds to M distinct paths in the tree, we have $\omega^i = (\omega_0(i), \omega_1(i), \ldots, \omega_{N-1}(i))$. Furthermore, there is a one-to-one correspondence between the sequence of disturbances ω^i and the sequence of state vectors $z^i = (z_0(i), z_1(i), \ldots, z_N(i))$. To see this, recall that $z_{k+1}(i) = f_k(z_k(i), \omega_k(i))$.

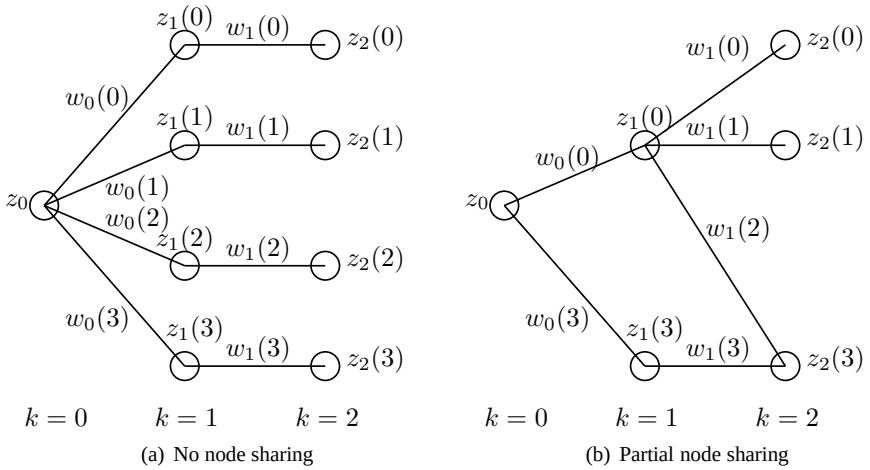
With the above, we will refer to a sequence of disturbances, say ω^i, as a scenario. In particular, ω^i denotes the ith scenario. We are interested in scenario generation for finite-horizon stochastic MPC. If the disturbances takes on only a finite number of possible values at each time instant, then we can form a scenario tree, e.g. as the one shown in Fig. 3. Of course, many other trees are possible, cf. Figs. 4(a) and 4(b).

It is often the case that the disturbances take on a continuum of values. In this case, we seek to form a finite number of scenarios by discretizing the set of possible disturbances. Specifically, we wish to design M distinct scenarios, whose trajectories capture the evolution of the the most likely state sequences. In other words, the set of M candidate scenarios should (on average) be a good approximation of all possible sequences of disturbances. Thus, we actually wish to design a codebook $\mathcal{C}$ in the ω-space having M codewords where the codewords $\{\omega^i \in \mathcal{C}\}_{i=1}^M$ are themselves scenarios.

Let J_N be the N-horizon cost function defined by

$$J_N \triangleq \mathbb{E} \min_{\omega^i \in \mathcal{C}} \rho_N(z, z^i) \tag{28}$$

$$= \min_{\mathcal{S}} \sum_{i=1}^{M} P(\omega \in S^i) \mathbb{E}[\rho_N(z, z^i) | \omega \in S^i] \tag{29}$$



(a) No node sharing (b) Partial node sharing

Figure 4: Different scenario trees.

where

$$\hat{\omega}^i = \arg\min_{\omega^i \in \mathbb{R}^{L(N+1)}} \mathbb{E}[\rho_N(z, z^i) | \omega \in S^i] \tag{30}$$

is the generalized centroid of an optimal nearest neighbor quantizer, $S^i \subset \mathbb{R}^{L(N+1)}$ and $\rho_N(\cdot, \cdot)$ is given by

$$\rho_N(z, z^i) = \|z - z^i\|_Q^2 \tag{31}$$

$$= \sum_{k=0}^{N} \|z_k - z_k(i)\|_{Q_k}^2 \tag{32}$$

where $\{Q_k \in \mathbb{R}^{L \times L}\}, k = 0, \ldots, N$ is a sequence of weighting matrices and $Q = \text{diag}(Q_0, \ldots, Q_N)$.

Interestingly, this approach yields a jointly (locally) optimal distribution of codewords over the temporal as well as spatial dimensions. Note, however, that one has no control over the resulting structure of the scenario tree. In fact, a likely outcome is a scenario tree with a root node that branches into M separate *deterministic* paths (similar to Fig. 4(a) for the case of $M = 4$).

4.3 Imposing a Tree Structure on the Scenarios

In the previous section we allowed arbitrary scenario trees. Clearly, this yields the lowest possible cost. Nonetheless, in stochastic closed loop planning one requires that the future uncertainty be progressively reduced as the stages proceed. Thus, a tree like structure is required in the scenario space.

To enforce the tree structure one needs to ensure that nodes of the tree share common points. Let Υ_N^M denote the set of all possible tree structures containing exactly M distinct *paths* each having $N + 1$ nodes. For example, Figs. 3, 4(a), and 4(b), all belong to Υ_2^4. It should be clear that any codebook $\mathcal{C}$ having M codewords $\{\omega^i\}_{i=1}^M$ (each having N elements $\omega_k^i, k = 0, \ldots, N-1$) *admits* a tree $\Gamma \in \Upsilon_N^M$. We write $\mathcal{C} \triangleright \Gamma$ if $\mathcal{C}$ admits the specific scenario tree described by Γ.

When we restrict the codebook to admit a specific scenario tree, the codeword separation described in (30) does not apply and one needs to design the full codebook simultaneously. Specifically, given a scenario tree $\Gamma \in \Upsilon_N^M$ and a set of decision cells $\mathcal{S} = \{S^i \in \mathbb{R}^L : i = 1, \ldots, M\}$, the optimal set $\hat{C}$ of codewords must jointly satisfy:

$$\hat{\mathcal{C}} = \arg\min_{\mathcal{C} \triangleright \Gamma} \sum_{i=1}^{M} P(\omega \in S^i)\mathbb{E}[\rho_N(z, z^i)|\omega \in S^i] \tag{33}$$

where the minimization is now over discrete sets $\mathcal{C} \subset \mathbb{R}^{L(N+1)}$ satisfying $|\mathcal{C}| = M$ in addition to $\mathcal{C} \triangleright \Gamma$, which has the equivalent interpretation of minimizing over points in a higher dimensional vector space, i.e. $\mathcal{C} \in \mathbb{R}^{ML(N+1)}$ subject to $\mathcal{C} \triangleright \Gamma$.

We thus modify Lloyd's algorithm to alternate between updating the decision cells $\mathcal{S}$ using (18) keeping the codebook $\mathcal{C}$ fixed and updating the codebook using (33) keeping the decision cells fixed.

4.4 Example of Scenario Generation

To illustrate the principle behind the proposed scenario generation technique, we carry out a simple simulation. Let $z_{k+1} = 0.9z_k + \omega_k$ where $z_k, \omega_k \in \mathbb{R}$ and ω_k is a zero-mean unit-variance Gaussian distributed random variable. Let the horizon length be $N = 4$ and let the number of codewords be $M = 16$. It follows that we are interested in finding 16 "good" 4-dimensional codewords $\{\omega^i\}_{i=1}^4$ defined in the ω-space, which admit a specific scenario tree, say a binary tree. For simplicity, we will minimize the squared error in the state-space domain, i.e. $\rho_N(z, z^i) = \|z - z^i\|^2$.

We now first randomly pick a set of 16 codewords (from the distribution of ω) which admits a binary scenario tree. We then randomly draw $20,000$ 4-dimensional vectors (also from the distribution of ω) to be used as "training" vectors.[2] Finally, we alternate between numerically evaluating (18) and (33) given the training set. The resulting codebook and scenario tree after five iterations is illustrated in Figs. 5(a) and 5(b), respectively.

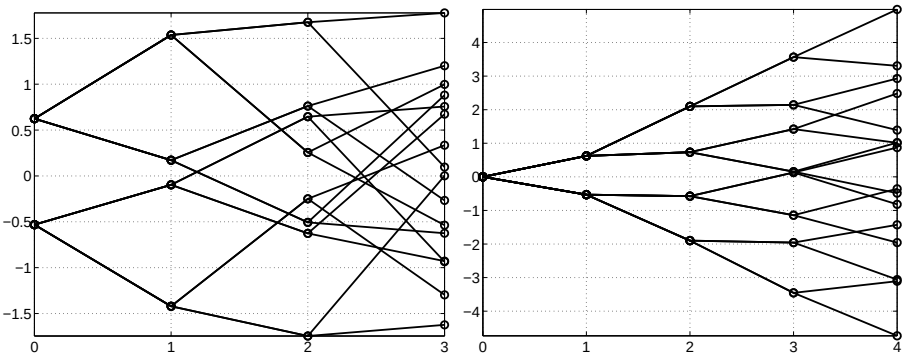
5 Return to the Motivational Problem

Finally, we return to the optimal mine planning example. We recall that the state for this problem has two decoupled components; namely the mine depth at various locations and the ore price. For simplicity we assume that the current ore price states can be measured.

We can set the problem up as a large problem in which we assign a different control action to each node of the scenario tree. This implicitly imposes a mapping from the measured state to the control input.

We utilize the scenario tree for ore price to evaluate the current control as a function of the measured state x_0. Now time is advanced 1 step to $k = 1$. Since we have descretized the scenario space, the measured value of x_1 will, with probability one, not coincide with any of the values used in evaluating the current input. Thus, the scenario tool is really only a computational device to allow us to compute the current control action μ_0 whilst accounting (in some way) for the fact that, in the future, we

[2] In the case of simple distributions in the ω-space, it might be possible to explicitly derive the associated distribution in the z-space. This is convenient, since it eliminates the need for "training" vectors.



(a) 16 (4-dimensional) codewords (b) The resulting binary scenario tree

Figure 5: The resulting codebook and scenario tree after five iterations of the modified Lloyd's algorithm.

will actually know more than we do now. In other words, exactly as in example 3.2, we utilize scenarios to ensure that the first step is made in the knowledge that more will be known in the future.

Of course, the fact that the measured value of x_1 is not exactly equal to any of the values used the calculation should not be of great concern to us. All we need to do is to react to the measured value of x_1 at time 1 and recalculate u_1 by the same procedure as was used to evaluate μ_0.

We call the above strategy a receding horizon closed loop policy.

6 Conclusion

This paper has described a novel approach to scenario generation aimed at complex nonlinear model predictive control problems. We have shown that the problem can be casted in the framework of codebook design for vector quantization. We have also shown how the method can be embellished in several ways, e.g. by imposing a tree structure on the scenarios.

References

[1] D. Muñoz de la Peña, A. Bemporad, and T. Alamo. Stochastic programming applied to model predictive control. In *in Proc. 44th IEEE Conf. on Decision and Control and European Control Conf.*, pages 1361 – 1366, Sevilla, Spain, 2005.

[2] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[3] R. M. Gray and D. Neuhoff. Quantization. *IEEE Trans. Inf. Theory*, 44(6):2325 – 2383, 1998.

[4] R.M. Gray, J.C. Kieffer, and Y. Linde. Locally optimal block quantizer design. *Information and Control*, 45:178 – 198, May 1980.

[5] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Commun.*, 28(1):84 – 95, January 1980.

[6] S. P. Lloyd. Least squares quantization in PCM. Unpublished Bell Laboratories technical note, 1957.

[7] S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, IT-28(2):127 – 135, March 1982.