Dynamic Programming

0.1 Motivation: Why Dynamics Matter

- Many economic problems involve *decisions over time*.
- In static optimization, we choose a single vector x to maximize $f(x)$ once.
- In dynamic optimization, we choose an entire sequence $\{x_t\}_{t=0}^{T}$ to maximize

$$\sum_{t=0}^{T} \beta^t u(s_t, x_t)$$

  subject to evolving constraints.

The key feature of a dynamic problem is **intertemporal dependence** today's decision affects tomorrow's feasible set.

Consider a groundwater extraction problem:

$$\max_{q_1, q_2} \ \pi_1(q_1) + \beta \pi_2(q_2)$$

subject to

$$s_2 = s_1 - q_1, \quad q_t \leq s_t, \quad s_t \geq 0.$$

It will not be optimal to leave any resource in the ground at the end, so $s_2 = 0$ and $q_2 = s_1 - q_1$. Substituting q_2 into the objective gives

$$\max_{q_1} \ \pi_1(q_1) + \beta \pi_2(s_1 - q_1).$$

The first-order conditions give

$$\pi_1'(q_1) - \beta \pi_2'(q_2) = 0,$$

which equates the **marginal benefit of current extraction** to the **discounted marginal benefit of future extraction**.

Examples

- Resource extraction (trade-off between current and future use)
  - Suppose a manager extracts a nonrenewable resource such as groundwater or oil. Each period, extraction reduces the remaining stock. The manager faces a trade-off between the *immediate profit* from extracting today and the *future value* of the remaining stock. Dynamic optimization determines the optimal extraction path balancing current and future gains — the or *shadow value* of the resource.
- Investment and capital accumulation
  - Firms decide how much to invest in physical capital (machinery, buildings) or human capital (training, education). Investment today increases the productive capacity tomorrow, affecting future profits. The firm must weigh the *cost of investment* against the *expected future returns*, leading to an optimal investment strategy over time.

- Crop rotations or livestock feeding
  - Farmers choose how to allocate land and resources over multiple periods. Planting a crop today may deplete soil nutrients, affecting yields in subsequent seasons. Similarly, feeding strategies for livestock impact their growth and productivity over time. Dynamic programming helps determine optimal crop rotations or feeding schedules that maximize long-term farm profitability.
- Forest rotation and timber harvesting
  - Forest managers decide when to harvest trees to maximize the value of timber while ensuring sustainability. Harvesting too early may yield lower quantities of wood, while waiting too long can lead to overgrowth where the marginal wood growth does not justify the cost. Dynamic optimization helps find the optimal rotation period that balances immediate revenue with future discounted value.
- Pollution control and climate policy
  - Governments and firms must decide how much to invest in pollution abatement technologies or carbon reduction strategies. Reducing emissions today can mitigate future environmental damage and associated costs. Dynamic programming helps evaluate the trade-offs between current abatement costs and long-term benefits, guiding optimal policy decisions over time.

0.2 Backward Induction

- In a *finite-horizon sequential decision problem*, one can solve by starting at the last period and then working backward.
- At the final period, the decision is trivial (or known) because nothing is left to optimize afterward.
- Then one moves to the second-to-last period: given that the future is solved optimally, choose your action now to maximize current payoff plus continuation value.
- This process continues backward until reaching the initial period. This is the logic of dynamic programming in finite-horizon settings.
- Backward induction is prefered over brute force enumeration because it simplifies potentially complex problems. Starting from the beginning, the number of paths can explode combinatorially, making it infeasible to evaluate all possible sequences of decisions.
- This was the intuition in the problem above: we used the constraint to eliminate q_2 and reduce the problem to a single decision variable q_1. Because there was no salvage value (no value to not using all of the resource at the end), we could assume that $q_2 = s_2 = s_1 - q_1$.
- In a finite-horizon problem, backward induction works because there is a last period: after that, no decisions remain. We can anchor the recursion at T and move backward.
- In an infinite-horizon problem, there is no terminal period. We need a different anchor. What we look for is a stationary path (or policy) that is internally consistent over time: i.e. at every date, given the current state, the decision you make will also be optimal for the continuation. In other words, you want a path or policy such that there is no incentive to deviate at any point — because the same 'value' structure recurs over time. That is what gives the stability and time-independence of the optimal policy in many infinite-horizon problems.

0.3 Stochastic Dynamic Programming

- In many problems, there is uncertainty about future states or payoffs.
- The state variable s_t may evolve according to a stochastic process, influenced by random shocks.
- The decision-maker must consider the expected value of future payoffs, integrating over possible future states.

0.3.1 Markov property

A problem is **Markovian** if the current state s_t contains all relevant information from the past. The future state depends only on the current state and decision.

Examples

- Stochastic resource growth model:
  Let fish stock evolve as
  $$s_{t+1} = G(s_t) + \varepsilon_{t+1} - h_t$$
  where ε_{t+1} is a random growth shock (possibly discrete). Then transition probabilities $P(s' \mid s, h)$ come from the distribution of ε. Note that s_{t+1} depends only on s_t and h_t, not on earlier states or actions.
- Crop yield shocks in agriculture:
  A farmer chooses input (fertilizer) x_t. The next soil state evolves as
  $$s_{t+1} = g(s_t, x_t)$$
  . The stochastic yield shock induces $P(s_{t+1} \mid s_t, x_t)$.
- Capital accumulation with productivity shocks:
  Suppose production is
  $$y_t = F(k_t, z_t)$$
  where z_t is a stochastic productivity state evolving as a Markov chain (discrete or discretized AR(1)). The decision is investment i_t, state vector is (k_t, z_t), and transitions are given by $\Pr(z_{t+1} \mid z_t)$ combined with deterministic $k_{t+1} = (1-\delta)k_t + i_t$.

0.4 Vocabulary and the components of dynamic programming problem

- **State variable** s_t: summarizes all relevant information at time t needed to make decisions and predict future states. Examples: resource stock, capital stock, productivity level.
- **Control variable** x_t: the decision made at time t. Examples: extraction quantity, investment amount, input usage.
- **Transition function** $s_{t+1} = g(s_t, x_t, \varepsilon_{t+1})$: describes how the state evolves from t to $t+1$ given current state, action, and random shocks.
- **Reward function** $u(s_t, x_t)$: the immediate payoff from taking action x_t in state s_t. Examples: profit, utility, yield.
- **Policy** $\{x_t(s_t)\}$: a rule that specifies the action to take in each state at each time.
- **Value function** $V_t(s_t)$: the maximum expected discounted sum of future rewards starting from state s_t at time t.
- **Discount factor** $\beta \in (0, 1)$: reflects the preference for current rewards over future rewards. It reflects time preference or impatience: how much less future utility / profit is worth relative to now. In growth / resource models, discounting ensures that long-run payoffs are "worth less" the farther they are in the future, which prevents over-emphasis on infinitely deferred rewards.
  - It also enables contraction mappings in infinite-horizon problems.
  - We often assume a constant discount factor for simplicity, but in reality, discounting may vary over time or depend on circumstances.
- **Salvage value**: the value of remaining resources or capital at the end of the planning horizon. In finite-horizon problems, this can be zero or some terminal value function.

0.5 Problem Types

1. Here is a finite time discrete state and time example:
   $$\max_{h_t} \left[\sum_{t=0}^{T-1} \beta^t \pi_t(h_t) + \beta^T V_T(s_T) \right]$$

subject to the simple deterministic transition dynamics:

$$s_{t+1} = s_t - h_t + G(s_t)$$

Identify the components of the problem

- State: s_t (can be either discrete or continuous)
- Control: h_t
- Transition: $s_{t+1} = s_t - h_t + G(s_t)$
- Reward: $\pi_t(h_t)$
- Policy: $h_t(s_t)$
- Value function: $V_t(s_t)$
- Discount factor: β
- Salvage value: $V_T(s_T) = 0$ or some terminal value

2. If time is continuous:

$$\max_{h(t)} \int_0^T e^{-\rho t}\, \pi(s(t), h(t))\, dt \;+\; e^{-\rho T}\, V_T(s(T))$$

subject to

$$\frac{ds}{dt} = G(s(t)) - h(t), \quad s(0) = s_0$$

The components are similar, but the transition is now a differential equation.

3. In an infinite-horizon problem:

$$\max_{\{h_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t \pi(h_t)$$

subject to

$$s_{t+1} = s_t - h_t + G(s_t), \quad s_0 \text{ given}$$

The components are the same, but there is no terminal time or salvage value. The problem is usually well-behaved when $\beta < 1$ and $\pi(h_t)$ is bounded.

---

1 Discrete time and state space

- Dynamic optimization problems describe sequential decision-making over time, where future states depend on current actions.
- In discrete time and discrete state settings, time proceeds in periods $t = 0, 1, \ldots, T$, and the system occupies one of a finite set of states $s_t \in S$ at each time t.
- The decision variable is denoted by $x_t \in X(s_t)$, representing the feasible choices when the system is in state s_t.

1.1 General Problem

- The general **finite-horizon** problem is to choose a sequence $\{x_t\}_{t=0}^{T-1}$ to maximize the expected discounted sum of single-period payoffs:

$$\max_{\{x_t\}_{t=0}^{T-1}} \sum_{t=0}^{T-1} \beta^t \pi(s_t, x_t) + \beta^T V_T(s_T)$$

where:

- $\beta \in (0, 1)$ is the discount factor,

- $\pi(s_t, x_t)$ is the single-period reward (or profit),
- $V_T(s_T)$ is the *terminal value function* (salvage value) at the end of the horizon.

The **transition dynamics** are deterministic:

$$s_{t+1} = f(s_t, x_t),$$

with an initial state s_0 given.

1.2 Bellman Equation

- The recursive formulation expresses the problem in terms of a **value function** $V_t(s_t)$ that gives the maximal attainable value starting from state s_t at time t:

$$V_t(s_t) = \max_{x_t \in X(s_t)} \Big\{ \pi(s_t, x_t) + \beta\, V_{t+1}(f(s_t, x_t)) \Big\}.$$

- The **terminal condition** closes the recursion:

$$V_T(s_T) = \pi_T(s_T) \quad \text{or a given terminal value function.}$$

The Bellman equation expresses the principle of optimality:

> The value of being in state s_t is the maximum over current actions of the immediate payoff plus the discounted value of the next state.

1.2.1 Backward Recursion (Finite Horizon)

To solve:

1. Start from the terminal period T, where $V_T(s_T)$ is known.
2. For each earlier period $t = T-1, \dots, 0$, compute $V_t(s_t)$ from the Bellman equation.
3. The corresponding optimal policy is

$$x_t^*(s_t) = \max_{x_t \in X(s_t)} \Big\{ \pi(s_t, x_t) + \beta\, V_{t+1}(f(s_t, x_t)) \Big\}.$$

This recursion yields both the value function and the optimal policy for each possible state.

1.2.2 Example Template

Once this structure is clear, we can apply it directly to the **mine management problem**, where:

- state: remaining ore s_t, and initial stock $s_0 = 4$,
- control: extraction $x_t \in \{0, 1, \dots, s_t\}$,
- transition: $s_{t+1} = s_t - x_t$,
- reward: $\pi(s_t, x_t) = 10x_t - (2x_t + x_t^2)$,
- terminal condition: $V_T(s_T) = 0$
- horizon: $T = 3$,
- discount factor: $\beta = 0.9$.

Students: set up the problem/Bellman equation and solve via backward induction.

We solve by **backward induction**:

$$V_t(s) = \max_{x \in \{0,\dots,s\}} \{\pi(x) + \beta V_{t+1}(s - x)\}, \quad V_3(s) \equiv 0.$$

1.2.3 Parameters (chosen for clean mental math)

- Horizon: $T = 3$ (decisions at $t = 0, 1, 2$, terminal at $t = 3$)
- Initial stock: $s_0 = 4$
- Discount factor: $\beta = 0.9$
- Price per unit: $p = 10$
- Cost: $c(x) = 2x + x^2$
- Per-period profit: $\pi(x) = px - c(x) = 10x - (2x + x^2) = 8x - x^2$

This gives the handy lookup: - $\pi(0) = 0$, $\pi(1) = 7$, $\pi(2) = 12$, $\pi(3) = 15$, $\pi(4) = 16$.

We solve by **backward induction**:

$$V_t(s) = \max_{x \in \{0,\dots,s\}} \{\pi(x) + \beta V_{t+1}(s - x)\}, \quad V_3(s) \equiv 0.$$

1.2.4 Step 1: Final decision period $t = 2$

Here $V_3(\cdot) = 0$, so extract to maximize $\pi(x)$ subject to $x \leq s$:

- $V_2(0) = 0$
- $V_2(1) = \pi(1) = 7$
- $V_2(2) = \pi(2) = 12$
- $V_2(3) = \pi(3) = 15$
- $V_2(4) = \pi(4) = 16$

Optimal action at $t = 2$ is $x_2^*(s) = s$ (extract all that remains).

1.2.5 Step 2: Period $t = 1$

Compute $V_1(s) = \max_{x \leq s}\{\pi(x) + \beta V_2(s - x)\}$:

- $s = 0$: only $x = 0 \Rightarrow V_1(0) = 0$
- $s = 1$:
  - $x = 0$: $0 + 0.9 \cdot V_2(1) = 6.3$
  - $x = 1$: $7 + 0.9 \cdot 0 = 7$
    $\Rightarrow V_1(1) = 7$ at $x = 1$
- $s = 2$:
  - $x = 0$: $0 + 0.9 \cdot 12 = 10.8$
  - $x = 1$: $7 + 0.9 \cdot 7 = 13.3$
  - $x = 2$: $12 + 0 = 12$
    $\Rightarrow V_1(2) = 13.3$ at $x = 1$
- $s = 3$:
  - $x = 0$: $0 + 0.9 \cdot 15 = 13.5$

- $x = 1$: $7 + 0.9 \cdot 12 = 17.8$
- $x = 2$: $12 + 0.9 \cdot 7 = 18.3$
- $x = 3$: $15 + 0 = 15$
  $\Rightarrow V_1(3) = 18.3$ at $x = 2$

- $s = 4$:
  - $x = 0$: $0 + 0.9 \cdot 16 = 14.4$
  - $x = 1$: $7 + 0.9 \cdot 15 = 20.5$
  - $x = 2$: $12 + 0.9 \cdot 12 = 22.8$
  - $x = 3$: $15 + 0.9 \cdot 7 = 21.3$
  - $x = 4$: $16 + 0 = 16$
    $\Rightarrow V_1(4) = 22.8$ at $x = 2$

Summary at $t = 1$:
$V_1(0) = 0,\ V_1(1) = 7,\ V_1(2) = 13.3,\ V_1(3) = 18.3,\ V_1(4) = 22.8.$

1.2.6 Step 3: Initial period $t = 0$ (with $s_0 = 4$)

Compute $V_0(4) = \max_{x \leq 4}\{\pi(x) + \beta V_1(4 - x)\}$:

- $x = 0$: $0 + 0.9 \cdot 22.8 = 20.52$
- $x = 1$: $7 + 0.9 \cdot 18.3 = 23.47$
- $x = 2$: $12 + 0.9 \cdot 13.3 = 23.97$
- $x = 3$: $15 + 0.9 \cdot 7 = 21.3$
- $x = 4$: $16 + 0 = 16$

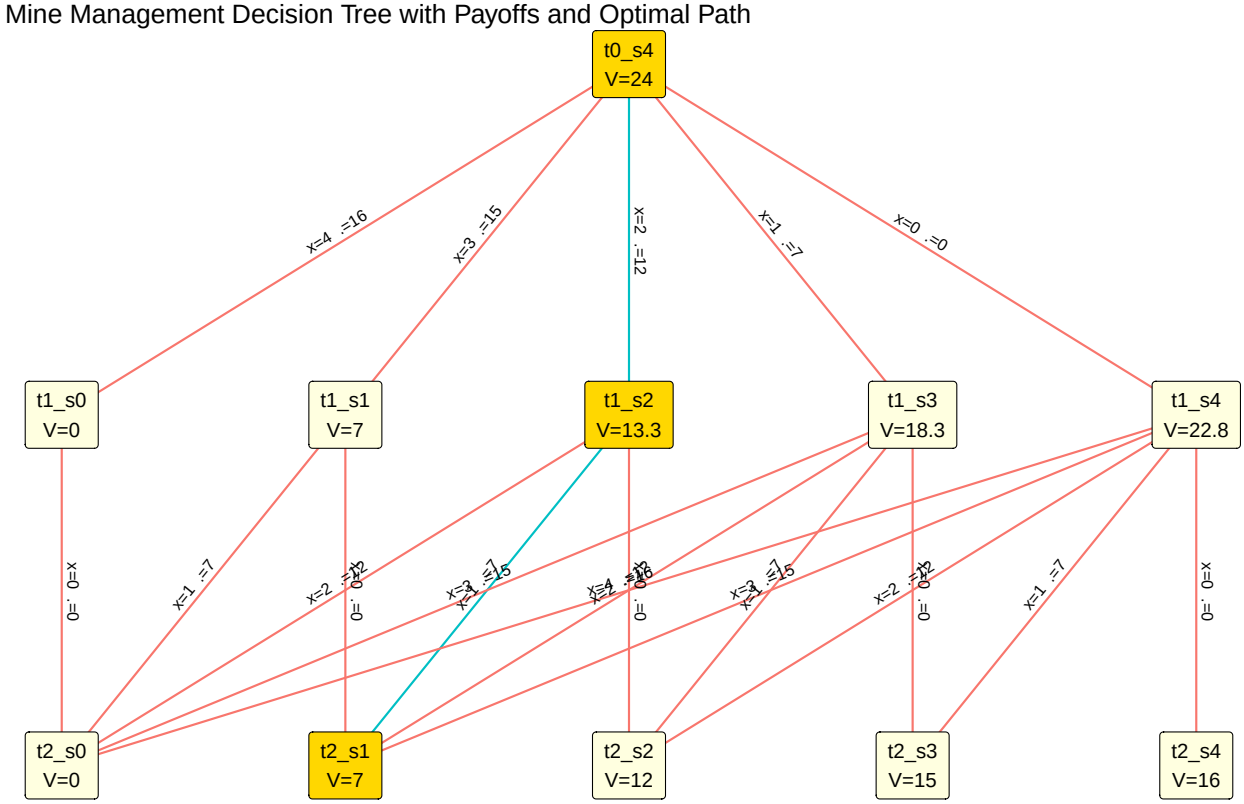
$\Rightarrow$ **Optimal** $x_0^*(4) = 2$ and $V_0(4) = 23.97$.

1.2.7 Optimal path and value

Starting at $s_0 = 4$: - $t = 0$: extract $x_0 = 2 \Rightarrow s_1 = 2$ - $t = 1$: with $s_1 = 2$, optimal $x_1 = 1 \Rightarrow s_2 = 1$ - $t = 2$: with $s_2 = 1$, extract all $x_2 = 1 \Rightarrow s_3 = 0$

Discounted payoff: - $t = 0$: $\pi(2) = 12$ - $t = 1$: $\beta\pi(1) = 0.9 \cdot 7 = 6.3$ - $t = 2$: $\beta^2\pi(1) = 0.81 \cdot 7 = 5.67$

Total $= 12 + 6.3 + 5.67 = 23.97 = V_0(4)$. Thus, the optimal extraction policy is to extract 2 units at $t = 0$, 1 unit at $t = 1$, and 1 unit at $t = 2$, yielding a total discounted profit of 23.97.



1.3 The Curse of Dimensionality

Dynamic programming offers an elegant recursive solution principle, but it suffers from a major practical limitation known as the **curse of dimensionality**.

The term was coined by Richard Bellman to describe the exponential growth in computational complexity as the number of state or control variables increases.

1.3.1 Illustration

In our mine management problem, the state s_t took on a small number of discrete values (e.g., $s_t \in \{0, 1, 2, 3, 4\}$).
If we introduce additional state variables — such as capital, technology, or another resource — the number of possible state combinations grows rapidly.

For example:

Dimension	Grid points per variable	Total grid points
1 state variable	100	100
2 state variables	100	10^4
3 state variables	100	10^6
4 state variables	100	10^8

Even modest discretizations quickly lead to millions of points. At each grid point, we must compute and compare values for all possible controls, making the full state–action space enormous.

Each additional state dimension multiplies the size of:

- The **value function table** $V_t(s)$

- The **policy function** $x_t^*(s)$
- The **transition mapping** $s_{t+1} = f(s_t, x_t)$
- And the computational effort of evaluating $\pi(s_t, x_t) + \beta V_{t+1}(f(s_t, x_t))$

This explosion makes naive enumeration infeasible.

For instance, if we allowed both **ore stock** and **equipment age** as states:

$$s_t = (s_t^{ore}, s_t^{equip}),$$

and if each dimension had only 10 discrete values, then V_t must be evaluated at $10 \times 10 = 100$ points per period. Add one more variable (say, capital), and it jumps to 1,000 points. For higher dimensions, this quickly exceeds computational limits.

The curse of dimensionality is not merely a computational nuisance—it shapes how economists model behavior.

In applied work, we rarely solve high-dimensional dynamic programs exactly. Instead, we rely on:

- **Approximations**: e.g., functional forms or interpolation of $V(s)$
- **Monte Carlo methods**: random sampling of states rather than full enumeration
- **Problem decomposition**: splitting the model into subproblems or exploiting structure (e.g., separability)
- **Aggregation or discretization**: grouping similar states or actions

These methods trade accuracy for tractability.

1.4 Introducing Stochastic Transitions: A Random Ore Price

So far, our mine problem was **deterministic**: the ore stock s_t evolved predictably according to

$$s_{t+1} = s_t - x_t,$$

and the price of ore was constant across periods.

Now suppose the **ore price fluctuates randomly** over time. We introduce a stochastic state variable p_t representing the market price.

1.4.1 The stochastic model

At each time t, the manager observes the current state (s_t, p_t) and chooses extraction x_t. Profits in period t are given by

$$\pi(s_t, p_t, x_t) = p_t x_t - c(x_t),$$

where $c(x_t)$ is the cost of extraction.

The state evolves according to:

$$s_{t+1} = s_t - x_t,$$
$$p_{t+1} \sim \Pr(p' \mid p_t),$$

where $\Pr(p' \mid p_t)$ describes the **transition probability** from the current price p_t to the next price p_{t+1}.

This is a **Markov process**: future prices depend only on the current price, not on the full past history.

1.4.2 Bellman equation under uncertainty

With stochastic prices, the decision problem becomes:

$$V_t(s_t, p_t) = \max_{x_t \in [0, s_t]} \Big[\pi(s_t, p_t, x_t) + \beta \, \mathbb{E}_{p_{t+1}|p_t} [V_{t+1}(s_t - x_t, p_{t+1})] \Big].$$

The expectation operator $\mathbb{E}_{p_{t+1}|p_t}$ averages future value across all possible next-period prices, weighted by their probabilities.

The terminal condition remains:

$$V_T(s_T, p_T) = 0.$$

1.4.3 Discrete-state example

To make this concrete, suppose the ore price can take **two values**:

Price state	Value	Label
Low	$p_L = 8$	state 1
High	$p_H = 12$	state 2

and transitions follow a **Markov matrix**:

$$P = \begin{pmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{pmatrix},$$

where $P_{ij} = \Pr(p_{t+1} = j \mid p_t = i)$.

This means:

- When price is **low** (p_L), it stays low with 80% probability, rises to high with 20%.
- When price is **high** (p_H), it stays high with 70% probability, falls to low with 30%.

The Markov transition matrix P captures **price persistence** or **volatility**.
- If diagonal elements are close to 1, prices are persistent and predictable. - If off-diagonals are large, the price fluctuates more frequently.

These transition probabilities summarize *all relevant uncertainty* about future prices — no additional information about earlier prices is needed. That is the **Markov property**.

1.4.4 Bellman recursion with discrete price states

At each period t and ore stock s_t, we solve for both price states:

$$V_t(s_t, p_L) = \max_{x_t \in [0, s_t]} \Big[p_L x_t - c(x_t) + \beta (0.8 \, V_{t+1}(s_t - x_t, p_L) + 0.2 \, V_{t+1}(s_t - x_t, p_H)) \Big],$$

$$V_t(s_t, p_H) = \max_{x_t \in [0, s_t]} \Big[p_H x_t - c(x_t) + \beta (0.3 \, V_{t+1}(s_t - x_t, p_L) + 0.7 \, V_{t+1}(s_t - x_t, p_H)) \Big].$$

The manager now faces **two dimensions of state**:

1. **Physical stock** (s_t): determines extraction feasibility.
2. **Price state** (p_t): determines current revenue and expected continuation value.

The decision rule $x_t^*(s_t, p_t)$ will generally satisfy:

- Extract more when price is high (p_H)
- Conserve when price is low (p_L)

The trade-off is now between:

- Exploiting the resource when prices are favorable
- Saving for the future when low prices make extraction less profitable