Big data decision making: real-time optimization for real-world problems

Sandjai Bhulai
s.bhulai@vu.nl

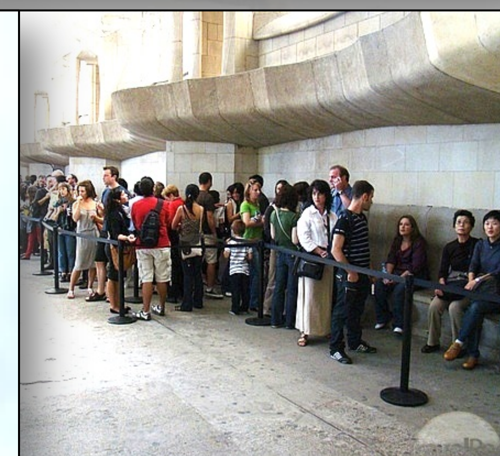
VU VRIJE UNIVERSITEIT AMSTERDAM

IS VERDER KIJKEN

Decision making in practice

- Sunday, September 23:
  - What to do with the time between 10am and 3pm?

Decision making in practice



Sandjai Bhulai (s.bhulai@vu.nl)

The challenge of data

Data problems in the real world:

- Bad data
  - We have the data, but it is not correct
- Uncertainty in forecasted data
  - Our decisions need to consider events in the future, but we cannot forecast them perfectly
- Incomplete information
  - We are simply not modeling certain costs or physical constraints
- Errors in implementation
  - The decisions recommended by the model are not the decisions that are implemented in the field

Sandjai Bhulai (s.bhulai@vu.nl)

The challenge of language

The languages of “optimization over time”

	Engineering	OR/AI/Probability	OR/Math programming
Discipline	Optimal control	Markov decision processes	Stochastic programming
Decision (English)	Control	Action	Decision
Decision (Math)	u	a	x
"Value function" (English)	Cost-to-go	Value function	Recourse function
"Value function" (Math)	J	V	Q
State variable	x	S	tenders
Optimality equations	Hamilton-Jacobi	Bellman	Huh?

The challenge of methodology

Competing methodologies:

- Deterministic optimization
  - Problem is NP-complete
  - Heuristics provide high-quality overall solutions, but can produce quirky solutions when evaluated up close
  - Puts equal weight on decisions now and in the future
  - Models "here and now" and the future at the same level of detail
- Simulation
  - Able to handle a very high level of detail, but …
  - Does not attempt to provide the best possible solution
  - Suffers from complex rules needed to make decisions
- Stochastic programming
  - Explodes problem size
- Dynamic programming/Markov decision processes
  - You have to be kidding!

A progression of models

Major problem classes

	Simple attributes	Complex attributes
Single entity	Textbook Markov decision process	Classical AI applications
Multiple entities	Classical OR applications	Opportunity for combining AI/OR

A resource allocation model

Attribute vectors:

$$a = \begin{bmatrix} \text{Asset class} \\ \text{Time invested} \end{bmatrix} \begin{bmatrix} \text{Type} \\ \text{Location} \\ \text{Eqpmnt} \end{bmatrix} \begin{bmatrix} \text{Location} \\ \text{ETA} \\ \text{Home} \\ \text{Experience} \\ \text{Driving hours} \end{bmatrix} \begin{bmatrix} \text{Location} \\ \text{ETA} \\ \text{A/C type} \\ \text{Fuel level} \\ \text{Home shop} \\ \text{Crew} \end{bmatrix}$$

A resource allocation model

Modeling resources:

- The attributes of a single resource:

$$a = \text{the attributes of a single resource}$$

$$a \in \mathcal{A}, \text{ the attribute space}$$

- The resource state vector:

$$R_{ta} = \text{the number of resources with attribute } a$$

$$R_t = \left(R_{ta}\right)_{a \in \mathcal{A}}, \text{ the resource state vector}$$

- The information process:

$$\hat{R}_{ta} = \text{change in the number of resources with attribute } a$$

A resource allocation model

Modeling demands:

- The attributes of a single demand:

$$b = \text{the attributes of a demand to be served}$$

$$b \in \mathcal{B}, \text{ the attribute space}$$

- The demand state vector:

$$D_{tb} = \text{the number of demands with attribute } b$$

$$D_t = \left(D_{tb}\right)_{b \in \mathcal{B}}, \text{ the demand state vector}$$

- The information process:

$$\hat{D}_{tb} = \text{the change in the number of demands with attribute } b$$

Energy resource modeling

The system state:

$S_t = (R_t, D_t, \rho_t)$ = system state, where:

R_t = resource state (how much capacity, reserves)

D_t = market demands

ρ_t = "system parameters"

- State of the technology (costs, performance)
- Climate, weather (temperature, rainfall, wind)
- Government policies (tax rebates on solar panels)
- Market prices (oil, coal)

Sandjai Bhulai (s.bhulai@vu.nl)

Energy resource modeling

The decision variable:

$$x_t = \begin{pmatrix} \text{New capacity} \\ \text{Retired capacity} \\ \textit{for each}: \\ \text{Type} \\ \text{Location} \\ \text{Technology} \end{pmatrix}$$

Energy resource modeling

Exogenous information:

$$W_t = \text{new information} = \left(\hat{R}_t, \hat{D}_t, \hat{\rho}_t\right)$$

$\hat{R}_t$ = exogenous changes in capacity, reserves

$\hat{D}_t$ = new demands for energy from each source

$\hat{\rho}_t$ = exogenous changes in parameters

Sandjai Bhulai (s.bhulai@vu.nl)

Energy resource modeling

The transition function:

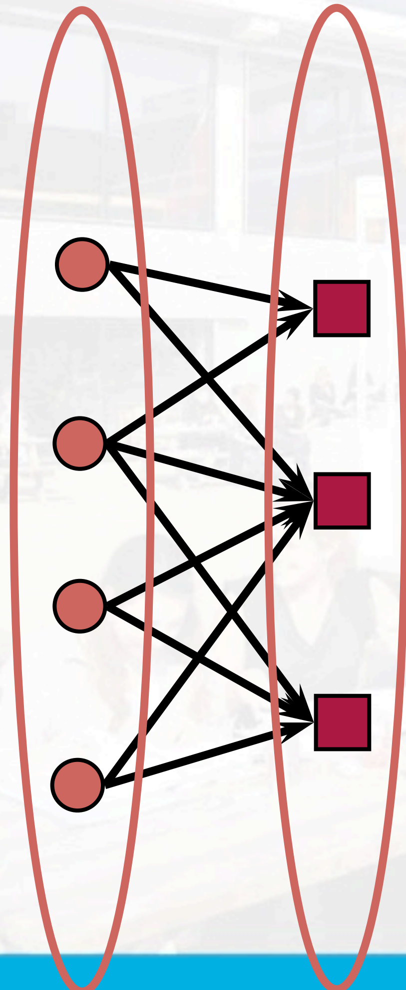



$$S_{t+1} = S^M(S_t, x_t, W_{t+1})$$

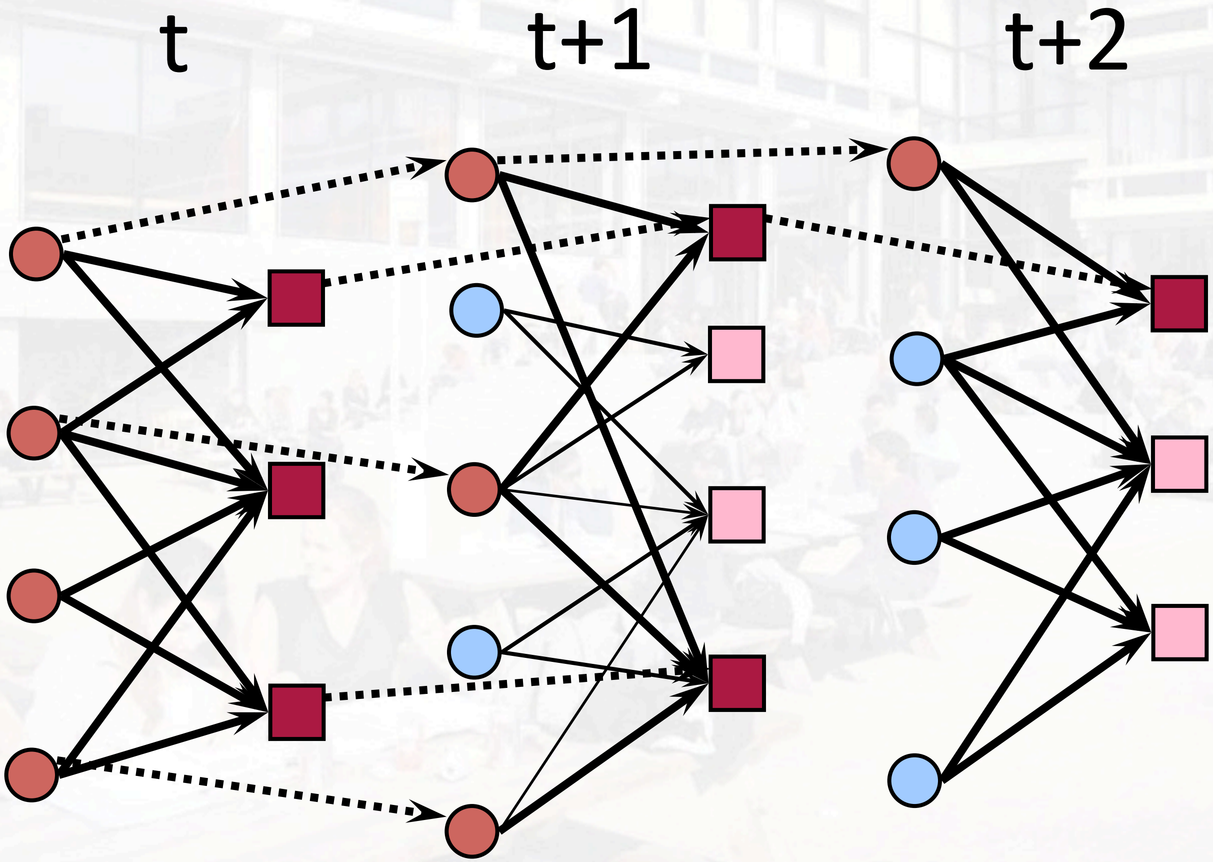
Sandjai Bhulai (s.bhulai@vu.nl)

A resource allocation model

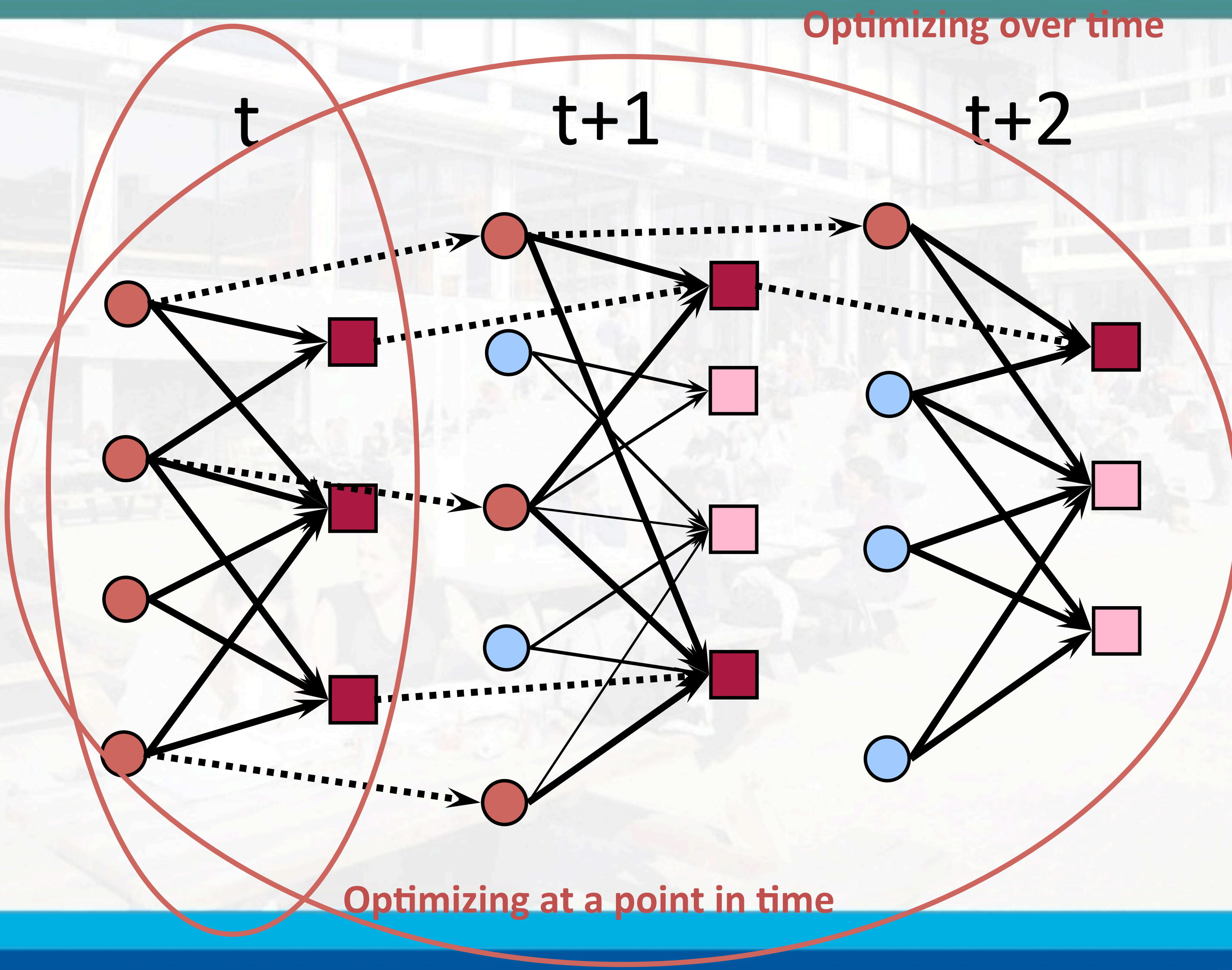
Resources Demands



A resource allocation model



A resource allocation model



Sandjai Bhulai (s.bhulai@vu.nl)

Curse of dimensionality

- We just solved Bellman's equation:

$$V_t(S_t) = \max_{x \in \mathcal{X}} C_t(S_t, x_t) + E\left\{V_{t+1}(S_{t+1}) \mid S_t\right\}$$

- We found the value of being in each state by stepping backward through the tree.

- Problem: Curse of dimensionality

Curse of dimensionality

The computational challenge:

$$V_t(S_t) = \max_{x \in \mathcal{X}} \left(C_t(S_t, x_t) + E\left\{ V_{t+1}(S_{t+1}) \mid S_t \right\} \right)$$

How do we find $V_{t+1}(S_{t+1})$?

How do we compute the expectation?

How do we find the optimal solution?

Modeling stochastic optimization problems

Policies:

- 1) Myopic policies
  - Take the action that maximizes contribution (or minimizes cost) for just the current time period:

$$X^M(S_t) = \arg\max_{x_t} C(S_t, x_t)$$

- 2) Lookahead policies
  - Plan over the next T periods, but implement only the action it tells you to do now:

$$X^M(S_t) = \arg\max_{x_t, x_{t+1}, \ldots, x_{t+T}} \sum_{t'=t}^{T} C(S_{t'}, x_{t'})$$

Modeling stochastic optimization problems

Policies:

- 3) Policies based on value function approximations

Let $\bar{V}_t(S_t)$ be an approximation of the value of being in state S_t

$$X^M(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \gamma E \bar{V}_{t+1}(S_{t+1}) \right)$$

- 4) Policy function approximations

Let $\bar{X}(S_t)$ be a function that directly tells you an action given that you are in a state S_t.

Approximate Dynamic Programming

Classical ADP:

- Most applications of ADP focus on the challenge of handling multidimensional state variables
  - Start with

$$V_t(S_t) = \max_{x \in X} \left(C_t(S_t, x_t) + E\left\{ V_{t+1}(S_{t+1}) \mid S_t \right\} \right)$$

  - Now replace the value function with some sort of approximation

$$V_t(S_t) \approx \bar{V}_t(S_t)$$

Approximate Dynamic Programming

Approximating the value function:

- We have to exploit the structure of the value function (e.g., convexity, submodularity, etc.)
  - We might approximate the value function using a simple polynomial

$$\bar{V}_t(S_t \mid \theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2$$

  - … or a complicated one:

$$\bar{V}_t(S_t \mid \theta) = \theta_0 + \theta_1 S_t + \theta_2 S_t^2 + \theta_3 \ln\left(S_t\right) + \theta_4 \sin(S_t)$$

  - Sometimes, they get really messy

Approximate Dynamic Programming

- We can write a model of the observed value of being in a state as:

$$\hat{v} = \theta_0 + \theta_1 S_t + \theta_2 S_t^2 + \theta_3 \ln\left(S_t\right) + \theta_4 \sin(S_t) + \varepsilon$$

- This is often written as a generic regression model:

$$Y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 + \theta_4 X_4$$

- The ADP community refers to the independent variables as *basis functions*:

$$Y = \theta_0 \varphi_0(S) + \theta_1 \varphi_1(S) + \theta_2 \varphi_2(S) + \theta_3 \varphi_3(S) + \theta_4 \varphi_4(S)$$

$$= \sum_{f \in \mathrm{F}} \theta_f \varphi_f(S)$$

$\varphi_f(R)$ are also known as *features*

Approximate Dynamic Programming

- Methods for estimating θ
  - Generate observations $\hat{v}^1, \hat{v}^2, \ldots, \hat{v}^N$, and use traditional regression methods to fit θ
  - Stochastic gradient for updating θ^n:

$$\theta^n = \theta^{n-1} - \alpha_{n-1}\left(\bar{V}^{n-1}(S^n \mid \theta^{n-1}) - \hat{v}^n\right)\nabla \bar{V}^{n-1}(S^n \mid \theta^{n-1})$$

$$= \theta^{n-1} - \alpha_{n-1}\left(\bar{V}^{n-1}(S^n \mid \theta^{n-1}) - \hat{v}^n\right)\begin{pmatrix} \varphi_1(S) \\ \varphi_2(S) \\ \\ \varphi_F(S) \end{pmatrix}$$

Approximate Dynamic Programming

Other statistical methods:

- Regression trees
  - Combines regression with techniques for discrete variables
- Data mining
  - Good for categorical data
- Neural networks
  - Engineers like this for low-dimensional continuous problems
- Kernel/locally polynomial regression
  - Approximations portions of the value function locally using simple functions
- Dirichlet mixture models
  - Aggregate portions of the function and fit approximations around these aggregations.

Approximate Dynamic Programming

What you will struggle with:

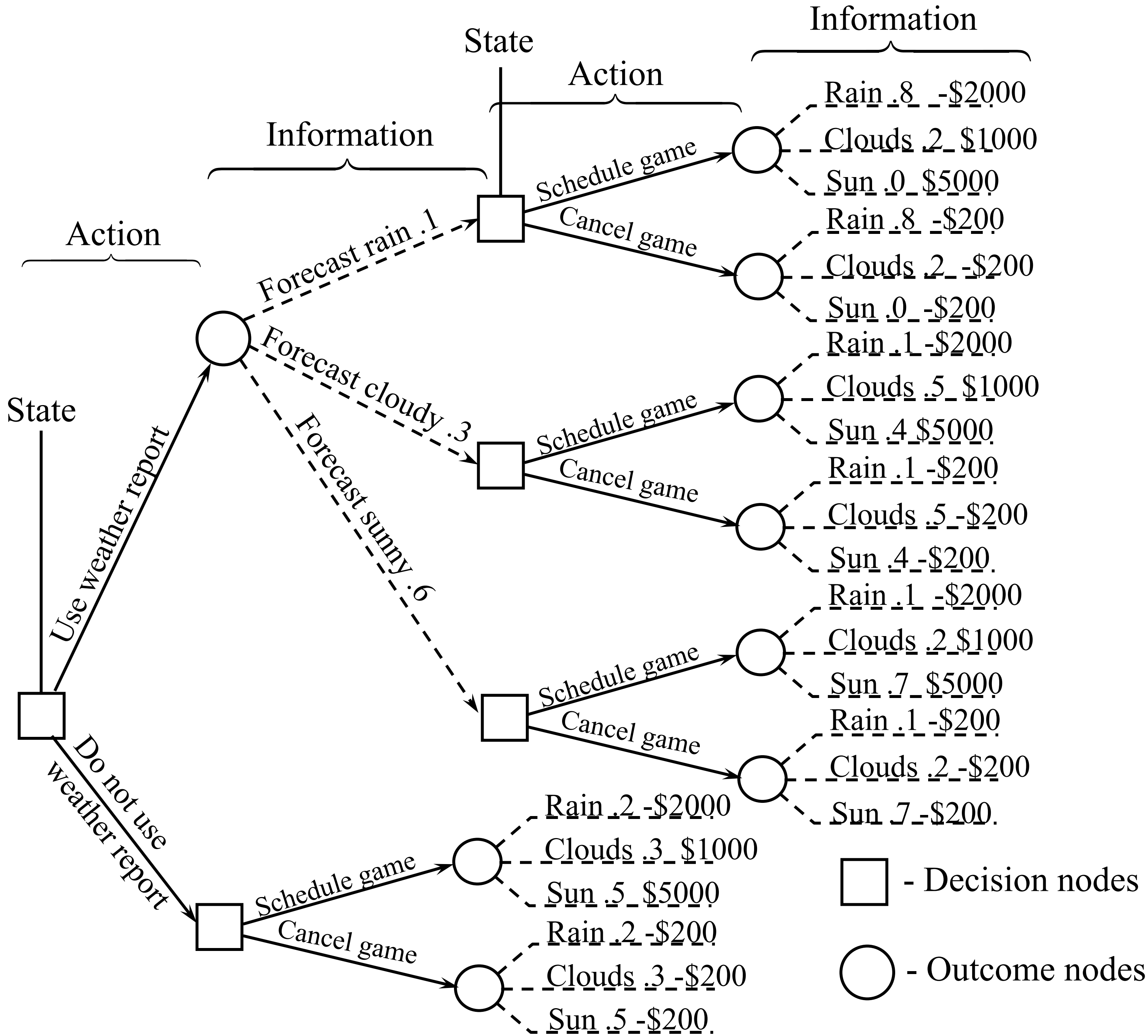
- Stepsizes
  - Can't live with 'em, can't live without 'em
  - Too small, you think you have converged but you have really just stalled ("apparent convergence")
  - Too large, and the system is unstable
- Stability
  - There are two sources of randomness:
    - The traditional exogenous randomness
    - An evolving policy
- Exploration vs. exploitation
  - You sometimes have to choose to visit a state just to collect information about the value of being in a state

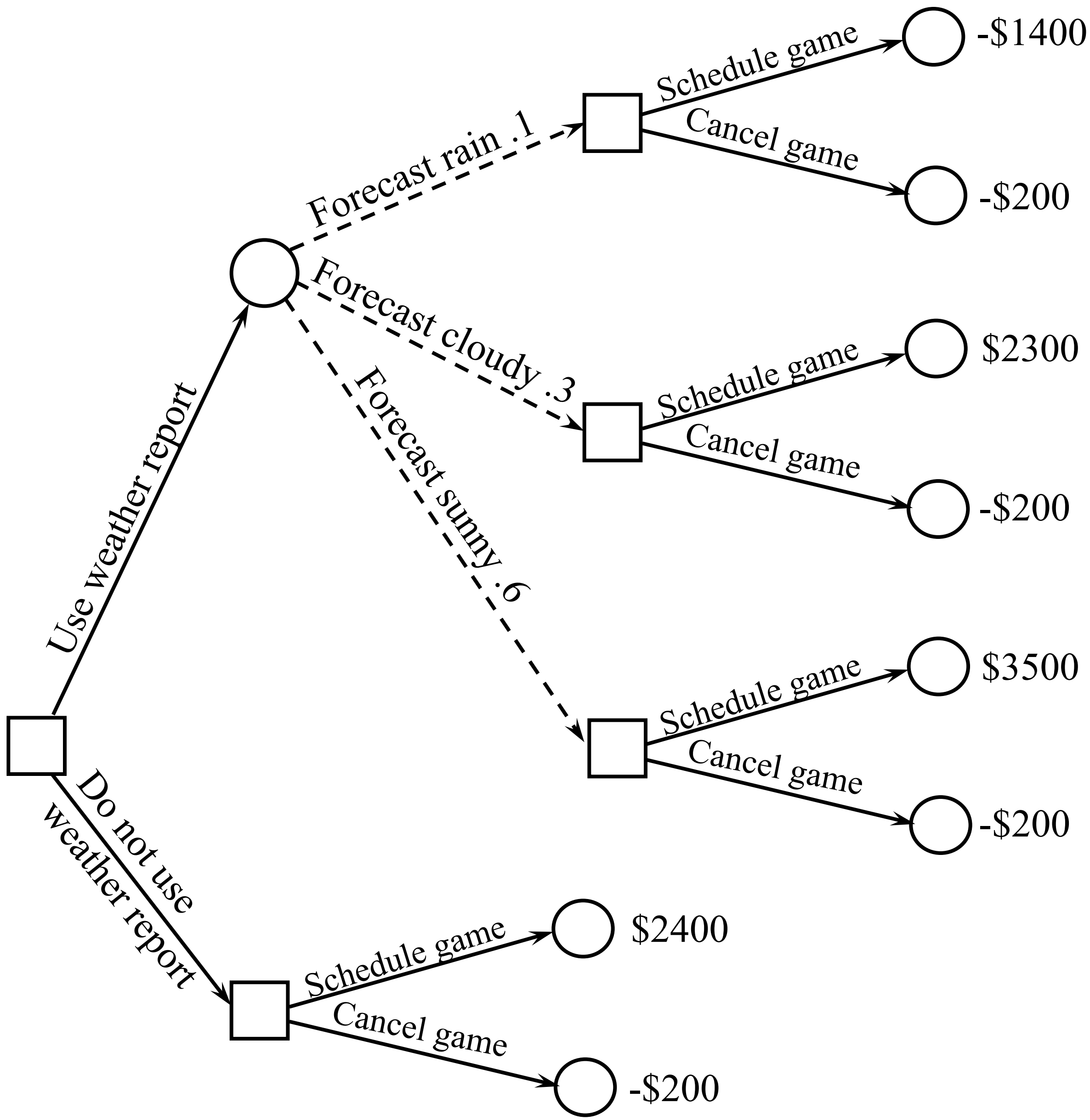
Approximate Dynamic Programming

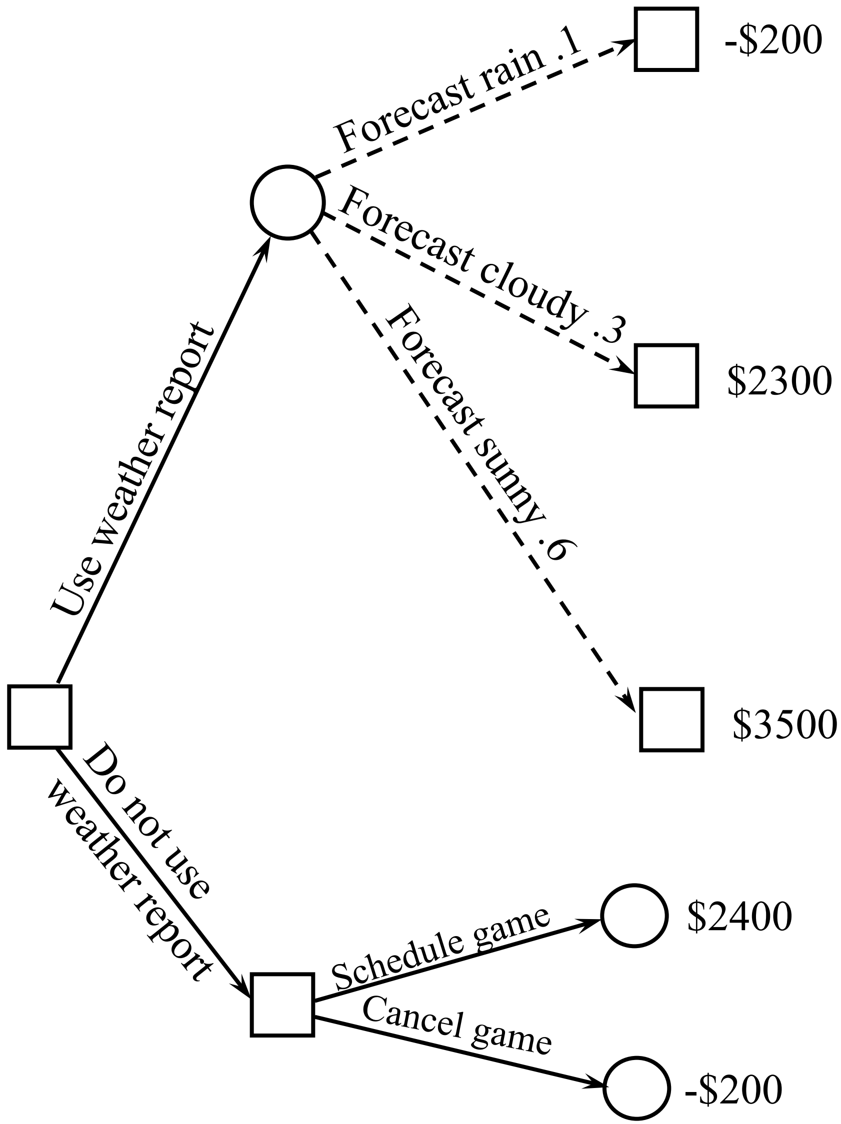
- But we are not out of the woods…
  - Assume we have an approximate value function
  - We still have to solve a problem that looks like

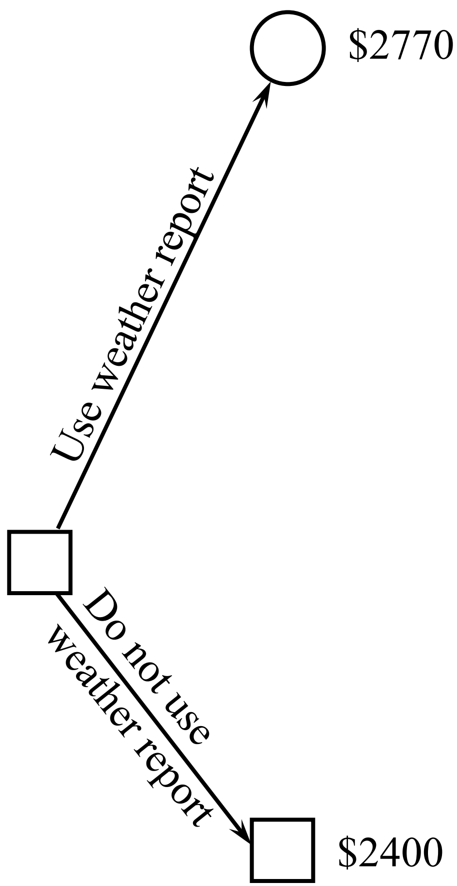
$$V_t(S_t) = \max_{x \in \mathcal{X}} \left(C_t(S_t, x_t) + E \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_{t+1}) \right)$$

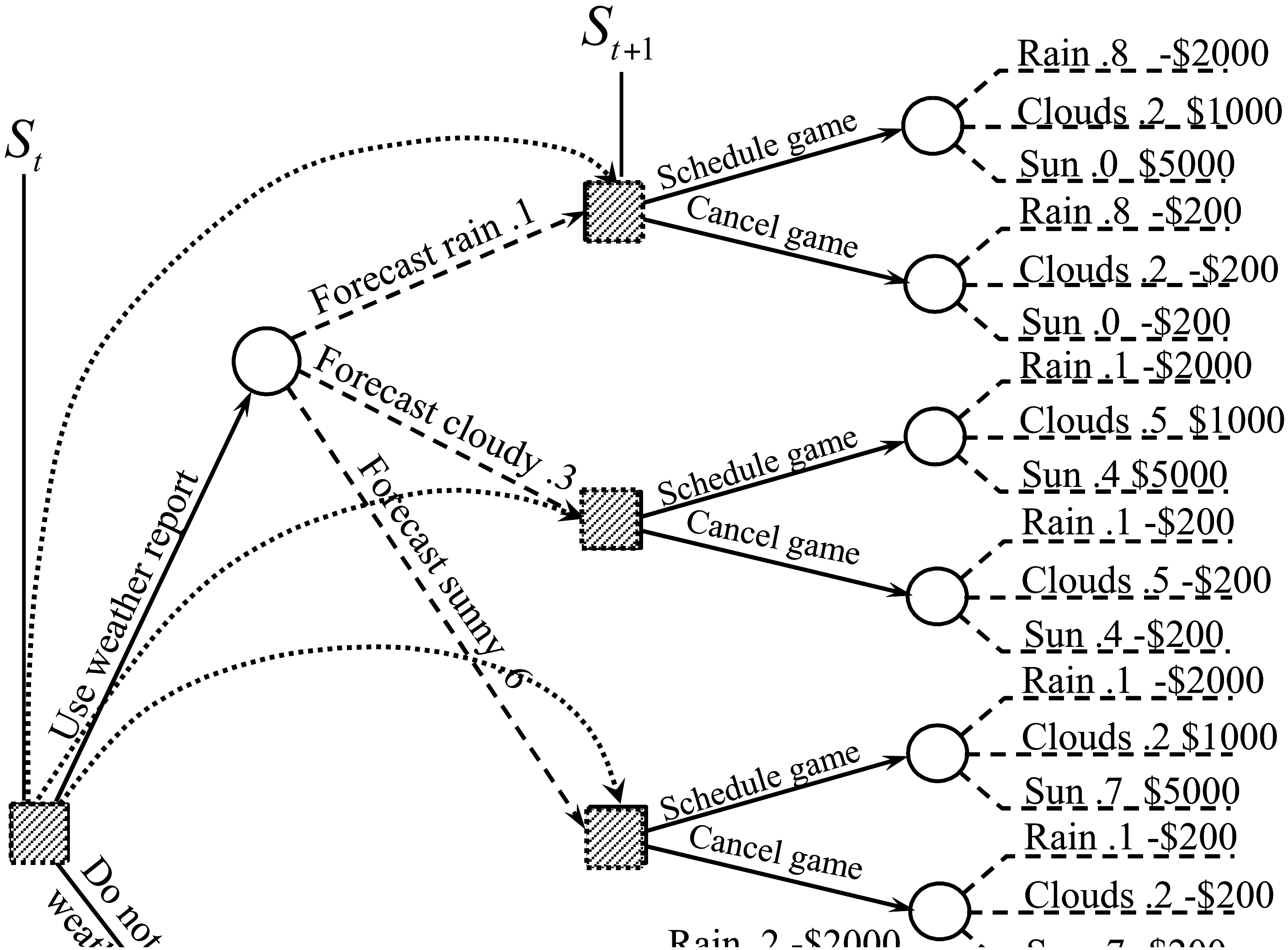
  - This means we still have to deal with a maximization problem (might be a linear, nonlinear or integer program) with an expectation


Information
State
Action
Information
Action
State
Use weather report
Do not use weather report
Forecast rain .1
Forecast cloudy .3
Forecast sunny .6
Schedule game
Cancel game
Rain .8 -$2000
Clouds .2 $1000
Sun .0 $5000
Rain .8 -$200
Clouds .2 -$200
Sun .0 -$200
Rain .1 -$2000
Clouds .5 $1000
Sun .4 $5000
Rain .1 -$200
Clouds .5 -$200
Sun .4 -$200
Rain .1 -$2000
Clouds .2 $1000
Sun .7 $5000
Rain .1 -$200
Clouds .2 -$200
Sun .7 -$200
Rain .2 -$2000
Clouds .3 $1000
Sun .5 $5000
Rain .2 -$200
Clouds .3 -$200
Sun .5 -$200
- Decision nodes
- Outcome nodes

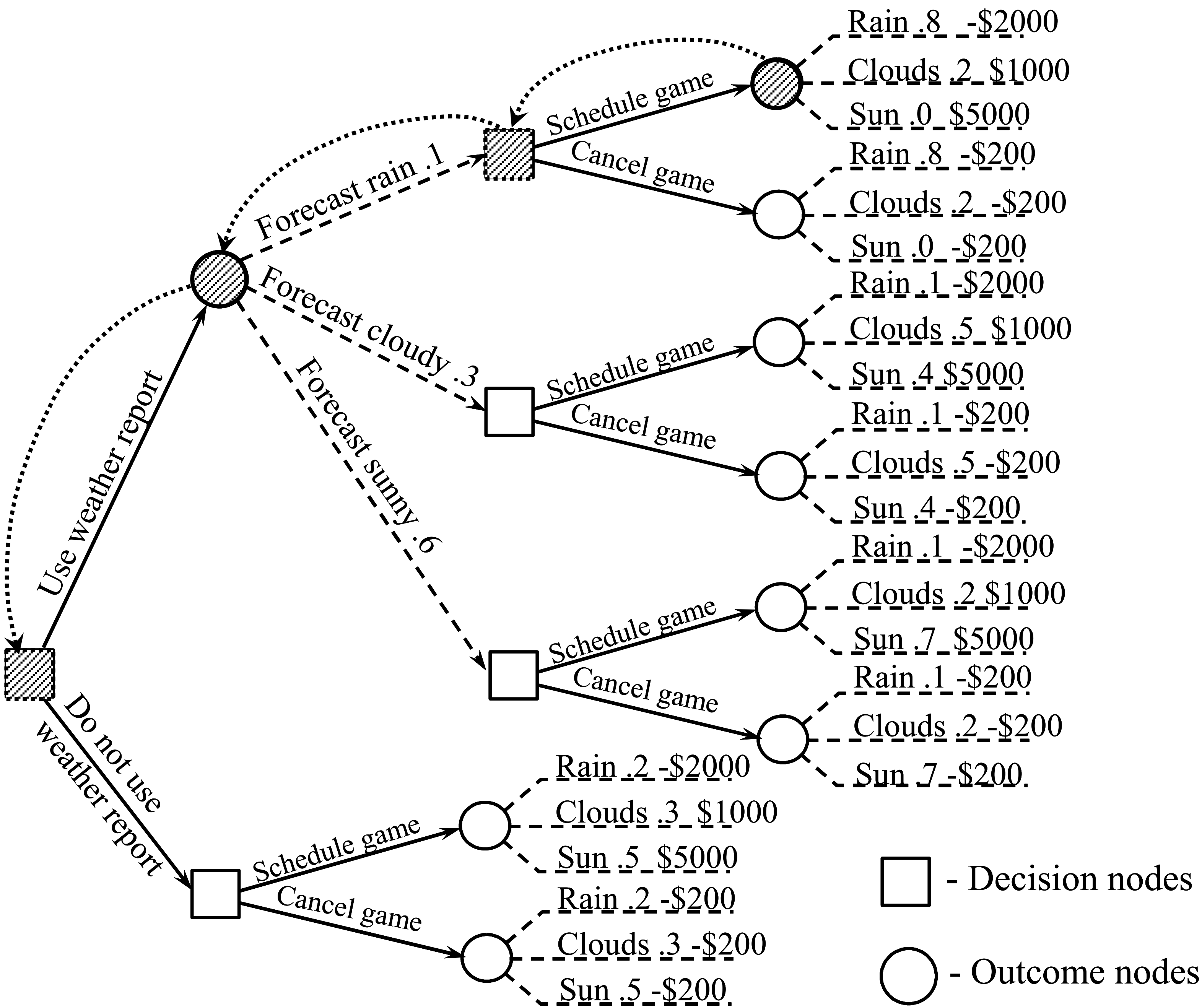

Use weather report
Forecast rain .1
Forecast cloudy .3
Forecast sunny .6
Schedule game
Cancel game
-$1400
-$200
Schedule game
Cancel game
$2300
-$200
Schedule game
Cancel game
$3500
-$200
Do not use
weather report
Schedule game
Cancel game
$2400
-$200


Use weather report
Forecast rain .1
-$200
Forecast cloudy .3
$2300
Forecast sunny .6
$3500
Do not use
weather report
Schedule game
$2400
Cancel game
-$200


$2770
Use weather report
Do not use
weather report
$2400



$$V_t(S_t) = \max_{x \in \mathcal{X}} \left(C_t(S_t, x_t) + E\left\{ V_{t+1}(S_{t+1}) \mid S_t \right\} \right)$$



Use weather report
Do not use weather report
Forecast rain .1
Forecast cloudy .3
Forecast sunny .6
Schedule game
Cancel game
Rain .8 -$2000
Clouds .2 $1000
Sun .0 $5000
Rain .8 -$200
Clouds .2 -$200
Sun .0 -$200
Rain .1 -$2000
Clouds .5 $1000
Sun .4 $5000
Rain .1 -$200
Clouds .5 -$200
Sun .4 -$200
Rain .1 -$2000
Clouds .2 $1000
Sun .7 $5000
Rain .1 -$200
Clouds .2 -$200
Sun .7 -$200
Rain .2 -$2000
Clouds .3 $1000
Sun .5 $5000
Rain .2 -$200
Clouds .3 -$200
Sun .5 -$200
- Decision nodes
- Outcome nodes

Transition function

Traditional modeling of dynamics

$$S_{t+1} = S^M(S_t, x_t, W_{t+1})$$

System model

Transition function

- S_{t+1}: New state
- S_t: Current state of the system
- x_t: Decision
- W_{t+1}: New information

The post-decision state

- New concept:
  - The "pre-decision" state variable:
    - S_t = The information required to make a decision x_t
    - Same as a "decision node" in a decision tree
  - The "post-decision" state variable:
    - S_t^x = The state of what we know immediately after we make a decision.
    - Same as an "outcome node" in a decision tree

Sandjai Bhulai (s.bhulai@vu.nl)

The post-decision state

- Breaking down the system dynamics:
  - Instead of modeling from “pre-” to “pre-” …

$$S_{t+1} = S^M(S_t, x_t, W_{t+1})$$

  - … we use one function to go from “pre-” to “post-” …

$$S_t^x = S^{M,x}(S_t, x_t)$$

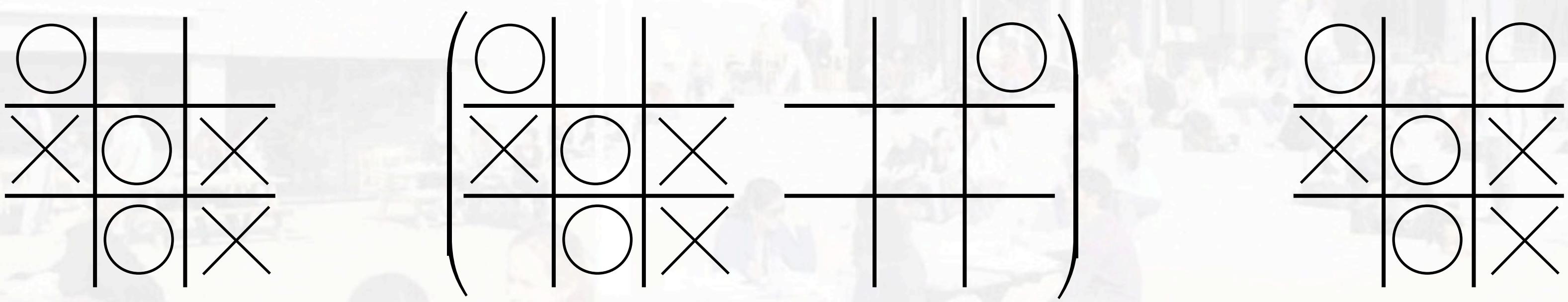
  - … with another function from “post-” to “pre-”

$$S_{t+1} = S^{M,W}(S_t^x, W_{t+1})$$

The post-decision state

Pre-decision, state-action, and post-decision

Pre-decision state | State | Action | Post-decision state

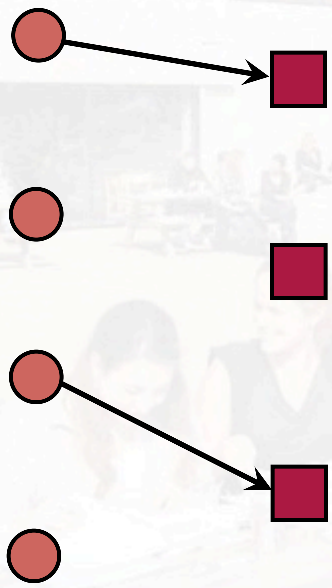


3^9 states | $3^9 \times 9$ state-action pairs | 3^9 states

The post-decision state

Pre-decision: resources and demands

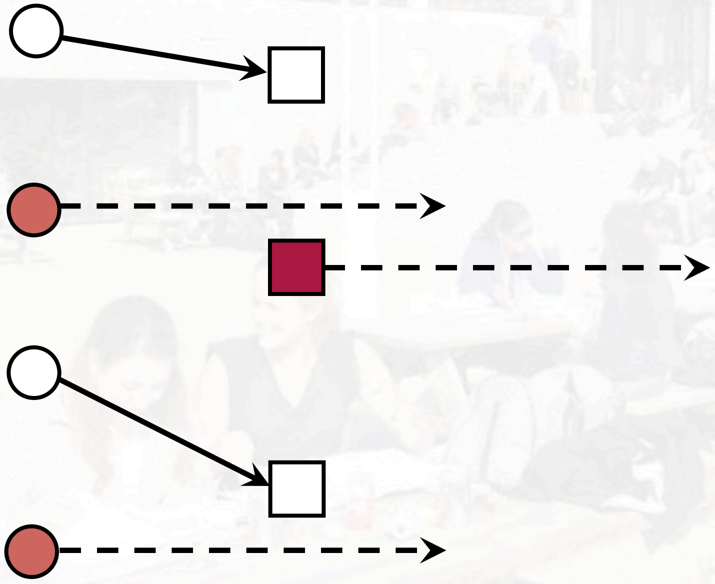
$$S_t = (R_t, D_t)$$



Sandjai Bhulai (s.bhulai@vu.nl)

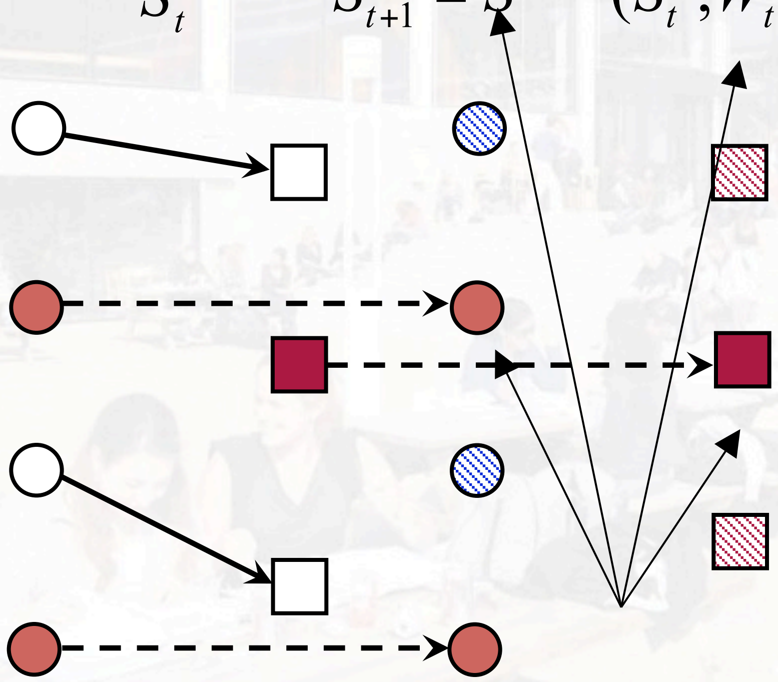
The post-decision state

$$S_t^x = S^{M,x}(S_t, x_t)$$



The post-decision state

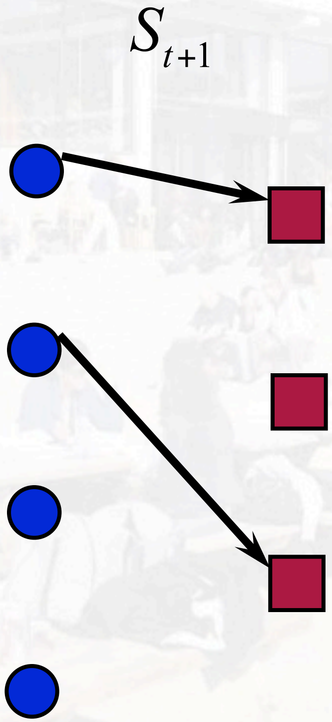
$S_t^x \qquad S_{t+1} = S^{M,W}(S_t^x, W_{t+1})$



$$W_{t+1} = (\hat{R}_{t+1}, \hat{D}_{t+1})$$

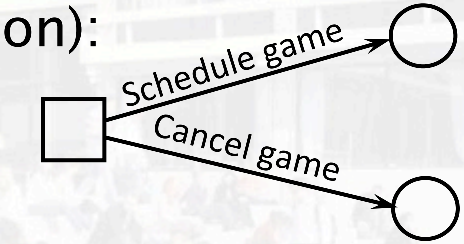
Sandjai Bhulai (s.bhulai@vu.nl)

The post-decision state



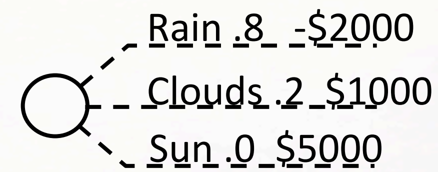
The algorithm

- Bellman's equations broken into stages:
  - Optimization problem (making the decision):



$$V_t(S_t) = \max_x \left(C_t(S_t, x_t) + V_t^x\left(S_t^{M,x}(S_t, x_t)\right)\right)$$

    - Note: this problem is deterministic!
  - Simulation problem (the effect of exogenous information):



$$V_t^x(S_t^x) = E\left\{V_{t+1}(S^{M,W}(S_t^x, W_{t+1})) \mid S_t^x\right\}$$

    - Need to compute expectation.
  - Challenge: What is $V_t^x\left(S_t^{M,x}(S_t, x_t)\right)$

The algorithm

- Comparison to other methods:
  - Classical MDP (value iteration)

$$V^n(S) = \max_x \left(C(S,x) + \gamma E V^{n-1}(S_{t+1}) \right)$$

  - Classical ADP (pre-decision state):

$$\hat{v}_t^n = \max_x \left(C_t(S_t^n, x_t) + \sum_{s'} p(s' \mid S_t^n, x_t) \bar{V}_{t+1}^{n-1}(s') \right) \quad \text{Expectation}$$

$$\bar{V}_t^n(S_t^n) = (1-\alpha_{n-1}) \bar{V}_t^{n-1}(S_t^n) + \alpha_{n-1} \hat{v}_t^n \quad \hat{v}_t \text{ updates } \bar{V}_t(S_t)$$

  - Our method (update $\bar{V}_t^{x,n-1}$ around post-decision state):

$$\hat{v}_t^n = \max_x \left(C_t(S_t^n, x_t) + \bar{V}_t^{x,n-1}(S^{M,x}(S_t^n, x_t)) \right)$$

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1-\alpha_{n-1}) \bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1} \hat{v}_t^n \quad \hat{v}_t \text{ updates } \bar{V}_{t-1}(S_{t-1}^x)$$

The algorithm

Step 1: Start with a pre-decision state S_t^n

Step 2: Solve the deterministic optimization using an approximate value function:

$$\hat{v}_t^n = \max_x \left(C_t(S_t^n, x_t) + \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t)) \right)$$

to obtain x_t^n .

Deterministic optimization

Step 3: Update the value function approximation

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1-\alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n$$

Recursive statistics

Step 4: Obtain Monte Carlo sample of $W_t(\omega^n)$ and compute the next pre-decision state:

$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

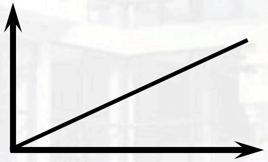
Simulation

Step 5: Return to step 1.

The algorithm

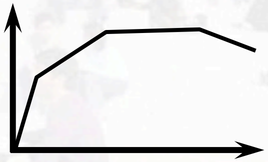
- Value function approximations:
  - Linear (in the resource state):

$$\bar{V}_t(R_t^x) = \sum_{a \in A} \bar{v}_{ta} \cdot R_{ta}^x$$



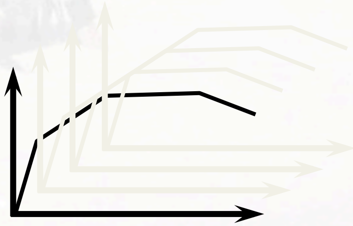
  - Piecewise linear, separable:

$$\bar{V}_t(R_t^x) = \sum_{a \in A} \bar{V}_{ta}(R_{ta}^x)$$



  - Indexed PWL separable:

$$\bar{V}_t(R_t^x) = \sum_{a \in A} \bar{V}_{ta}\left(R_{ta}^x \mid (features_t)\right)$$



Sandjai Bhulai (s.bhulai@vu.nl)

Stepsizes

Stepsizes:

- Fundamental to ADP is an updating equation that looks like:

$$V_{t-1}^{n}(S_{t-1}^{x}) = (1-\alpha_{n-1})V_{t-1}^{n-1}(S_{t-1}^{x}) + \alpha_{n-1}\hat{v}_t^n$$

Updated estimate

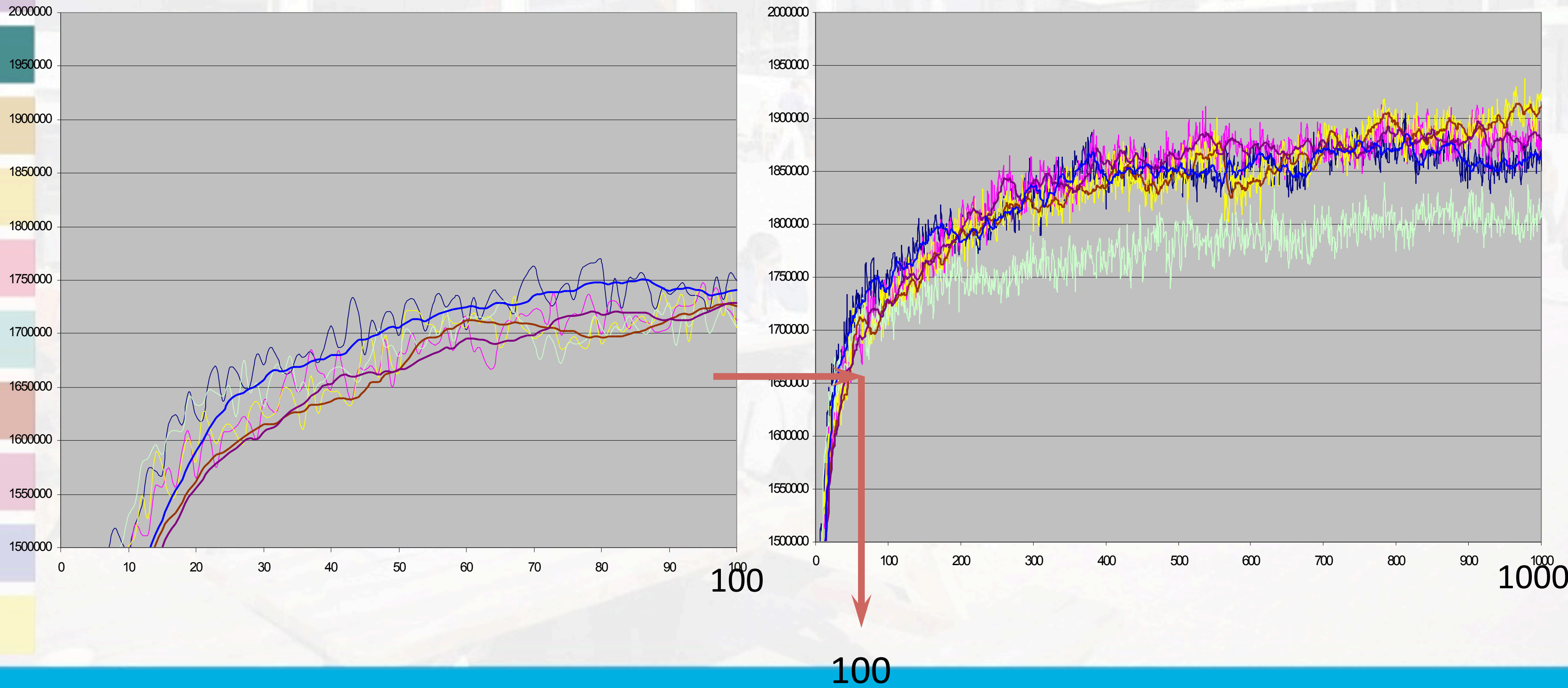
Old estimate

New observation

The stepsize
"Learning rate"
"Smoothing factor"

Stepsizes

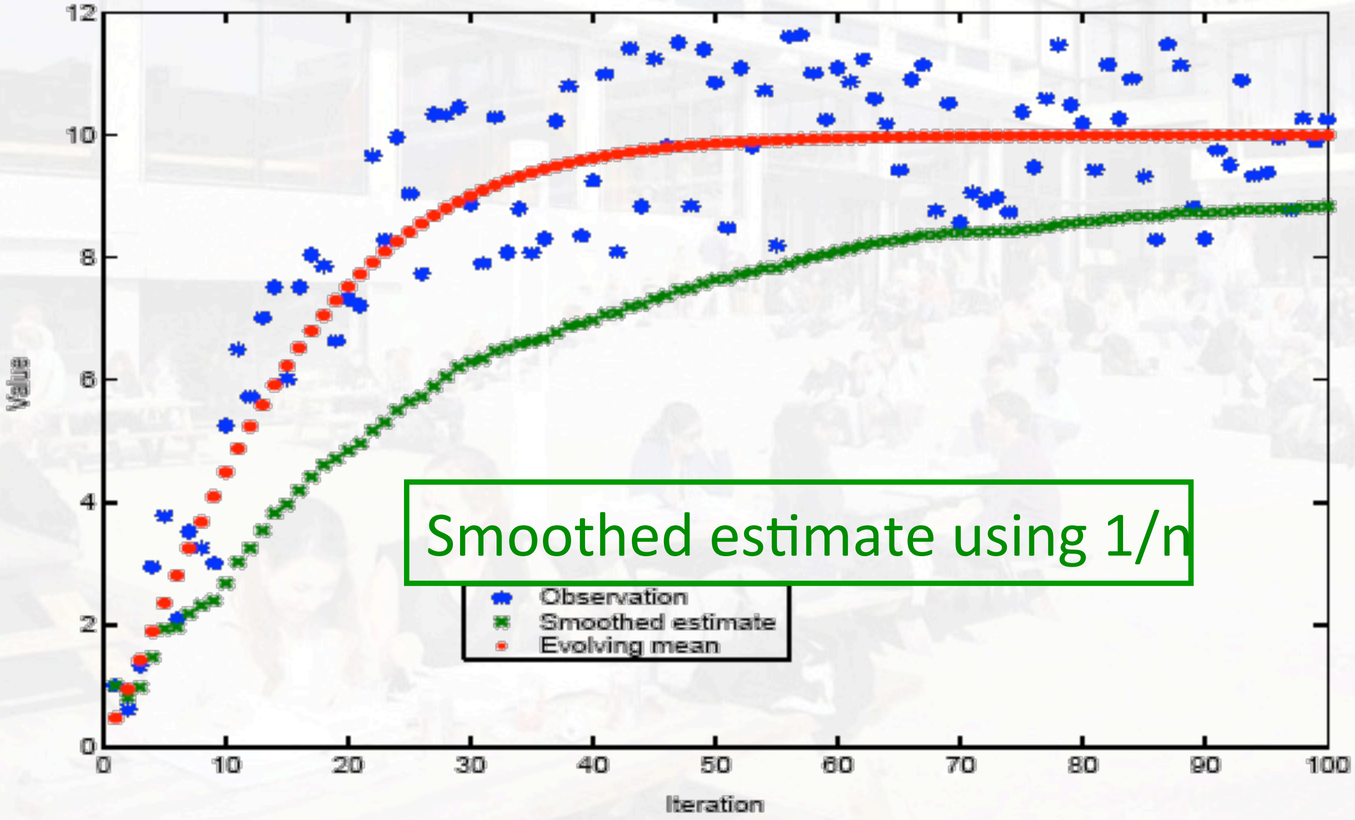
- The challenge of stepsizes:
  - When have we converged?



We need to improve our understanding of adaptive stepsizes.

Sandjai Bhulai (s.bhulai@vu.nl)

Stepsizes



Smoothed estimate using 1/n
Value
Iteration
Observation
Smoothed estimate
Evolving mean
0
2
4
6
8
10
12
0
10
20
30
40
50
60
70
80
90
100

Stepsizes

- Deterministic stepsize rules:

$$\alpha_n = \frac{1}{n} \text{ or } \frac{1}{n^\beta}$$ Basic averaging

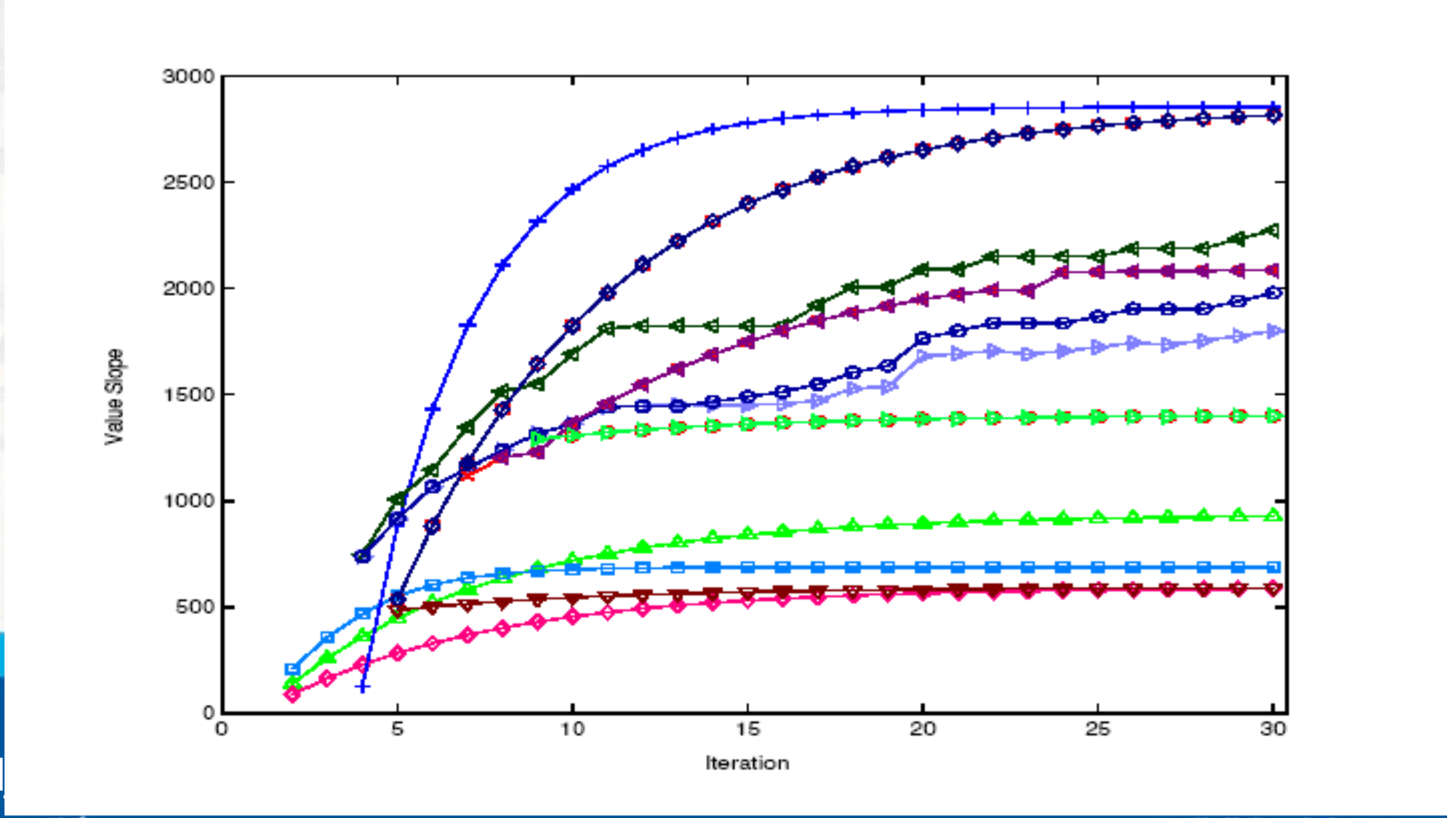
$$\alpha_n = \frac{a}{a+n-1}$$ Slows the rate of descent

$$\alpha_n = \frac{\frac{b}{n}+a}{\frac{b}{n}+a+n^\beta}$$ "Search then converge"

$$\alpha_n = \frac{\alpha_n}{1+\alpha_n-\bar{\alpha}}$$ McClain's formula

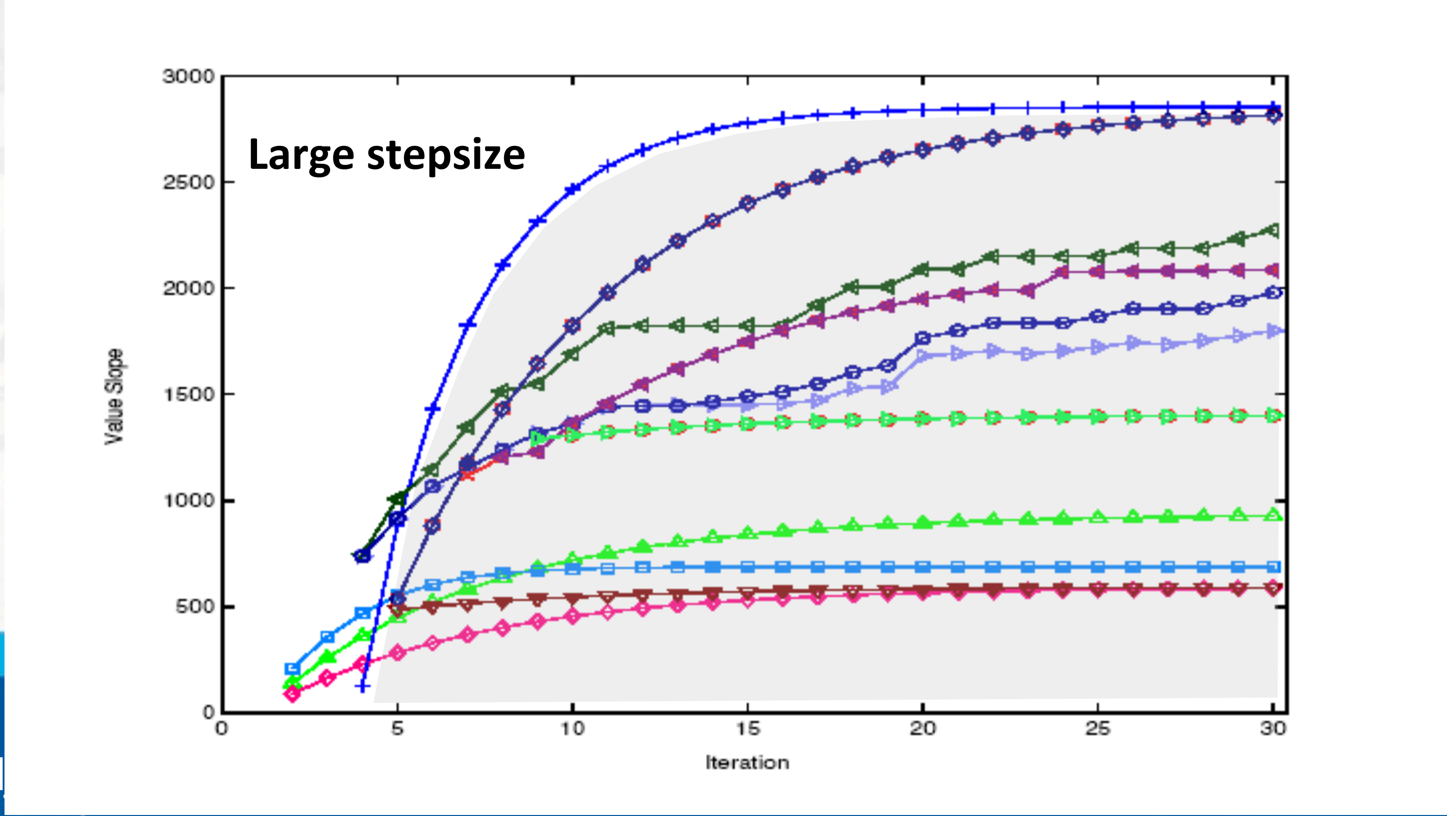
Stepsizes

- The right stepsize rule depends on the rate of change in the value function.
- This varies widely for different parameters in the same problem:



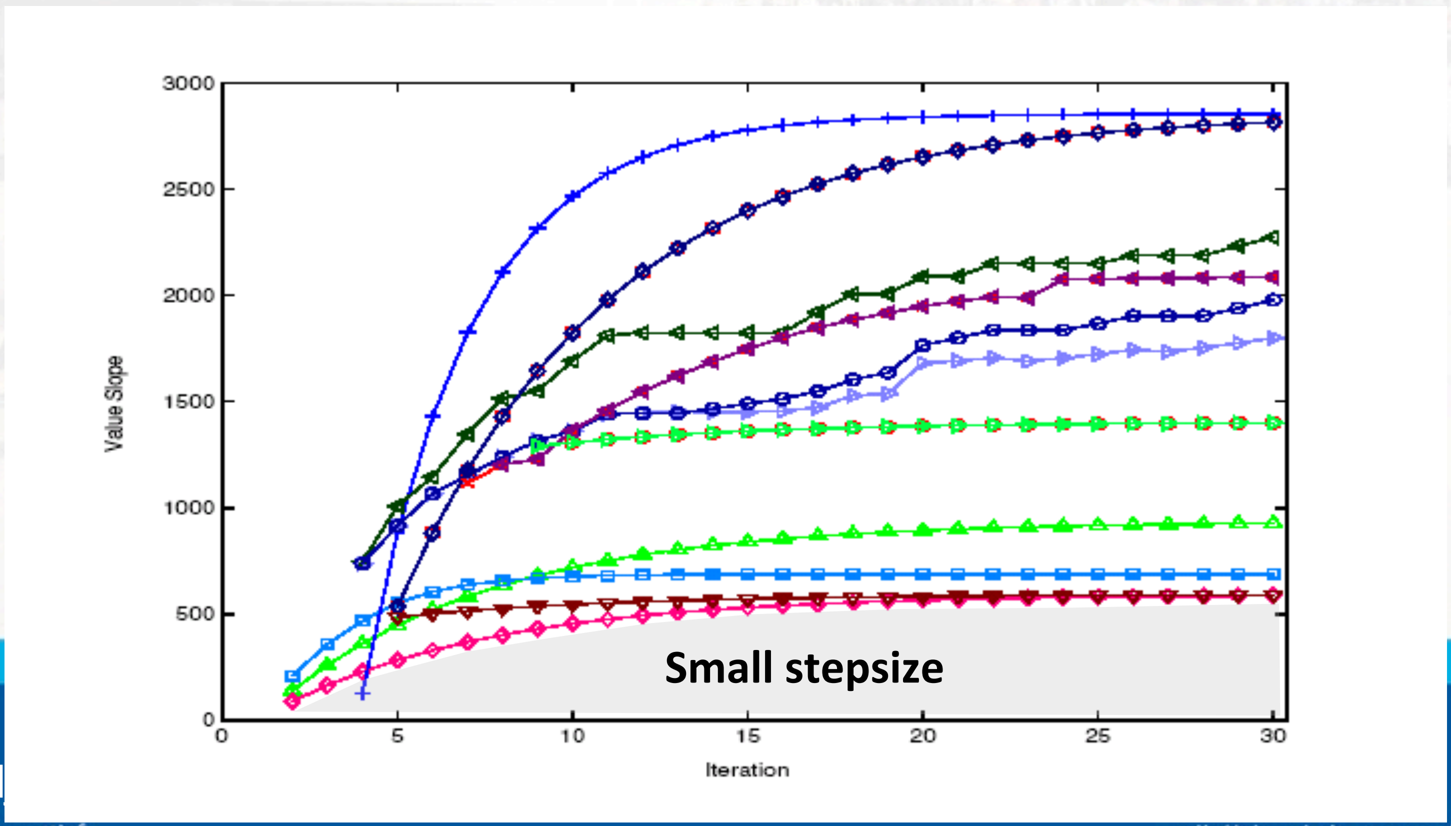
Stepsizes

- The right stepsize rule depends on the rate of change in the value function.
- This varies widely for different parameters in the same problem:



Stepsizes

- The right stepsize rule depends on the rate of change in the value function.
- This varies widely for different parameters in the same problem:

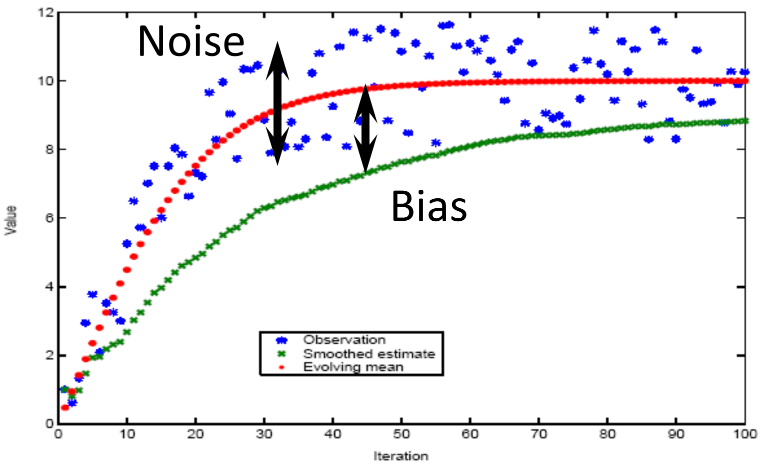


Bias-adjusted Kalman filter

Estimate of the variance

$$\alpha_n = 1 - \frac{\sigma^2}{\left(1+\lambda^{n-1}\right)\sigma^2 + \left(\beta^n\right)^2}$$

Estimate of the bias



where:

$$\lambda^n = \left(1-\alpha_n\right)^2 \lambda^{n-1} + \left(\alpha_n\right)^2$$

As σ^2 increases, stepsize decreases

As β^n increases, stepsize increases.

Sandjai Bhulai (s.bhulai@vu.nl)

Stepsizes

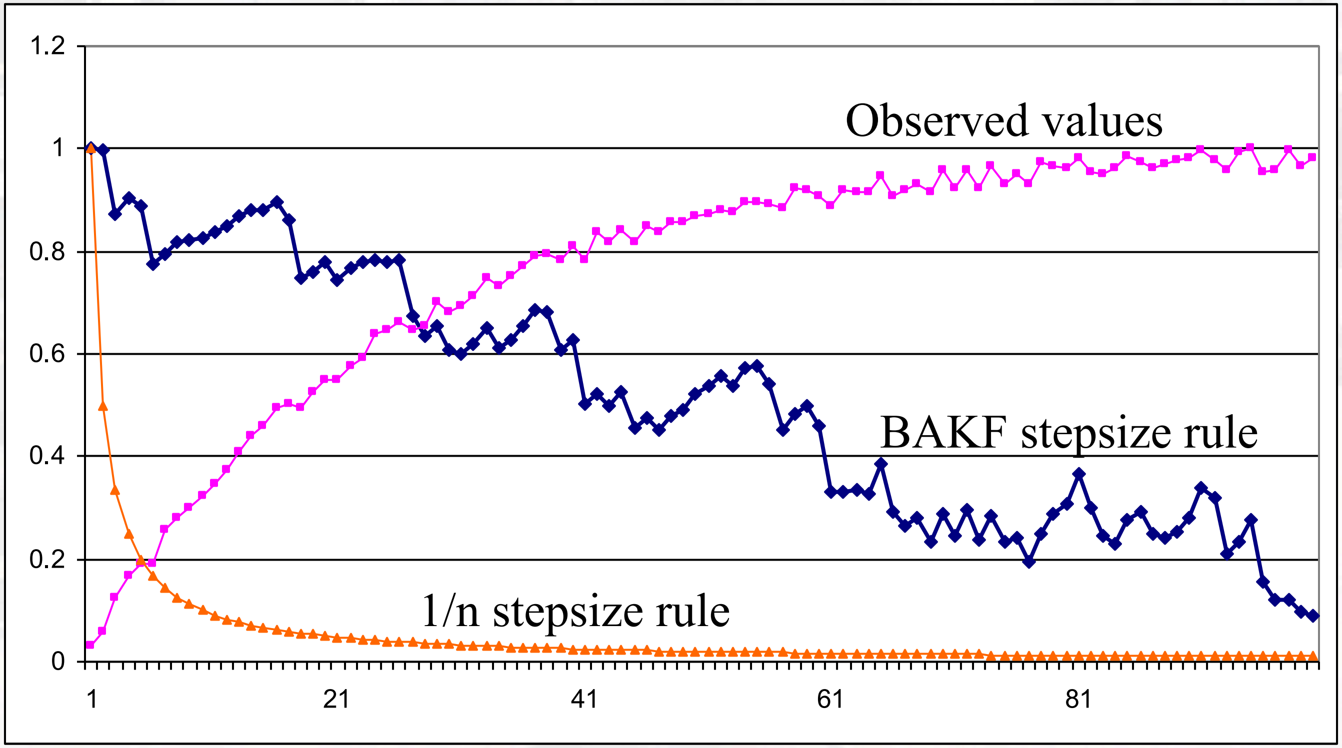
- Bias-adjusted Kalman filter
  - Properties:

$$\alpha_n \to 1 \quad \text{as } \sigma^2 \to 0$$

$$\alpha_n \to 1/n \quad \text{as } \beta \to 0 \text{ or } \sigma^2 \to \infty$$

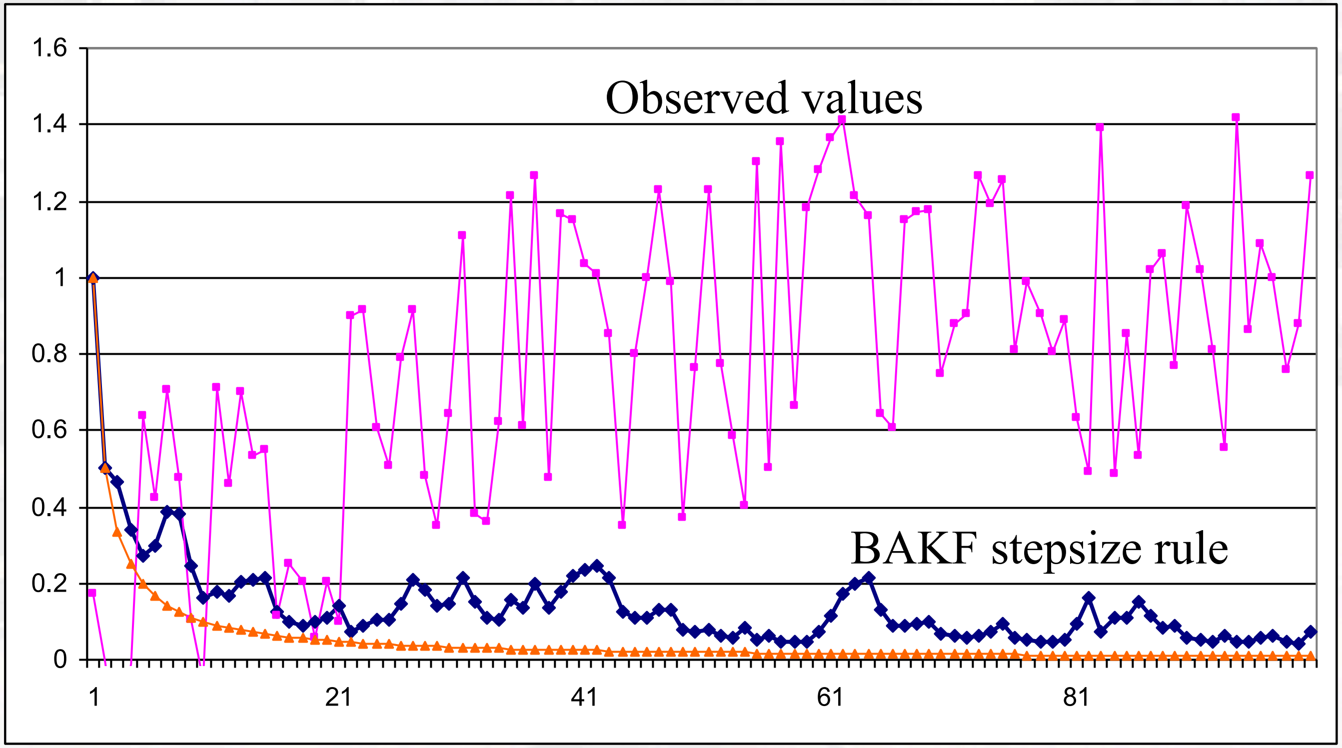
Stepsizes

- Bias-adjusted Kalman filter



Stepsizes

- Bias-adjusted Kalman filter

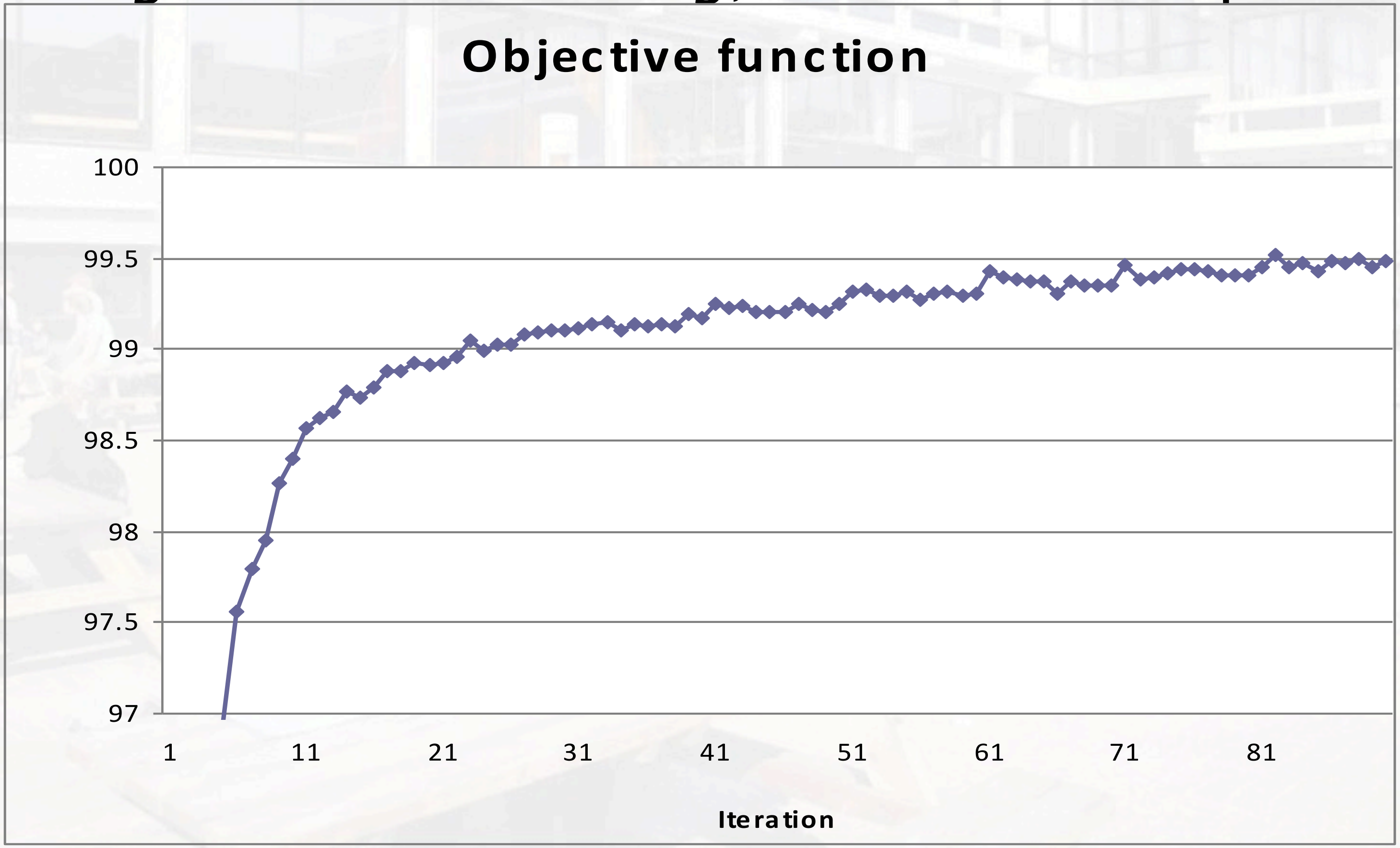


Stepsizes

- Recommendation
  - Start with a constant stepsize
    - Vary it, get a sense of what works best
  - Next try a deterministic stepsize such as a/(a+n)
    - Choose *a* so that it declines to roughly the best deterministic stepsize at an iteration comparable to where your constant stepsize seems to stabilize
      - Might be 50 iterations
      - Might be 5000
  - Try an (optimal) adaptive stepsize rule
    - Can work very well if there is not too much noise
    - Adaptive rules work well when there is a need to keep the stepsize from declining too quickly (but you do not know how quickly)

The result

Through iterative learning, the solution improves



Questions

Sandjai Bhulai (s.bhulai@vu.nl)