Management of the Unknowable

Dr. Alva L. Couch
Tufts University
Medford, Massachusetts, USA
couch@cs.tufts.edu

A counter-intuitive story

- … about **breaking well-accepted rules of practice**, and getting away with it!
- … about **intentionally ignoring available information,** and **benefiting from ignorance!**
- … about accomplishing what was considered **impossible**, by facing the **unknowable.**
- … in a way that will **seem obvious!**

What I am going to do

- **Intentionally ignore dynamics** of a system, and instead model static steady-state.
- **“Manage to manage”** the system within rather tight tolerances anyway.
- Derive **agility and flexible response** from **lack of assumptions**.
- Try to understand **why this works.**

Management now: the knowable

- Management now is based upon **what can be known.**
  - Create a model of the world.
  - Test options via the model.
  - Deploy the best option.

The unknowable

- Models of realistic systems are **unknowable.**
- The model of end-to-end response time for a network:
  - **Changes** all the time.
  - Due to perhaps **unpredictable or inconceivable influences.**
- The model of a virtual instance of a service:
  - Can’t account for **effects of other instances** running on the same hardware.
  - Can’t predict their use of **shared resources**.

Kinds of unknowable

- **Inconceivable:** unforeseen circumstances, e.g., states never experienced before.
- **Unpredictable:** never-before-experienced measurements of an otherwise predictable system.
- **Unavailable:** legal, ethical, and social limits on knowability, e.g., inability to know, predict, or even become aware of 3rd-party effects upon service.

Lessons from HotClouds 2009

- Virtualized services are influenced by 3rd party effects.
- One service can discover inappropriate information about a competitor by reasoning about influences.
- This severely limits privacy of cloud data.
- The environment in which a cloud application operates is **unknowable.**

Closed and Open Worlds

- Key concept: whether the management environment is open or closed.
- A **closed world** is one in which all influences are **knowable.**
- An **open world** contains **unknowable influences.**

Inspirations

- **Hot Autonomic Computing 2008**: “Grand Challenges of Autonomic Computing”
- Burgess’ “**Computer Immunology**”
- The theory of **management closures**.
- Limitations of **machine learning.**

Hot Autonomic Computing 2008

- Autonomic computing as proposed now will work, provided that:
  - There are **better models** of system behavior.
  - One can **compose management systems** with predictable results.
  - Humans will **trust** the result.
- These are **closed-world assumptions** that one can "**learn everything**" about the managed system.

Burgess' Computer Immunology

- Mark Burgess: management **does not require complete information.**
  - Can **act locally** toward a **global result**.
  - Desirable behavior is an **emergent property** of action.
  - Autonomic computing can be **approximated** by immunology (Burgess and Couch, MACE 2006).
- Immunology involves an **open-world assumption** that the full behavior of managed systems is **unknowable.**

Management closures

- A closure is a **self-managing component** of an **otherwise open system**.
  - A **compromise** between a **closed-world** (autonomic) and an **open-world** (immunological) approach.
  - **Domain of predictability** in an otherwise unpredictable system (Couch et al, LISA 2003).
- Closures can create **little islands** of closed-world behavior in an otherwise open world.

Machine Learning

- Machine learning approaches to management **start with an open world and try to close it.**
  - Learning involves **observing** and **codifying** an **open world**.
  - Once that model is learned, the management system functions based upon a **closed world assumption** that the model is correct.
- Learning can make a **closed world** out of an **open world** for a **while**, but that closure is not permanent.

Open worlds require open minds

- “Seeking closure” is the best way to manage an inherently closed world.
- “Agile response” is the best way to manage an inherently open world.
- This requires avoiding the temptation to **try to close an open world!**

Three big questions

- Is it **possible** to manage open worlds?
- What **form** will that management take?
- How will we **know** management is working?

The promise of open-world management

- We get **predictable composition** of management systems “for free.”
- We gain **agility and flexible response** by refusing to believe that the world is closed.
- But we have to give up an **illusion of complete knowledge** that is very comforting.

Some experiments

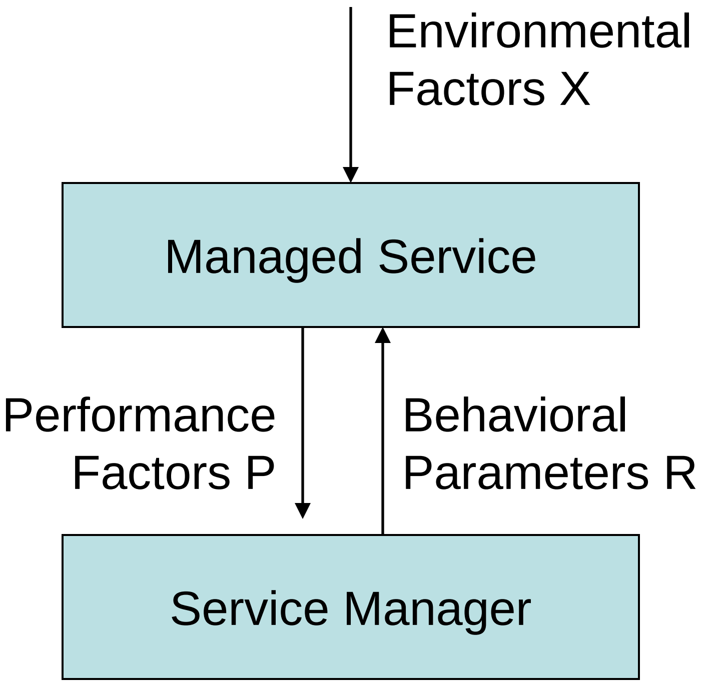
- How little can we know and still manage?
- How much can we know about how well management is doing in that case?

A minimalist approach

- Consider the **absolute minimum** of information required to control a resource.
- Operate in an **open world.**
- Model **end-to-end behavior.**
- Formulate control as a **cost/value tradeoff**.
- Study mechanisms that maximize **reward = value-cost**.
- **Avoid modeling** whenever possible.

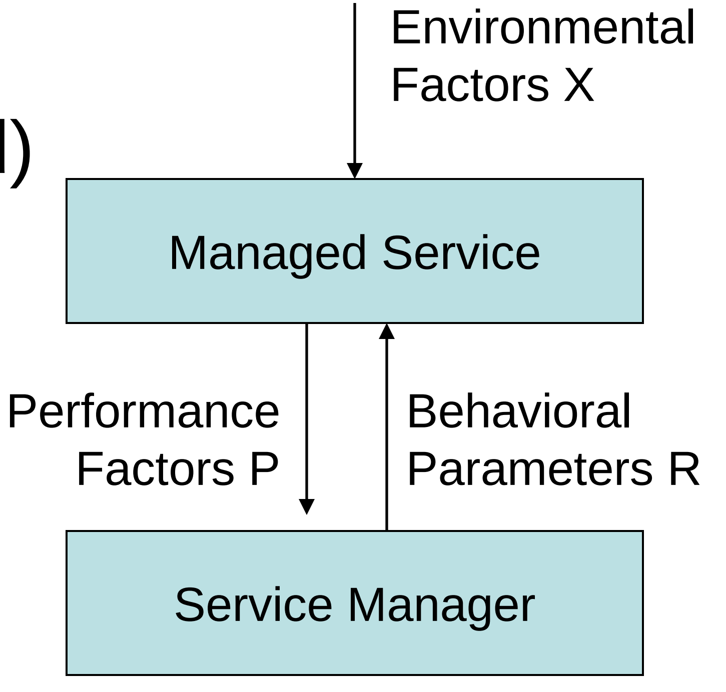
Overall system diagram

- **Resources R**: increasing R improves performance.
- **Environmental factors X** (e.g. service load, co-location, etc).
- **Performance P(R,X)**: throughput changes with resource availability and load.



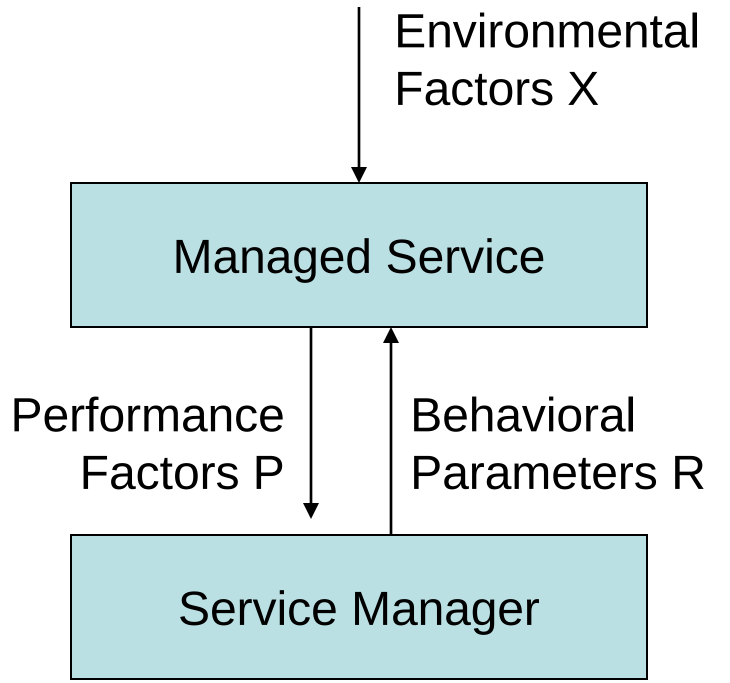
Example: streaming service in a cloud

- **X** includes input load (e.g., requests/second)
- **P** is throughput.
- **R** is number of assigned servers.



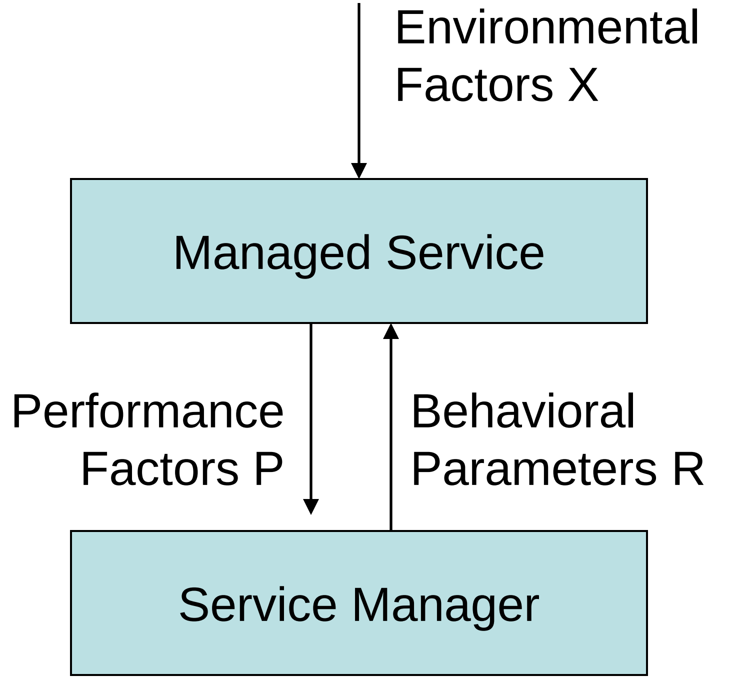
Value and cost

- **Value V(P)**: value of performance P.
- **Cost C(R)**: cost of providing particular resources R.
- Objective function **V(P(R,X))-C(R)**: net reward for service.



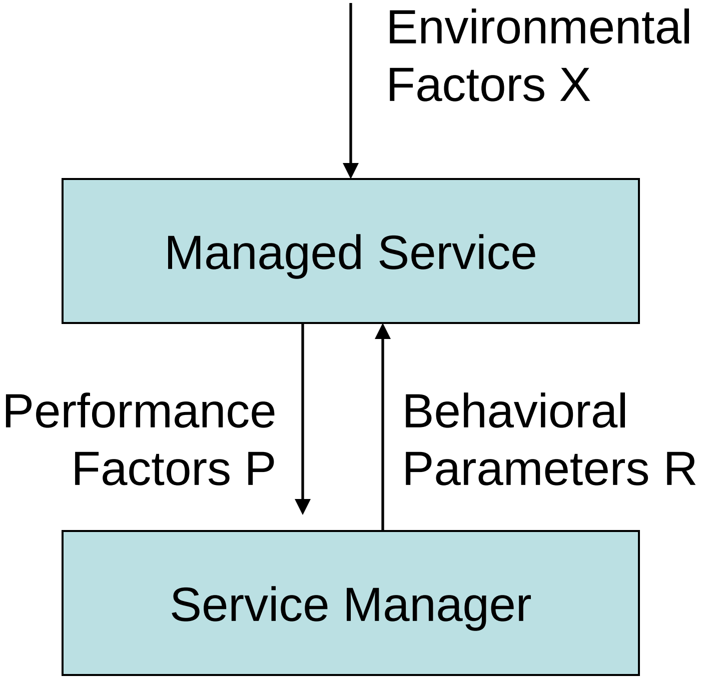
Closed-world approach

- Model X.
- Learn everything you can about it.
- Use that model to maximize V(P(R,X))-C(R).

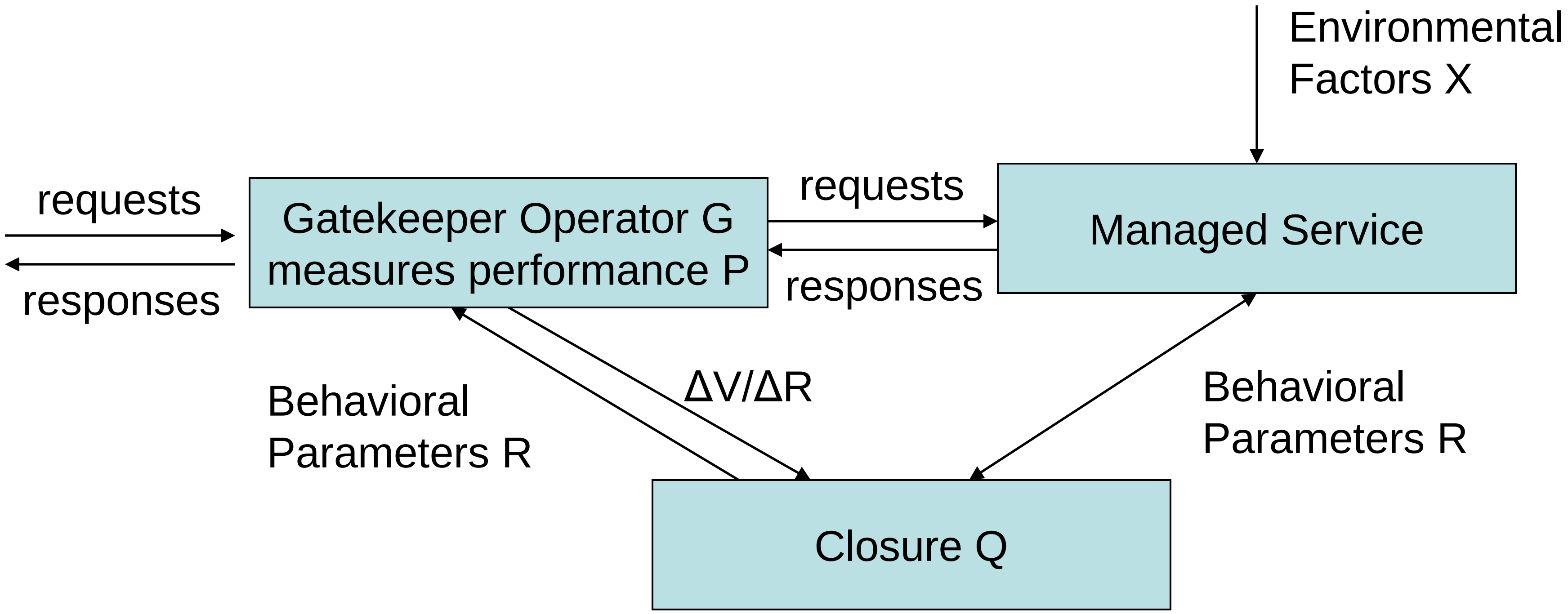


Open-world approach

- X is unknowable.
- Model P(R) rather than P(R,X).
- Use that model to maximize V(P(R))-C(R).
- Maintain agility by using short-term data.



An open-world architecture



- **Immunize R** based upon partial information about P(R,X).
- Distributed agent G knows V(P), predicts **changes in value** ΔV/ΔR.
- Closure Q
  - knows C(R),
  - computes ΔV/ΔR-ΔC/ΔR, and
  - increments or decrements R.

Key differences from traditional control model

- Knowledge is **distributed**.
  - Q knows **cost but not value**
  - G knows **value but not cost**.
  - There can be multiple, distinct concepts of value.
- We **do not model X** at all.

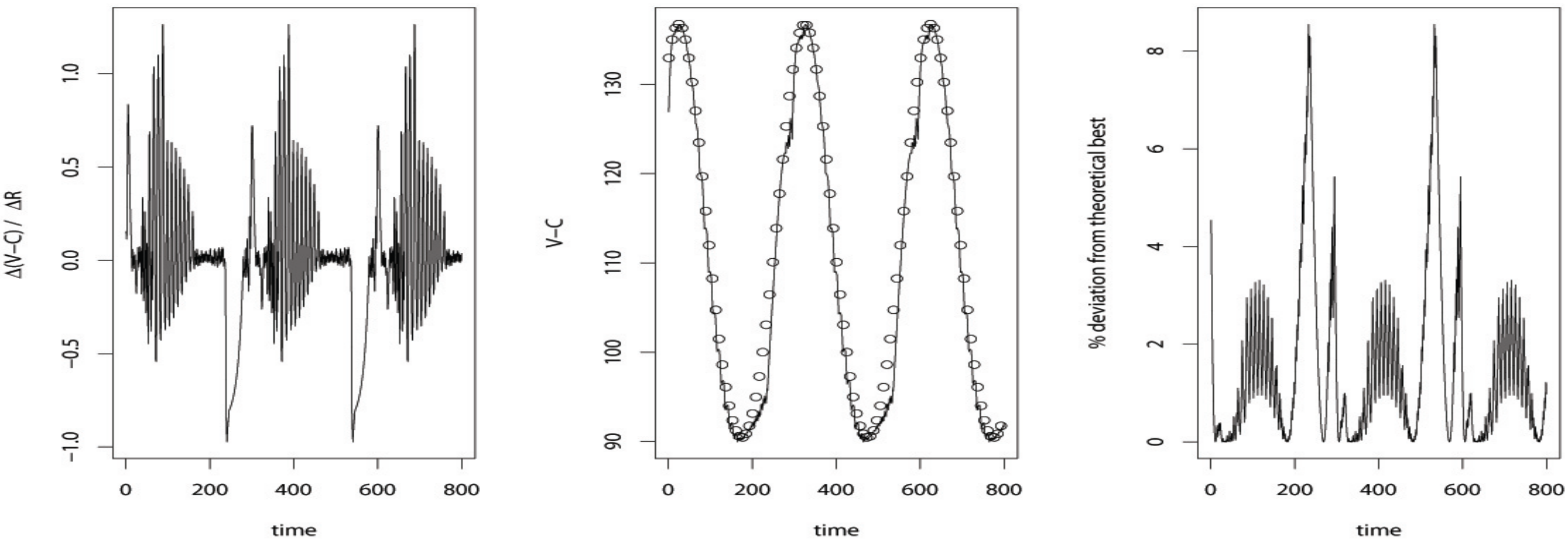
A simple proof-of-concept

- We tested this architecture via simulation.
- Scenerio: cloud elasticity.
- Environment X = sinusoidal load function.
- Resource R = number of servers assigned.
- Performance (response time) P = X/R.
- Value V(P) = 200-P
- Cost C(R) = R
- Objective: maximize V-C, subject to $1 \le R \le 1000$
- Theoretically, objective is achieved when $R=X^{1/2}$

Some really counter-intuitive results

- Q sometimes **guesses wrong,** and is only **statistically correct**.
- Nonetheless, Q can keep V-C **within 5% of the theoretical optimum** if tuned properly, while remaining highly adaptive to changes in X.

A typical run of the simulator



- Δ(V-C)/ΔR is stochastic (left).
- V-C closely follows ideal (middle).
- Percent differences from ideal remain small (right).

Naïve or clever?

- One reviewer: Naïve approaches sometimes work..
- My response: This is not naïve. Instead, it avoids **poor assumptions** that **limit responsiveness.**

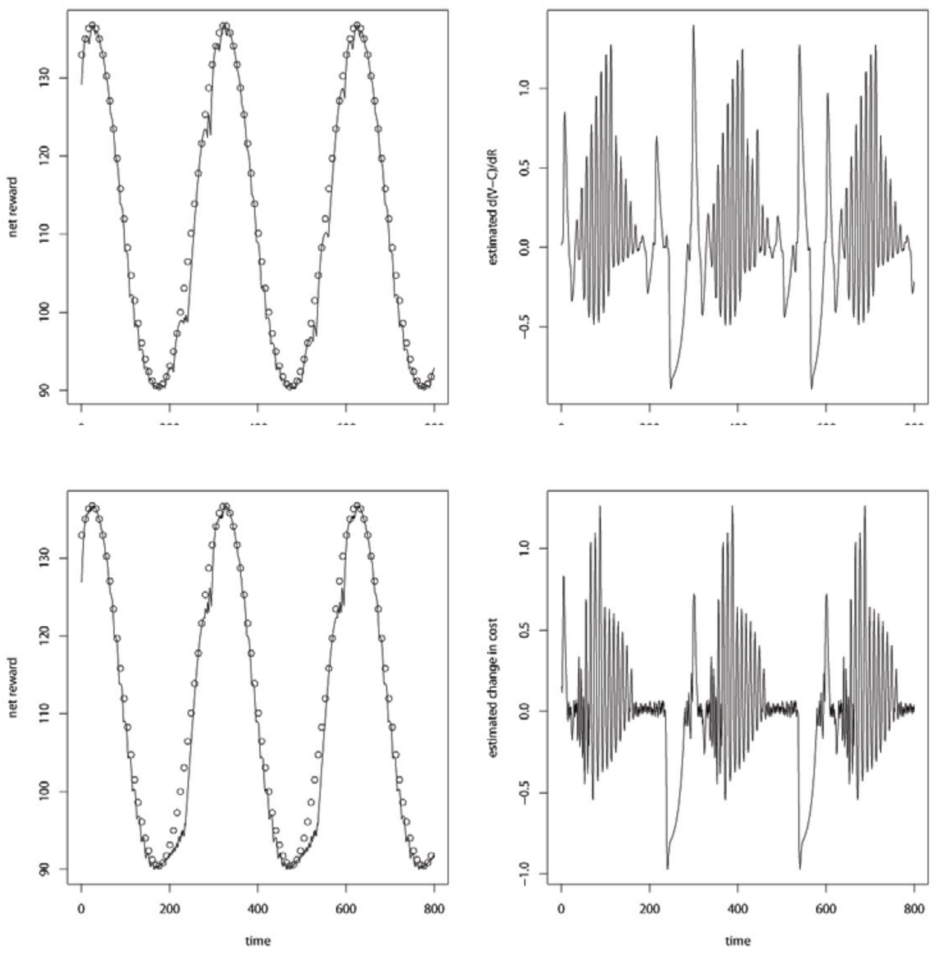
Parameters of the system

- Increment **ΔR**: the amount by which R is incremented or decremented.
- Window **w**: the number of measurements utilized in estimating $\Delta V/\Delta R$.
- Noise **σ**: the amount of noise in the measurements of performance P.

Tuning the system

- The accuracy of the estimator that G uses is **not critical.**
- The window w of measurements that G uses is **not critical, (**but larger windows **magnify** estimation errors!)
- The increment ΔR that Q uses is a **critical parameter** that affects how closely the ideal is tracked.
- **This is not machine learning!!!**

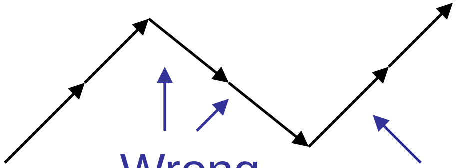
Model is not critical



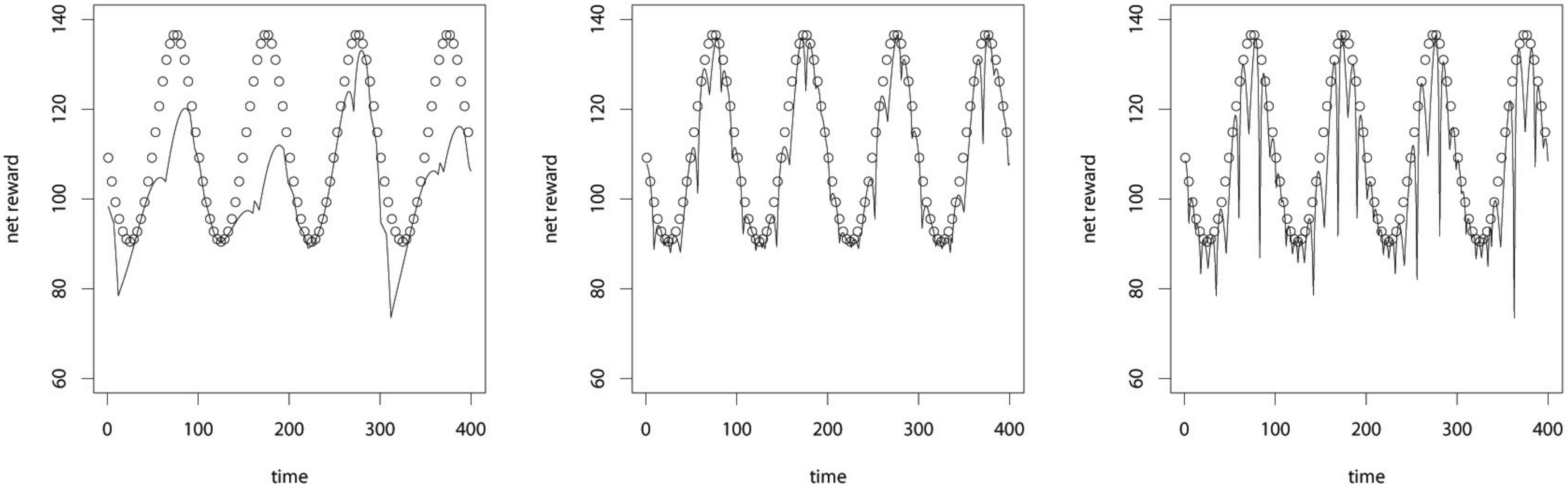
- Top run fits V=aR+b so that ΔV/ΔR≈a, bottom run fits to more accurate model V=a/R+b.
- Accuracy of G's estimator is **not critical**, because estimation errors from unseen changes in X dominate errors in the estimator!

Why Q guesses wrong

- We don't model or account for X, which is changing.
- Changes in X cause **mistakes in estimating ΔV/ΔR**, e.g., load goes up and it appears that value is going down with increasing R.
- These mistakes are **quickly corrected**, though, because when Q acts incorrectly, it gets almost instant feedback on its mistakes from G.

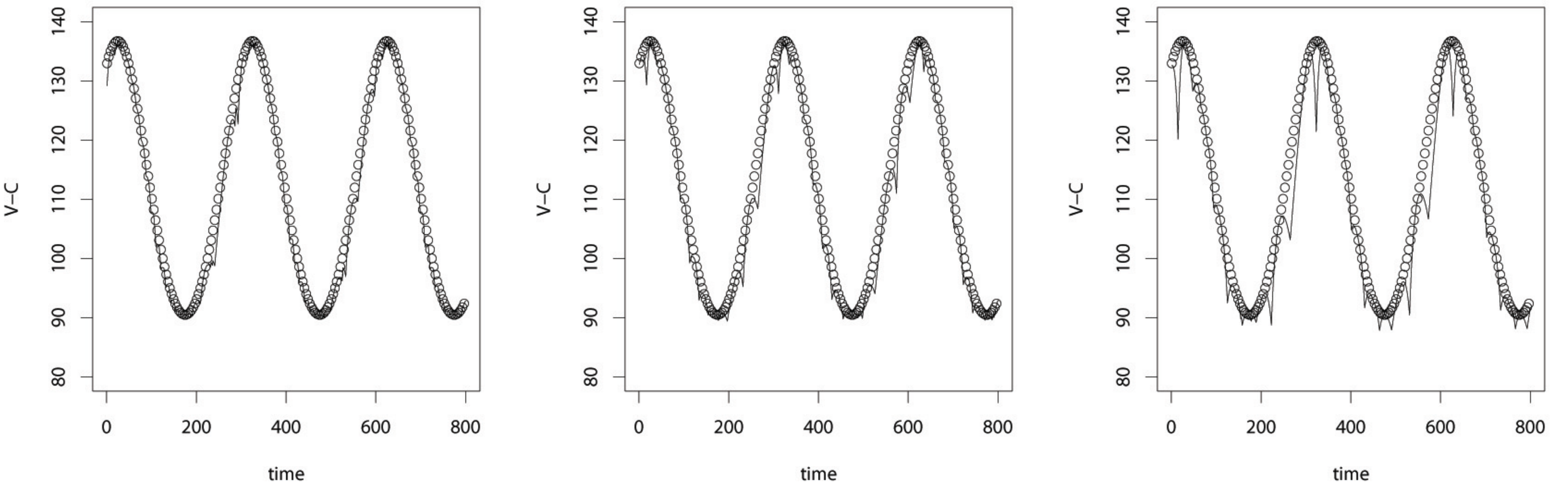


Increment ΔR is critical



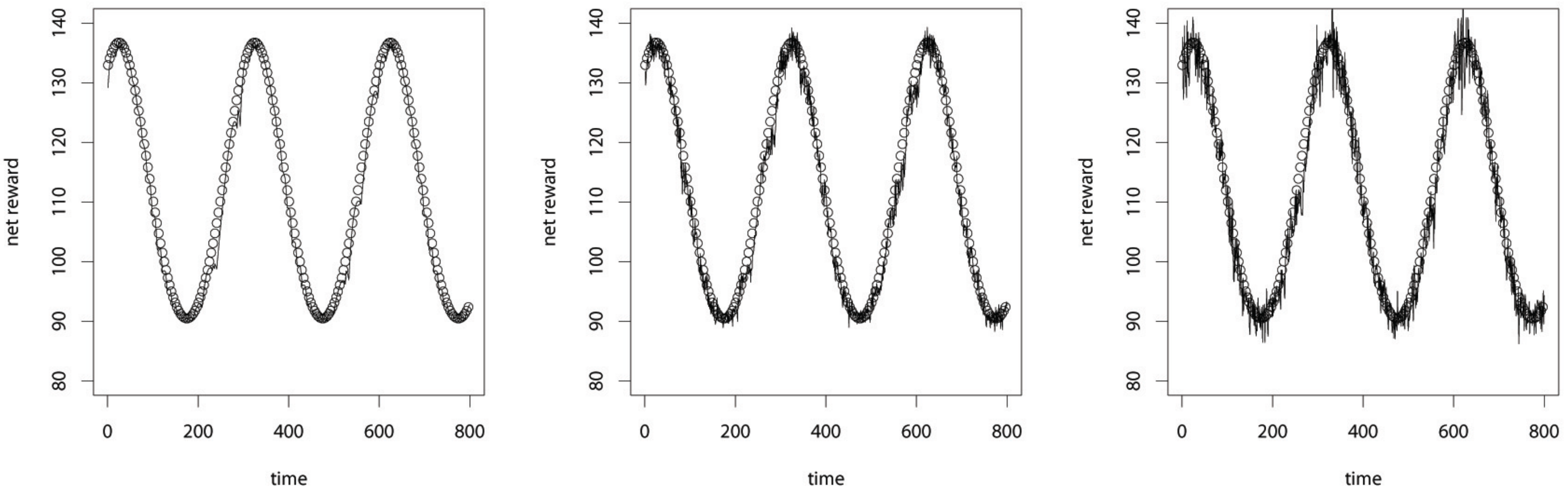
- Plot of time versus V-C.
- ΔR=1,3,5
- ΔR too small leads to undershoot.
- ΔR too large leads to overshoot and instability.

Window w is less critical



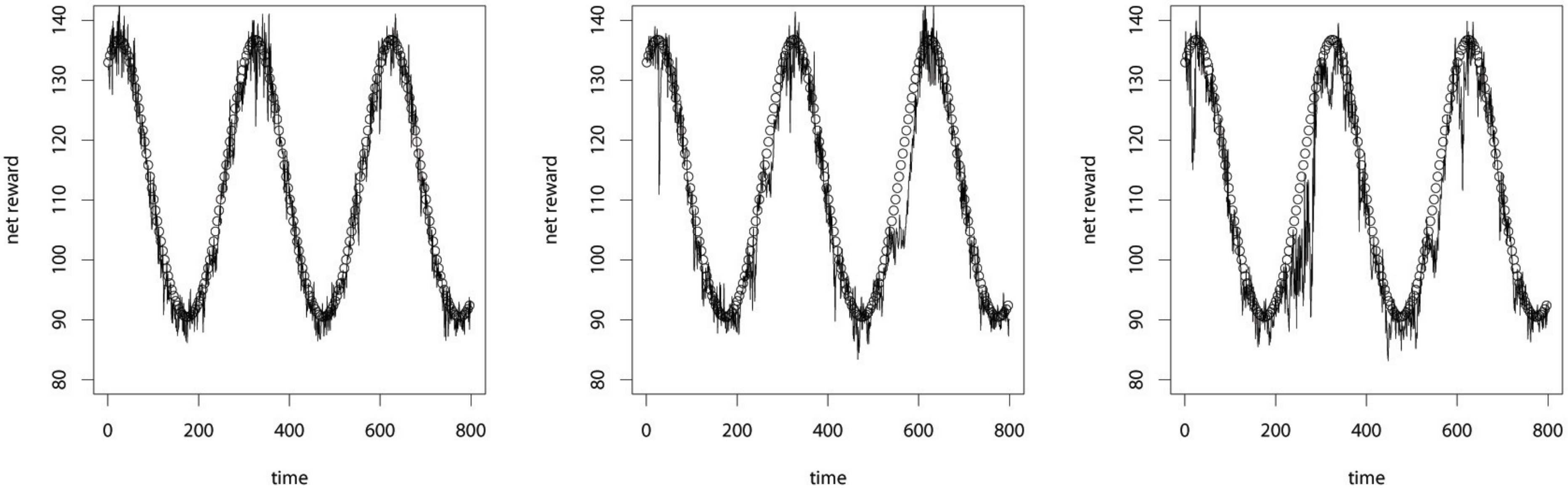
- Plot of time versus V-C.
- Window w=10,20,30
- Increases in w **magnify errors in judgment** and decrease tracking.

0%, 2.5%, 5% Gaussian Noise



- Plot of time versus V-C.
- Noise does not significantly affect the algorithm.

w=10,20,30; 5% Gaussian Noise



- Plot of time versus V-C.
- Increasing window size increases error due to noise, and does not have a smoothing effect.

Limitations

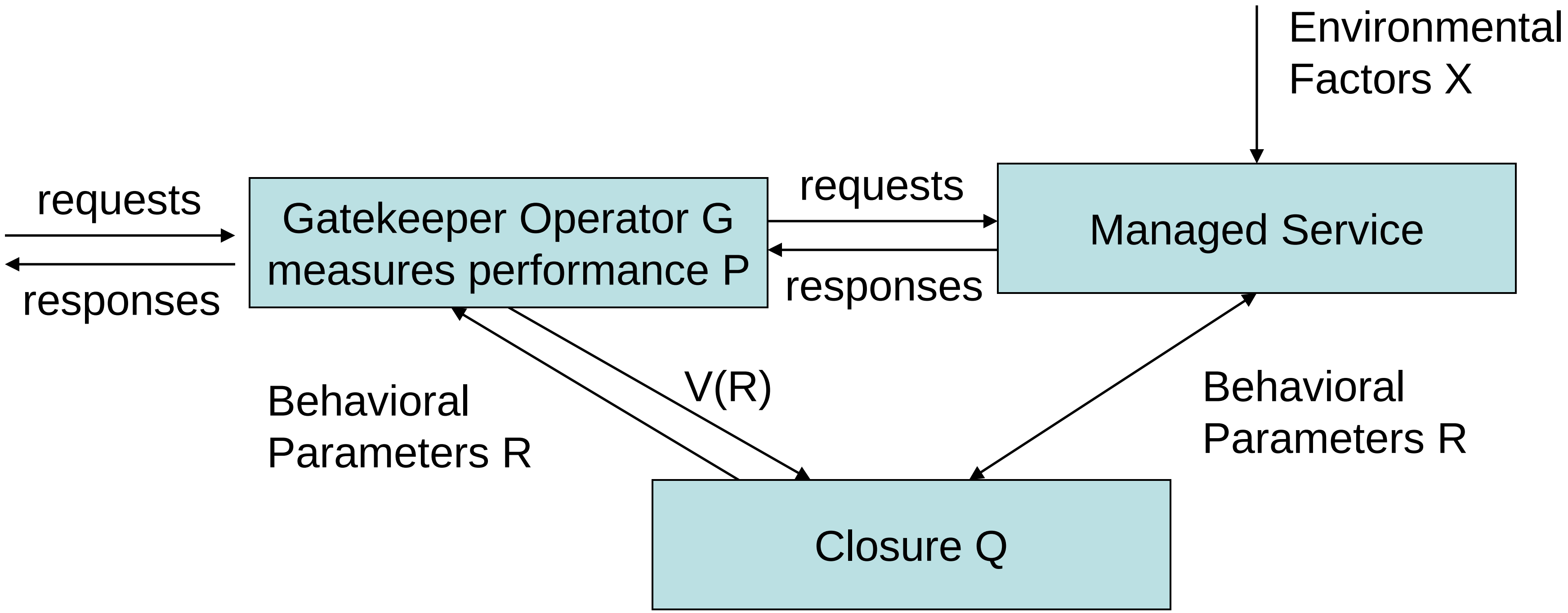
For this to work,

- One must have a reasonable concept of cost and value for R.
- V, C, and P must be simply increasing in their arguments (e.g., $V(R+\Delta R)>V(R)$)
- $V(P(R))-C(R)$ must be convex (i.e., a local maximum is a global maximum)

Modeling SLAs

- **SLAs are step functions describing value.**
- Cannot use an incremental control model.
- Must instead estimate the total value and cost functions.
- Model of static behavior **becomes critical**.

Handling step-function SLAs



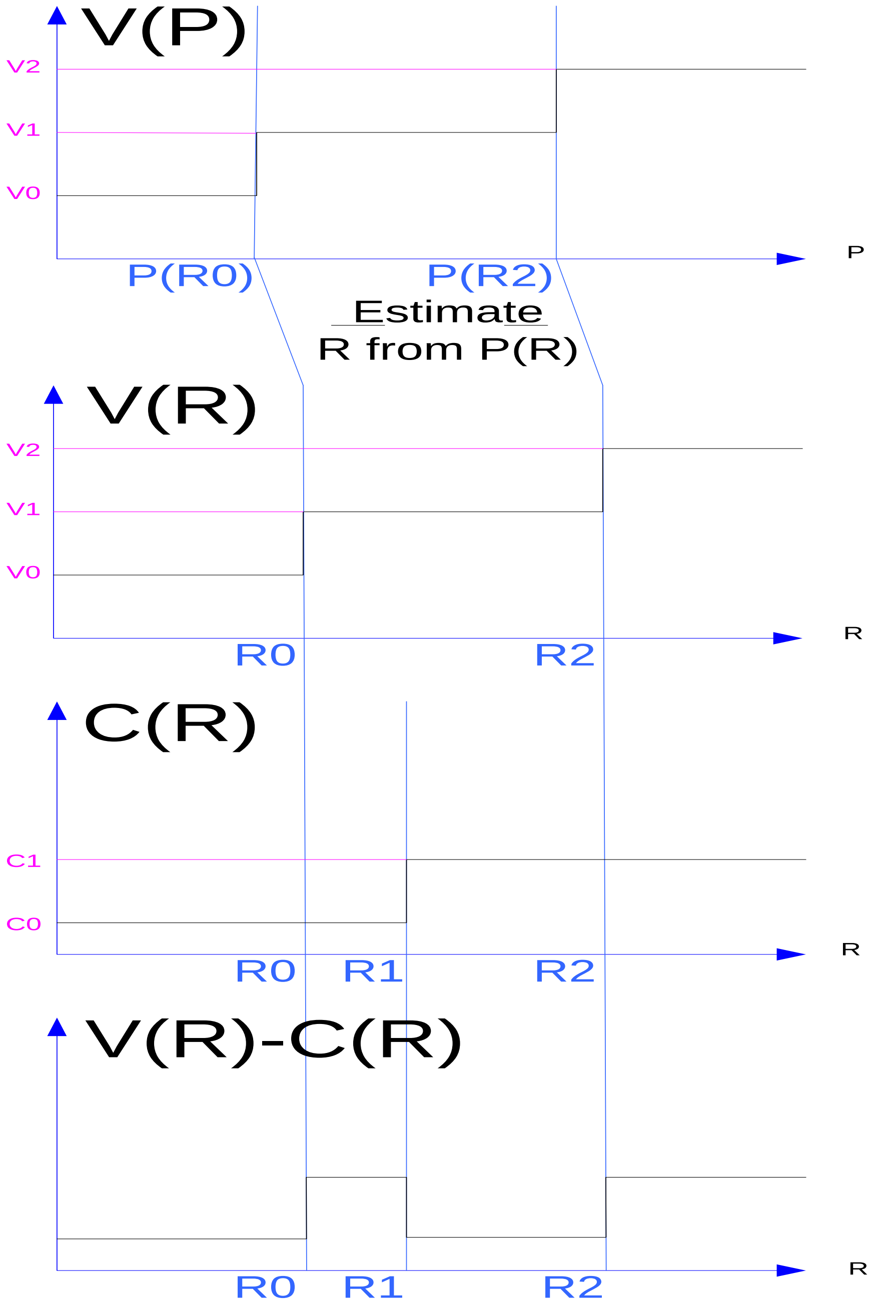
- Distributed agent G knows V(P), R; predicts **value** V(R).
- Q knows C(R), maximizes V(R)-C(R) by incrementally changing R.

Maximizing a step function

- Compute the estimated (V-C)(R) and the resource value at which it achieves its maximum R_{max}.
- If $R>R_{max}$, decrease R.
- If $R<R_{max}$, increase R.

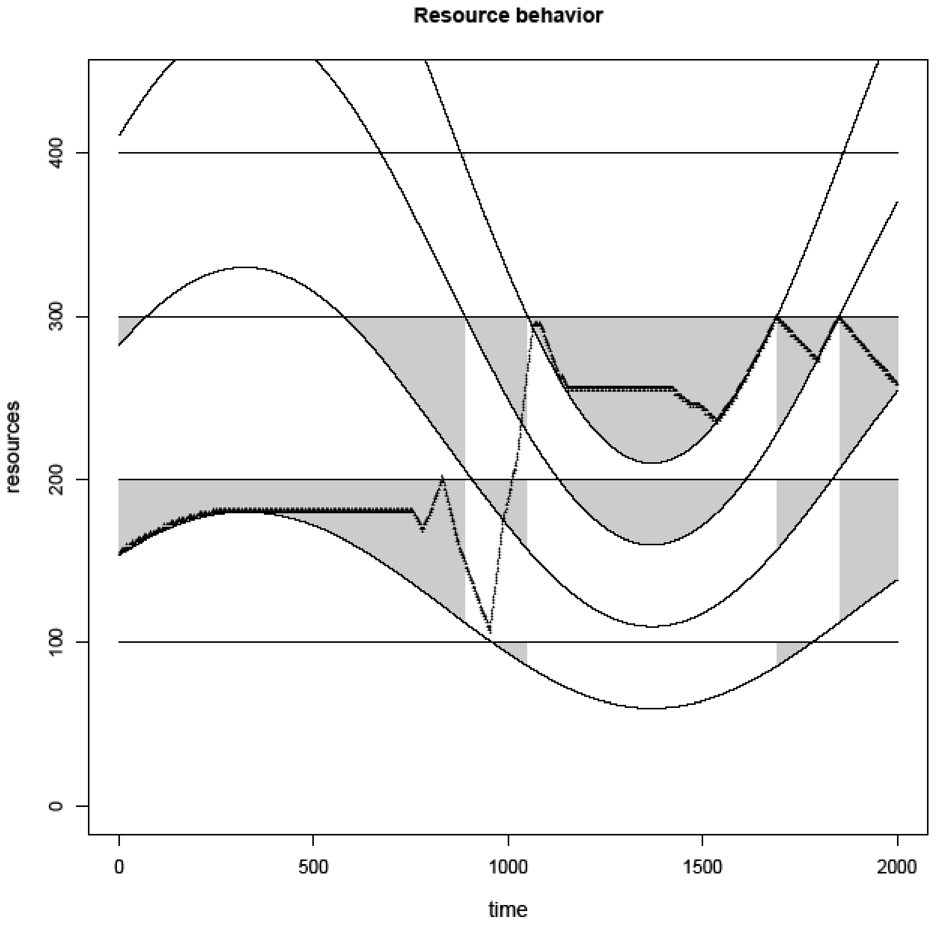
Estimating V-C

- Estimate R from P.
- Estimate V(R) from V(P).
- Subtract C(R).
- Levels V0, V1, V2, C0, C1 and cutoff R1 **do not change**.
- R0, R2 **change over time** as X and P(R) change.



Level curve diagrams

- Horizontal lines represent (constant) **cost cutoffs**.
- Wavy lines represent (varying) theoretical **value cutoffs**.
- Best V-C only changes at times where **a value cutoff crosses a cost cutoff.**
- Regions between lines and between crossovers represent **constant V-C.**
- Shaded regions are areas of **maximum V-C.**

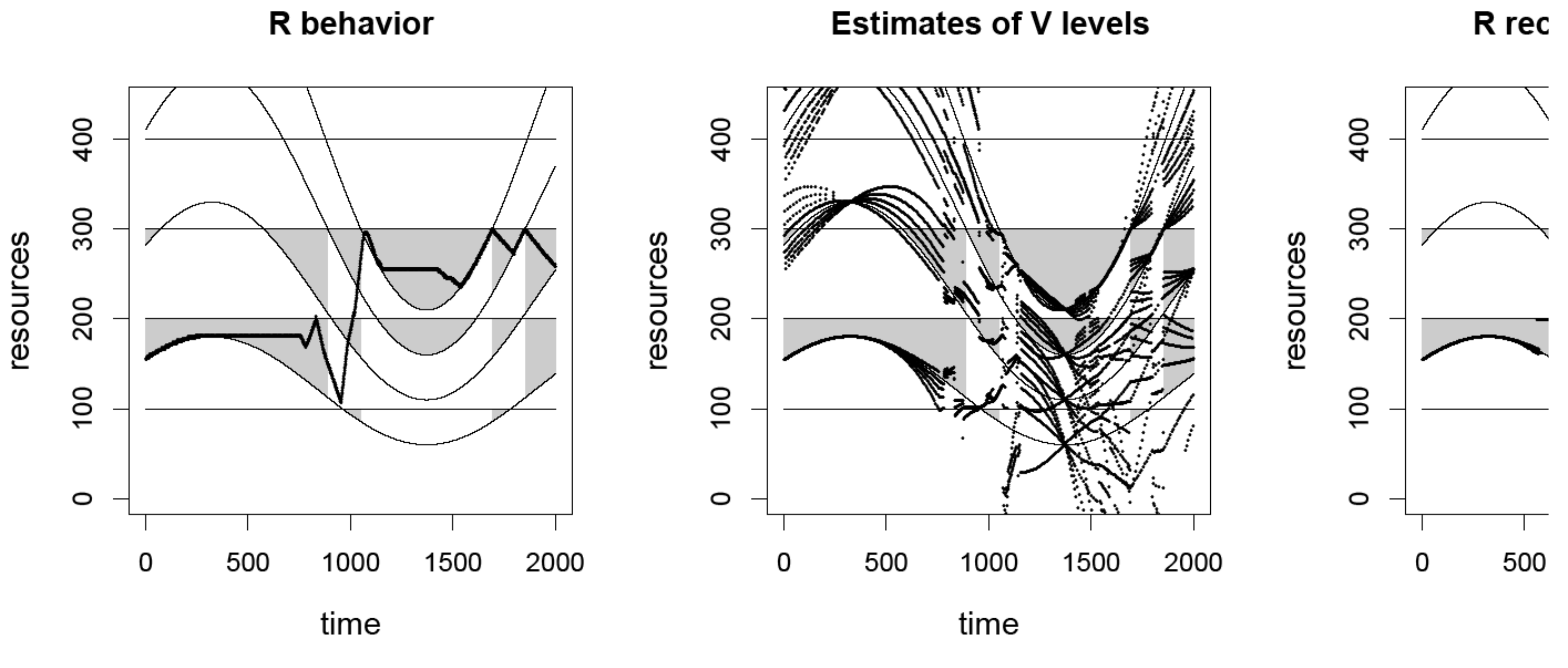


Maximizing V-C

- Two approaches
  - Estimate whole step-function V-C.
  - Estimate “nearest-neighbor” behavior of V-C

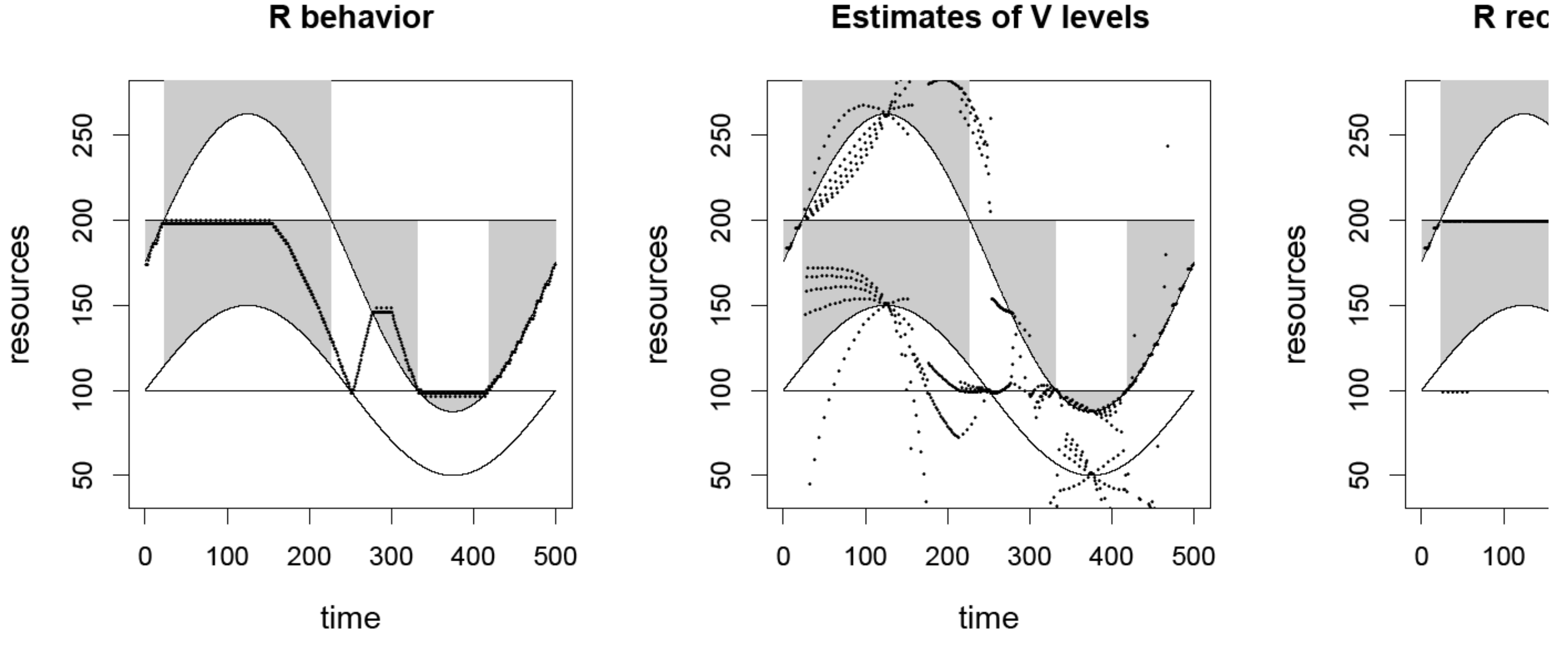
Estimating value cutoffs

- Accuracy of P(R) estimate **decreases with distance** from current R value.
- Choice of model for P(R) is **critical.**
- V-C **need not be convex in R.**



Estimating nearest-neighbor value cutoffs

- Estimate the **two steps** of V(R) around the current R.
- Fitted model for P(R) is **not critical**.
- **V-C must be convex in R.**



In other words,

- One can make tradeoffs between convexity of the value-cost function and accuracy!

How do we know how well we are doing?

- In a realistic situation, we don’t know optimum values for R.
- Must estimate ideal behavior.
- Our main tool: statistical variation of the estimated model.

Exploiting variation

- Suppose that your estimate of V-C varies widely, but is sometimes accurate.
- Suppose that on some time interval, the estimate of V-C is accurate **at least once**.
- Then on that interval, max(V-C)≥actual(V-C)
- Define
  - observed efficiency=sum(V-C)/n*max(V-C)
  - Actual efficiency=sum(actual(V-C))/sum(ideal(V-C))

How accurate is the estimate?

- Three-value tiered SLA.
- Sinusoidal load.

.

loadPeriod	optimum	observed	difference
100	0.800000	0.618421	0.181579
200	0.565310	0.453608	0.111702
300	0.751067	0.647853	0.103214
400	0.896478	0.760870	0.135609
500	0.826939	0.728775	0.098164
600	0.857651	0.760732	0.096919
700	0.946243	0.845524	0.100719
800	0.893867	0.807322	0.086545

In this talk, we…

- **Designed** for an open world.
- **Assumed** that behavioral models are **inaccurate** and/or **incomplete**.
- **Mitigated** inaccuracy of models via **cautious action**.
- Traded **time delays** against **potential for inaccuracy.**
- **Exploited unpredictable variation** to estimate efficiency.

You can use this now

- Analyze what is knowable and what is unknowable.
- Avoid assuming predictable behavior for the unknowable.
- It’s fine to have models, provided that one doesn’t believe them!

Yes, we can!

- We can manage without models and still estimate how well we are doing.
- We can utlilize inaccurate models at the cost of having inaccurate estimates of how well management is doing.
- We can compose management systems without chaos, because systems assume an open world in which another system can exist.

But…

- There are many algorithms between the extremes of model-based and model-free control.
- We can model X and P(R,X) and still obtain these benefits…
- … provided that we are willing to stop using models that become **observably incorrect** over time!
- More about this in the next installment (MACE 2009)!

Questions?

Managing the unknowable

MMNS 2009

Dr. Alva L. Couch

Associate Professor of Computer Science

Tufts University

http://www.cs.tufts.edu/~couch

Email: couch@cs.tufts.edu