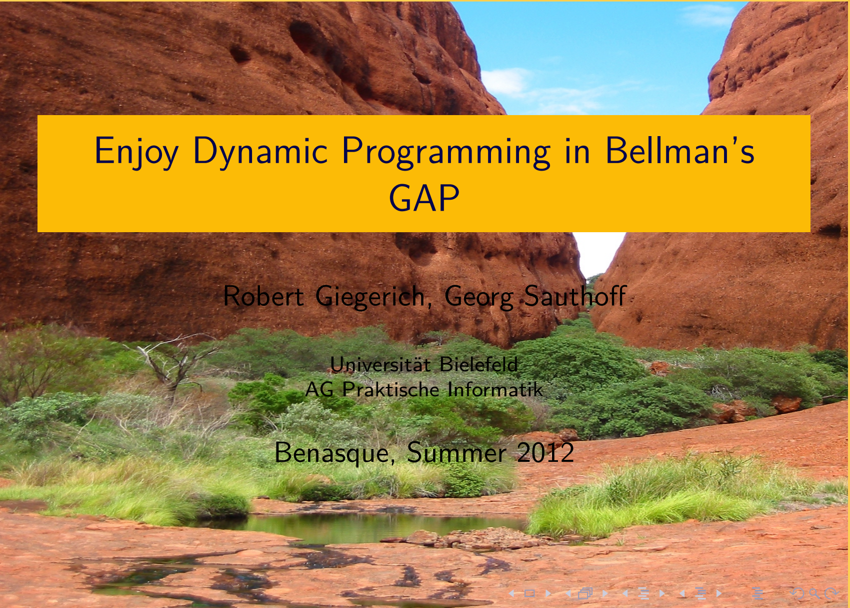
# Enjoy Dynamic Programming in Bellman's GAP



Robert Giegerich, Georg Sauthoff

Universität Bielefeld
AG Praktische Informatik

Benasque, Summer 2012

# Enjoy Dynamic Programming in Bellman's GAP

# Overview

Part 1: Pragmatics

Part 2: Abstractness and modularity in Bellman's GAP

Part 3: Discussion

Part 4: Acknowledgement

# Pragmatics

1. What is it?
2. Does it work on “real” problems?
3. Is it efficient?
4. Does it run on a Mac?
5. Is it difficult to learn?

# What is it?

Bellman's GAP supports dynamic programming over sequences.

It

- supports a programming method, not a specific application domain
- emphasizes abstractness and modularity
- is quick and clean
- achieves acceptable efficiency
- is a 3rd-gen version of ADP – *algebraic* dynamic programming

Our vision: Community creates libraries of re-usable modules for different application domains.

# Literature

Universität Bielefeld

- Compiler optimization techniques:

  R. Giegerich, G. Sauthoff: Yield grammar analysis in the Bellman's GAP Compiler. *Proc. of Languages, Tools and Applications*, 2011
- Formal semantics of GAP-L:

  G. Sauthoff, S. Janssen, R. Giegerich: Bellman's GAP: a declarative language for dynamic programming. *Proc. of Principles and Practice of Declarative Programming*, pages 29–40, 2011.
- Introduction to bioinformatics community:

  G. Sauthoff, M. Möhl, R. Giegerich: Bellman's GAP for Dynamic Programming in Sequence Analysis. *In revision*

# Real world applications?

Established tools converted to Bellman's GAP:

- PKNOTSRG, PKISS (Pseudoknots, Corinna Theis 2010)
- RNASHAPES (Abstract shape analysis, B. Voss 2004/2006)

New applications done

- RAPIDSHAPES (most likely shapes *only*, S. Janssen)
- RNAFOLD emulated and extended with prob. shape analysis ("Lost in Foldingspace?" Janssen, Steger et al.)
- Flowgram DENOISER a la Reeder/Knight
- G1FOLD (Group-I-intron thermodynamic matcher, A. Töpfer, MSc thesis)

# Lost in folding space



Janssen, Schudoma, Steger, Giegerich, BMC Bioinformatics 12(1), 2011

reference

| prediction | 1:pd | 2:go | 3:RN | 4:No | 5:UN | 6:RN | 7:Ma | 8:Mi | 9:UN | 10:RN | 11:Ov |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1: pdb structure | 0 | 44 | | | | | | | | | |
| 2: gold structure | 30 | 0 | | | | | | | | | |
| 3: RNAfold -d0 | 657 | 633 | 0 | 0 | 224 | 324 | 317 | 317 | 483 | 418 | 417 |
| 4: NoDangle | 631 | 605 | 0 | 0 | 188 | 286 | 284 | 284 | 463 | 362 | 362 |
| 5: UNAfold –nodangle | 701 | 676 | 222 | 186 | 0 | 429 | 418 | 418 | 367 | 416 | 411 |
| 6: RNAfold -d1 | 562 | 531 | 312 | 278 | 423 | 0 | 0 | 0 | 271 | 173 | 171 |
| 7: MacroState | 552 | 521 | 305 | 272 | 412 | 0 | 0 | 0 | 262 | 171 | 169 |
| 8: MicroState | 552 | 521 | 305 | 272 | 412 | 0 | 0 | 0 | 262 | 171 | 169 |
| 9: UNAfold | 593 | 564 | 471 | 451 | 356 | 265 | 256 | 256 | 0 | 294 | 287 |
| 10: RNAfold -d2 | 608 | 572 | 427 | 375 | 428 | 190 | 188 | 188 | 320 | 0 | 0 |
| 11: OverDangle | 606 | 570 | 426 | 375 | 423 | 188 | 186 | 186 | 313 | 0 | 0 |

# Real world applications? (ctd.)

Planned tool conversions

- RNAAlishapes (Shape analysis of aligned RNAs, B. Voss)
- Locomotif (Thermodynamic RNA motif matchers, generated from graphical description, J. Reeder 2007/A. Wittkopf)
- RNAHybrid (miRNA target prediction, M. Rehmsmeier)

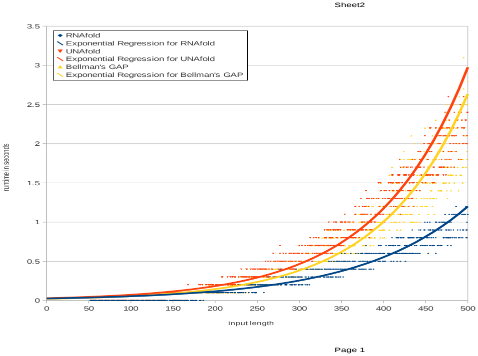
New applications ongoing:

- Ambivalent covariance models (Stefan Janssen)
- Statistical minisatellite alignment (Benedikt Loewes)

# Efficient?

There will always be an abstraction charge, but ...

- Competent code due to substantial optimizations
- Space-efficient due to automated table design and dimension analysis



# Available?

Yes. Open source.

# Available?



Yes. Open source.

- Available under GNU public licencse
- official release paper under revision

Developed under LINUX, (Ubuntu package)
but also runs on Solaris, MacOS (well ....)

# Is it fun?

YES!

- easy to follow on simple examples
- scales well to real world
- we develop ideas rather than debug code
- but ...

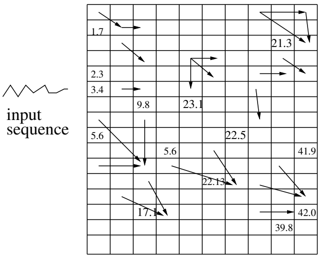
... it breaks with traditional mindset on dynamic programming

# The classical view



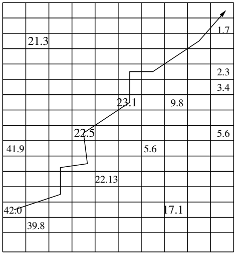
$$Z(k,i,j,x,y) = \sum_{r=i}^{j-\theta-1} \sum_{u,\, v \in N} \sum_{k'=0}^{K} Z(k',i,r-1,x,u) \cdot Z^B(k-k',r,j,v,y) + \sum_{u \in N} Z(k-\sigma_{a_{j-1},u},k,j-1,x,u) \quad (5)$$

$$Z_{M1}(i,j) = \sum_{r=i+\theta+1}^{j} Z_B(i,r) \cdot e^{-c(j-r)/RT} \quad (3)$$

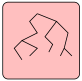


matrix recurrence

41.9
result score



backtrace through matrix



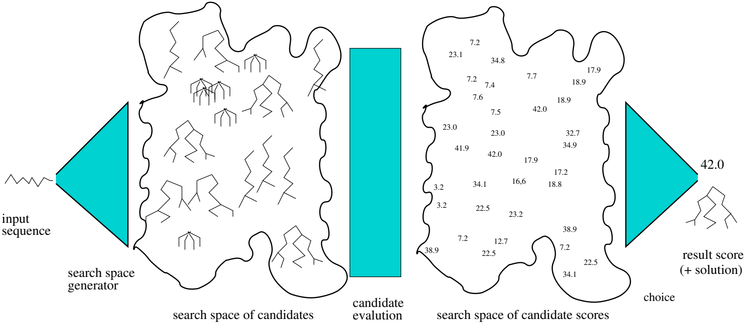
solution
(pretty−print)

# The algebraic view

$$\mathit{struct} \rightarrow \mathit{open}(\mathit{struct}, \mathit{base}) \mid \mathit{split}(\mathit{struct}, \mathit{closed})$$
$$\mathit{closed} \rightarrow \ldots$$

$$\mathit{open}(s, b) = s$$
$$\mathit{split}(s, c) = s * c$$

$$\mathit{choice} = \mathit{max}$$




New:

- perfect separation of search space, scoring, choice
- a data structure for the candidates (!)

# The basis: Bellman's Principle of Optimality

Richard Bellman (1964): *"An optimal solution can be composed solely from optimal solutions to sub-problems."*

That's a requirement, not a theorem!!

Alternative formulations:

- (strict) monotonicity of scoring wrto maxi/minimization:

$$x < y \Rightarrow f(x) < f(y)$$

- distributivity of choice over scoring:

$$h(F(X, Y)) = h(F(h(X), h(Y))$$

- semiring framework (e.g. Pachter/Sturmfels)

# The basis: Bellman's Principle of Optimality

Richard Bellman (1964): *"An optimal solution can be composed solely from optimal solutions to sub-problems."*

That's a requirement, not a theorem!!

Alternative formulations:

- (strict) monotonicity of scoring wrto maxi/minimization:

$$x < y \Rightarrow f(x) < f(y)$$

- distributivity of choice over scoring:

$$h(F(X, Y)) = h(F(h(X), h(Y))$$

- semiring framework (e.g. Pachter/Sturmfels)

In some fixed frameworks, Bellman's Principle is guaranteed to hold, e.g. with SCFGs.

A property of the scoring scheme, not of "the algorithm".

# Bellman's GAP

A third-generation implementation of ADP:

Bellman's Principle
+ Grammars
+ Algebras
+ Products
= Bellman's GAP

# ADP master equation

We design

- an abstract data type $\Sigma$ representing candidates (as trees resp. formulas)
- tree grammar $\mathcal{G}$ defining the problem decomposition and candidate space
- evaluation algebras $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ describing objectives $h_{\mathcal{A}}, h_{\mathcal{B}}, h_{\mathcal{C}} \ldots$

$$\mathcal{G}(\mathcal{A}, x) = h_{\mathcal{A}}([\mathcal{A}(t) | t \in L(\mathcal{G}), \mathit{yield}(t) = x])$$

We compile and call for input $x$

$$\mathcal{G}(\mathcal{A}, x), \quad \mathcal{G}(\mathcal{B}, x), \quad \mathcal{G}(\mathcal{C}, x), \quad \ldots$$

# ADP master equation

We design

- an abstract data type $\Sigma$ representing candidates (as trees resp. formulas)
- tree grammar $\mathcal{G}$ defining the problem decomposition and candidate space
- evaluation algebras $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ describing objectives $h_{\mathcal{A}}, h_{\mathcal{B}}, h_{\mathcal{C}} \ldots$

$$\mathcal{G}(\mathcal{A}, x) = h_{\mathcal{A}}([\mathcal{A}(t) | t \in L(\mathcal{G}), \mathit{yield}(t) = x])$$

We compile and call for input $x$

$$\mathcal{G}(\mathcal{A}, x), \quad \mathcal{G}(\mathcal{B}, x), \quad \mathcal{G}(\mathcal{C}, x), \quad \ldots$$

$$\mathcal{G}(\mathcal{A} * \mathcal{B}, x)$$

$$\mathcal{G}(\mathcal{A} \otimes \mathcal{B} \times \mathcal{C}, x)$$

# Cartesian ( $\times$) and lexicographic ($*$) product

$$f_{A\times B}((a,b),z) = (f_A(a,z), f_B(b,z))) \tag{1}$$

$$h_{A\times B}(as,bs) = (h_A(as), h_B(bs)) \tag{2}$$

$$f_{A*B} = f_{A\times B} \tag{3}$$

$$h_{A*B}[(a_1,b_1),\ldots,(a_m,b_m)] = [\,(l,r) \mid l \leftarrow \mathrm{set}(h_A[a_1,\ldots,a_m]),\ r \leftarrow h_B[\,r' \mid (l',r') \leftarrow [(a_1,b_1),\ldots,(a_m,b_m)],\ l' = l\,]\ ] \tag{4}$$

# Cartesian ( $\times$) and lexicographic ($*$) product



$$f_{A\times B}((a,b),z) = (f_A(a,z), f_B(b,z))) \quad (1)$$

$$h_{A\times B}(as,bs) = (h_A(as), h_B(bs)) \quad (2)$$

$$f_{A*B} = f_{A\times B} \quad (3)$$

$$\begin{aligned} h_{A*B}[(a_1,b_1),\ldots,(a_m,b_m)] = [\,(l,r) \mid \\ l \leftarrow \mathrm{set}(h_A[a_1,\ldots,a_m]), \\ r \leftarrow h_B[\,r' \mid (l',r') \leftarrow [(a_1,b_1),\ldots,(a_m,b_m)],\ \ l' = l\,]\ \ ] \end{aligned} \quad (4)$$

| | | |
|---|---|---|
| $rnafold(mfe * print, x)$ | = | [(−42.0, "(((((..((....)))).(((.....))))))"), (42.0, . . .), . . .] |
| $rnafold(mfe * count, x)$ | = | [(−42.0, 3)] |
| $rnafold(bpmax * mfe * print, x)$ | = | [(11, −41.3, "(((((..((....)))).((.((...))))))")] |
| $rnafold(shape * pf, x)$ | = | [([][], 0.73), ([[][]], 0.21), ...] |
| $rnafold((mfe * print) \times (probs * print), x)$ | = | your guess? |
| and why not | = | $rnafold((shape * (mfe\ .\ print) \times (probs\ .\ print)), x)$ ? |

# What was this?

Given some independent algebras over the same signature,

Bellman's GAP contributes

- reporting candidates via backtracing
- counting co-optimals (and anything else)
- optimization under lexicographic orderings
- classified dynamic programming

# Interleaved ($\otimes$) and overlay ($|$) product

$$f_{A \otimes B} = f_{A \times B} \tag{5}$$

$$\begin{aligned} & h_{(A \otimes B)(k)}[(a_1, b_1), \ldots, (a_m, b_m)] = \\ & \quad [\,(l, r) \mid (l, r) \leftarrow U, p \leftarrow V, p = r] \\ & \text{where} \\ & \quad U = h_{A * B(1)}[(a_1, b_1), \ldots, (a_m, b_m)] \\ & \quad V = \mathrm{set}(h_{B(k)}[\, v \mid (\_, v) \leftarrow U]) \end{aligned} \tag{6}$$

$$A \mid A_{h:=id} = A * A_{h:=id} \quad (\text{ compile with --sample}) \tag{7}$$

# Interleaved ($\otimes$) and overlay ($|$) product



$$f_{A \otimes B} = f_{A \times B} \quad (5)$$

$$\begin{aligned} & h_{(A\otimes B)(k)}[(a_1,b_1),\ldots,(a_m,b_m)] = \\ & \quad [\,(l,r) \mid (l,r) \leftarrow U, p \leftarrow V, p = r] \\ & \text{where} \\ & \quad U = h_{A*B(1)}[(a_1,b_1),\ldots,(a_m,b_m)] \\ & \quad V = \mathrm{set}(h_{B(k)}[\,v \mid (\_,v) \leftarrow U]) \end{aligned} \quad (6)$$

$$A \mid A_{h:=id} = A * A_{h:=id} \qquad (\text{ compile with --sample}) \quad (7)$$

| | | |
|---|---|---|
| $rnafold(shape \otimes mfe(3), x)$ | = | [ ([ ] [ ], −32.0, ), ([ ], −31.8), ([[][]], −31.1) ] |
| $rnafold(pf \mid pf\_id * shape, x)$ | = | [(1000,2,[[][]]), (1000,4,[ ][ ]), (1000,4,[ ][ ]), (1000,3,[[][]]), .... ] |
| $rnafold(print * count, x)$ | = | your guess? |

# What was this?

Products allow for

- optimizing *across* a classification
- stochastic sampling
- simple ambiguity testing

# Bellman's Principle as a proof obligation

All algebras must satisfy Bellmans's Principle of Optimality
alias distributivity,
alias (strict) monotonicity.

Products are always *defined*, but they may not preserve Bellman's Principle.

This generates *proof obligations* ...

# Rewards from abstractness and modularity

With Bellman's GAP,

- we focus on the creative part – designing signatures, grammars, and algebras ...
- ... and generic alphabets, and multi-tape scenarios ...
- we combine them with products in practically unlimited variety,
- obtain useful implementations without low-level coding and debugging.

That's fun.

See you soon in Bellman's GAP Cafe at URL gapc.eu (preliminary web site)

# Limitations of ADP and Bellman's GAP

Types of problems that do not fit:

- problems where recursion is trivial (and everything interesting happens in the scoring scheme),
- KNAPSACK or operations research-type DP problems, evolving a complex state variable over time
- problems on trees, graphs,
- generalized grammars as in the $O(n^6)$ time, $O(n^4)$ space algorithm PKNOTS (Rivas & Eddy)
- tricky tabulation schemes as in talks by Chitsaz, Stadler on interaction

# Limitations of ADP and Bellman's GAP

Types of problems that do not fit:

- problems where recursion is trivial (and everything interesting happens in the scoring scheme),
- KNAPSACK or operations research-type DP problems, evolving a complex state variable over time
- problems on trees, graphs,
- generalized grammars as in the $O(n^6)$ time, $O(n^4)$ space algorithm PKNOTS (Rivas & Eddy)
- tricky tabulation schemes as in talks by Chitsaz, Stadler on interaction

Planned extension:

- Automated support for semantic ambiguity checking (ACLA at `http://www.brics.dk/grammar/`)

# Acknowledgement



Many have contributed to ADP in the past decade.

Most recent contributors:

- Georg Sauthoff created Bellman's GAP-L and GAP-C
- Stefan Janssen creates applications
- Mathias Moehl (Freiburg): teaching and extensions
- Christian Höner zu Siederdissen: Haskell-ADP

# 10 year Acknowledgement

| | | |
|---|---|---|
| Grammar | $\mathcal{W}$ | analyzes scientific meetings |
| Algebras | *Reward* | maximizes over inspiration and impact |
| | *Ambiente* | evaluates site and surrounding |
| | *Orgs* | converts organizers to ASCII |
| Data | $bioinfo_{all}$ | all bioinformatics meetings (past 10 yrs) |

$$\mathcal{W}((Reward * Ambiente) * Orgs, bioinfo_{all}) =$$

# 10 year Acknowledgement

| | | |
|---|---|---|
| Grammar | $\mathcal{W}$ | analyzes scientific meetings |
| Algebras | *Reward* | maximizes over inspiration and impact |
| | *Ambiente* | evaluates site and surrounding |
| | *Orgs* | converts organizers to ASCII |
| Data | $bioinfo_{all}$ | all bioinformatics meetings (past 10 yrs) |

$$\mathcal{W}((Reward * Ambiente) * Orgs, bioinfo_{all}) =$$

```
[
((1.0, "Benasque RNA 2003"), (Elena Rivas, Eric Westhof))
((1.0, "Benasque RNA 2012"), (Elena Rivas, Eric Westhof))
((1.0, "Benasque RNA 2006"), (Elena Rivas, Eric Westhof))
((1.0, "Benasque RNA 2009"), (Elena Rivas, Eric Westhof))]
```

*Thank you!*


Hunting RNA motifs
Paul Gardner