Modelling Languages, Model Expansion, and MXG

David G. Mitchell

Computational Logic Laboratory
School of Computing Science
Simon Fraser University

April 29 2008

Outline

Outline

Combinatorial Search Problems

Occur Widely in Applications:

- Operations: Timetabling, Scheduling, Planning, Routing, ...
- HW/SW: Equivalence, Verification (bug finding), test generation ...
- Bioinformatics: phylogeny, haplotyping, protein folding, ...
- ...

Many are hard - solving is expensive:

- NP-hard, and hard in practice
- good application-specific implementations expensive

Solve by reduction

E.g., Solve NP-search problems by reduction to SAT (say)

SAT Solving: Good News

SAT solver technology is effective for a growing collection of problems

SAT Solving Methodology

1. Design a reduction from given problem to SAT
2. Implement reduction (say with a C program)
3. Run off-the-shelf SAT solver(s)

Plenty more to do, but "Ground Solver" work mature and healthy:

- theory (proof complexty, tractable cases, etc.)
- impressive implementations (SAT, SMT, CSP, ASP, etc.)
- experimentation
- application successes
- steady progress

SAT Solving: Bad News

Refinement of reductions is needed in practice:

- multiple iterations of 2 and 3
- time-consuming, error-prone process
- for specialists

"Front-End" tools would be useful

- modelling languages – to model *problems*
- automatic reduction to SAT – i.e., grounding
- debugging aids
- ...

Modelling Languages

Solvers could be exploited more fully with "front end" modelling languages

Alternate Methodology

1. Construct a Declarative Problem Model
2. Grounder produces SAT instance from Model and Instance
3. Run SAT solver

- speed up development time;
- facilitate more experimentation;
- reduce errors
- improve performance (?!)

Example: Social Golfer Problem

Given $g \times s$ golfers and w weeks,
find a schedule for golfers to play:

- w games
- in s groups of size g
- such that not two golfers play together twice.

Social Golfer Problem in Essence

```
given    w,g,s:int ( 1..  )
letting  golfers be new type of size g × s
find     schedule:set (size w)
            of partition (numParts g, partSize s) from golfers
such that
         ∀ week1, week2 ∈ sched . week1 != week2 →
            ∀ group1 ∈parts(week1), group2 ∈parts(week2) .
               |group1 ∩ group2| < 2
```

Example: Social Golfer Problem, cont'd

Social Golfer Problem in ESRA

cst weeks, groups, groupsize : N

dom$Players = 1..groups \times groupsize$,

$Weeks = 1..weeks$, $Groups = 1..groups$;

var $Sched : (Players \times Weeks) \rightarrow^{groupsize \times weeks} Groups$

solve

$\forall(h : Groups, w : Weeks)count(groupsize)(p : Players | Sched(p, w) = h) \wedge$

$\forall(p1 < p2 : Players)count(0...1)(w : Weeks | Sched(p1, w) = Sched(p2, w))$

Social Golfer Problem in NP-Spec

SPECIFICATION

IntFunc({1..groups×groupsize} >< {weeks}, Schedule, 1..groups).

fail ← COUNT(Schedule(*,W,Gr),X), X != groupsize.

fail ← Schedule(P1,W1,Gr1), Schedule(P2,W1,Gr1), P1 != P2,
Schedule(P1,W2,Gr2), Schedule(P2,W2,Gr2), W1 != W2.

Modelling Languages: Bad News

Recognized need $\Rightarrow$ many projects:

- "Constraint Modelling" languages: Essence, ESRA, OPL, ...
- Answer Set Programming languages: DLV, Lparse, ...
- FO(ID)-based languages: MXG, IDP,
- NP-Spec & Spec2SAT, ASPPS, ...

But rather ad hoc: Modelling language work relatively immature.

As a field:

- little or no theory,
- little analysis or careful experimentation
- few tools and techniques
- few good implementations
- few applications

Yet many hard problems.

Outline

Formal Foundation: Model Expansion

Claim: we should start with a formal foundation

Model Expansion for Logic $\mathcal{L}$ ($\mathcal{L}$-MX)

`Given:`

- finite σ-structure $\mathcal{A} = (A; \sigma^{\mathfrak{A}})$,
- formula ϕ of logic $\mathcal{L}$, with vocabulary $\sigma \cup \varepsilon$

`Find:`

- $\sigma \cup \varepsilon$-structure $\mathcal{B} = (A; \sigma^{\mathfrak{A}}, \varepsilon^{\mathfrak{B}})$, s.t. $\mathfrak{B} \models \phi$
  (i.e., An expansion of $\mathcal{A}$ that satisfies ϕ)

- Expansion of a structure a standard notion in model theory
- MX as as a task proposed in (M.-Ternovska'05) as formal basis for declarative programming for search problems:
  - Problem instance: finite structure $\mathcal{A}$,
  - Formula ϕ specifies relation between instances and solutions,

MX and Declarative Programming

Graph Colouring as FO-MX

- Instance: Graph $\mathcal{A} = G = (Vtx \cup Clr; Edge^G)$,
- Expansion Vocabulary: $Colour$ (where $Colour : Vtx \longrightarrow Clr$).
- Problem specification:
$$\phi = \forall x \forall y (Edge(x, y) \supset Colour(x) \neq Colour(y))$$
- Solutions: Interpretations for $Colour$ such that
$$\mathcal{B} = (Vtx \cup Clr; Edge^G, Colour^B) \models \phi$$

- ϕ is a problem specification.
- $\mathcal{L}$ can be tailored to:
  - what problems we want to express;
  - natural ways to express problems of interest.
- Every computational search problem can be formalized as MX, (for suitable $\mathcal{L}$).
- Most existing languages naturally viewed as MX-specifications.

Descriptive Complexity Theory

Characterizes complexity classes in terms of logics:

- NP = $\exists SO$-definable finite structures (Fagin'74)
- PH = SO-definable finite structures
- P = LFP-definable ordered finite structures (Immerman-Vardi)

Suggests logics as declarative programming languages for complexity classes

- FO-MX captures NP
- Conventional Wisdom: not practically useful
- We strongly disagree
- Constraint Modelling languages are doing it
  although many authors don't know it
- E.g., languages describe MX tasks in (syntactic variants of extensions of) standard logics. *(***M***–Ternovska '08 examines ESSENCE)*,

Two Questions

Why "FO-MX" not ∃SO?

- emphasize we must witness the SO variables
- we are interested in MX for various logics: e.g.,
  - GF_k(ID)-MX
  - Π_2(ID)-MX

Why do we care about "capturing" complexity classes?

- provide assurance the user can model their problem
- provide assurance of "implementability", e.g.:
  - we want polytime grounding for problems in NP;
  - the above languages do not provide it!

Outline

The MX Project

- Joint work with E. Ternovska, plus students, post-docs and others
- Main Goal: Build practical MX-based tools for solving search problems.

Project tennets:

- Logic from top to bottom

  Classical logic, as far as possible

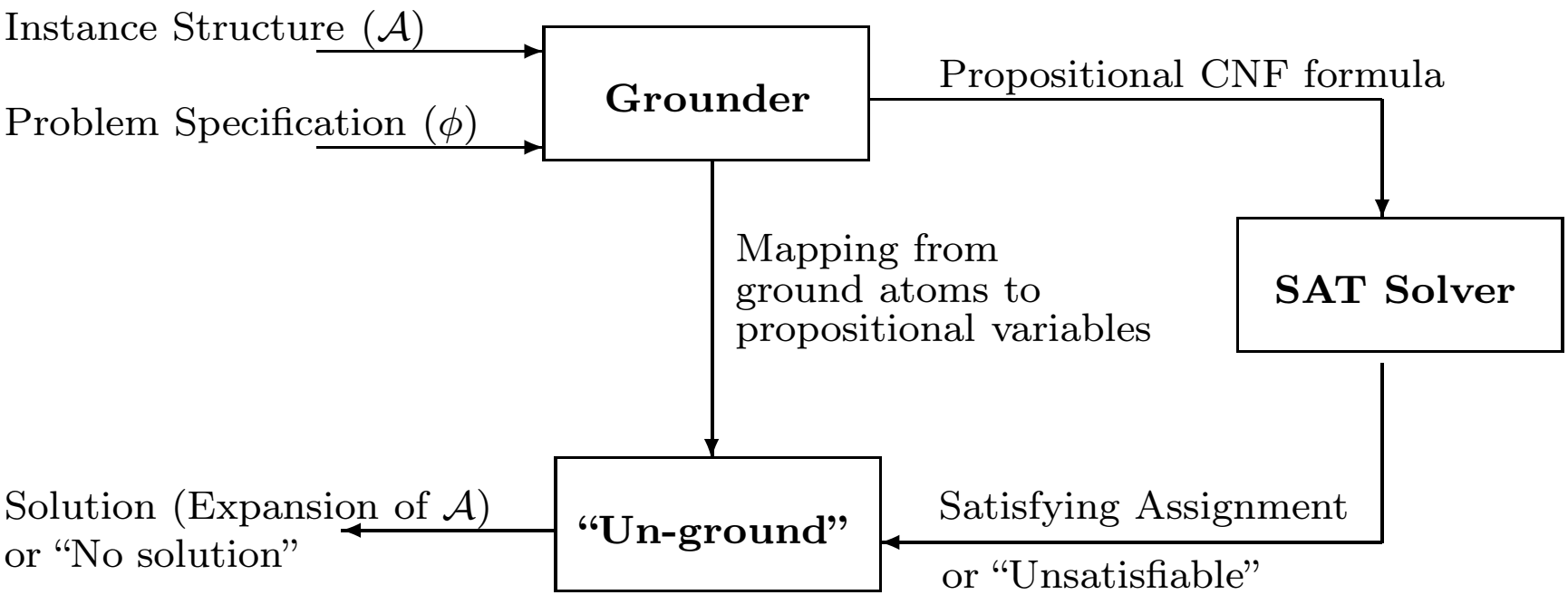
- Every component based on theory

  no implementation without formalization

- Theory and Practice proceed hand-in-hand

- Active development of theory under way,
- Prototype Solver MXG, for solving NP-search by reduction to SAT.

MXG General Solving Scheme



MXG: Modelling Language

- Classical First Order Logic, without functions

$$\textit{FO-MX} \equiv \exists \textit{SO} \Rightarrow \textit{Captures NP (Fagin'74)}$$

- Extend with:
  - Sorts : (rudimentary type system)
  - Order : All structures are ordered.
    - ★ $<, \leq, \ldots, \forall x < y$, MIN, MAX, SUCC, for each sort.
  - Inductive Definitions: FO(ID)-MX
  - Cardinality Constraints :
    - ★ e.g. $\forall u : \mathit{CARD}(1; v; \mathit{Edge}(v, u))$ = "each vertex has in-degree 1".
- Future:
  - Functions
  - Arithmetic
  - Further Aggregates
  - Type System
  - Syntactic Variants: e.g., Extended SQL

Example

MXG Specification for Graph Colouring

```
Given:
    type Vertex Colours;
    Edge( Vertex, Vertex )

Find:
    Colour( Vertex, Colours )

Satisfying :
    ∀ x y z>y : ¬ ( Colour(x,y) ∧ Colour(x,z) )
    ∀ x y : Edge(x,y) ⇒ (∀ z: ¬ (Colour(x,z) ∧ Colour(y,z)))
    ∀ x: ∃ z : Colour(x,z)
```

Outline

Grounding for MX

Grounding is an important system component, but:

- Most grounders do (nearly) naive substitution for variables.
- no theory to support serious study or development of techniques

Grounding as Generalized Query Answering

- Boolean Query: $\phi : \mathcal{A} \rightarrow \{true, false\}$
- MX Specification: $\phi : \mathcal{A} \rightarrow \{$ expansions satisfying $\phi\}$
- MX Grounding: $\phi : \mathcal{A} \rightarrow \psi$: Mod($\psi$)=$\{$ expansions satisfying $\phi\}$

- Reduced grounding for a sentence ϕ wrt structure $\mathcal{A}$ is:
  - a ground formula ψ,
  - no instance vocabulary symbols (from $\mathcal{A}$),
  - any expansion of $\mathcal{A}$ satisfies ϕ, iff it satisfies ψ.
- MXG: constructs reduced grounding using generalized relational algebra [Patterson et al 2007].
- Now: use database techniques, etc., to improve performance.

Outline

Performance Comparison: MXG *vs.* ASP Solvers

Specification for Latin Square Compl.

```
Given:    type Num;
          Preassigned(Num, Num, Num)

Find:     Cell(Num, Num, Num)

Satisfying :
        ∀ x y z : (Preassigned(x, y, z) ⊃ Cell(x, y, z))
        ∀ x y : CARD(1, z; Cell(x, y, z))
        ∀ x z : CARD(1, y; Cell(x, y, z))
        ∀ z y : CARD(1, x; Cell(x, y, z))
```

Key

MXG+MXC

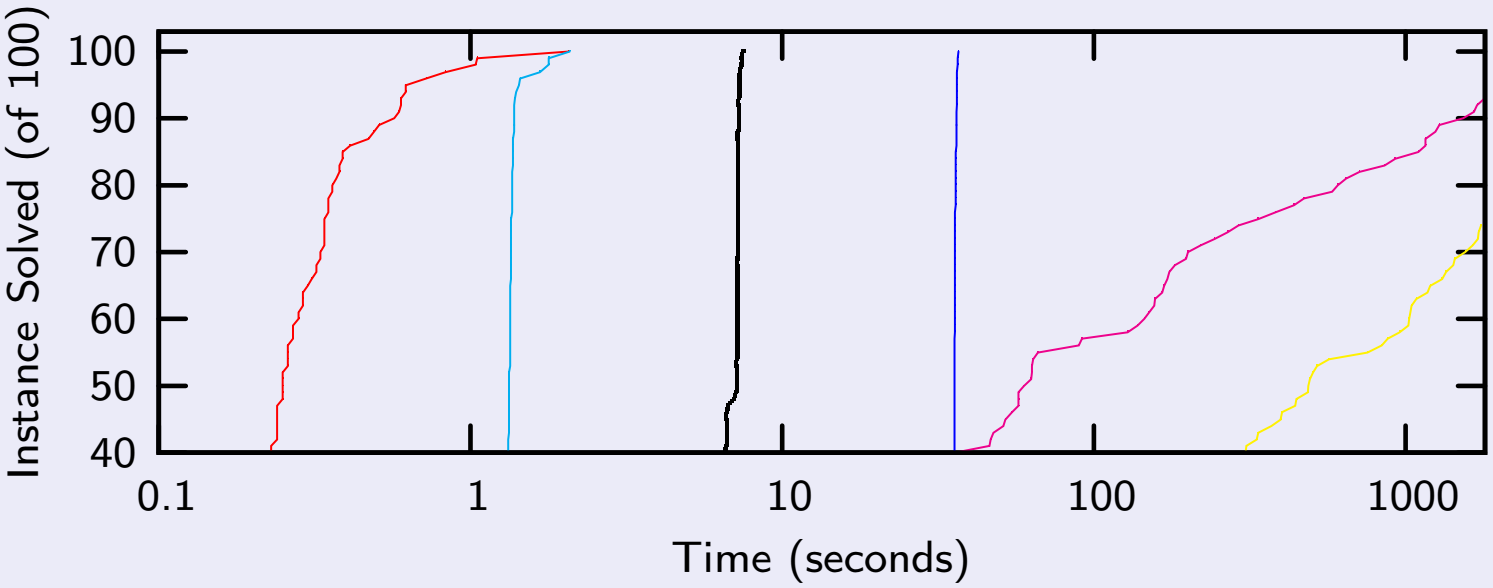
clasp

DLV

Smodels

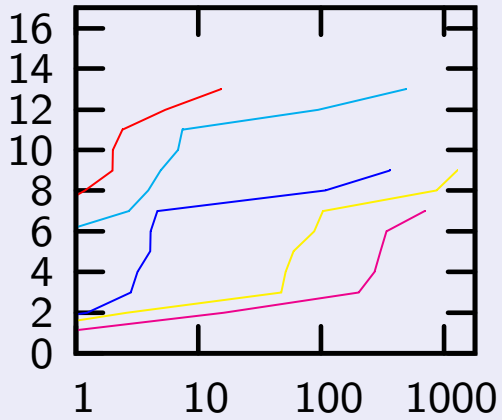
MidL

Plot

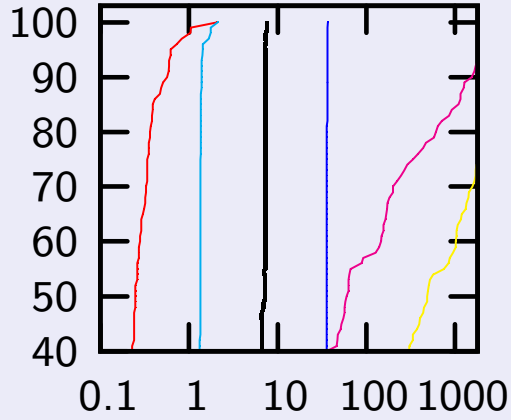


MXG Performance

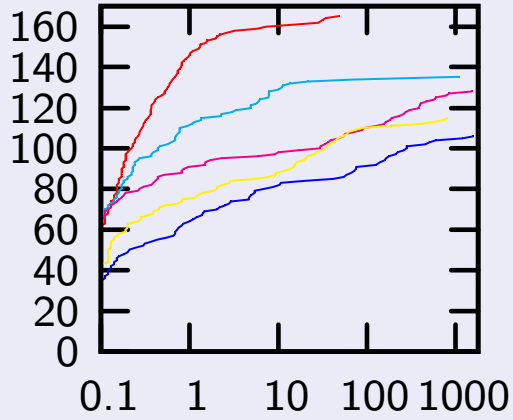
K-Coloring



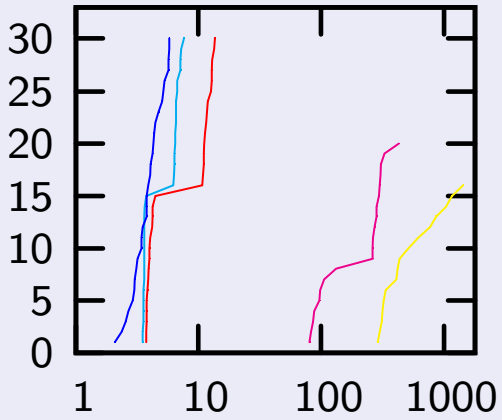
Latin Square



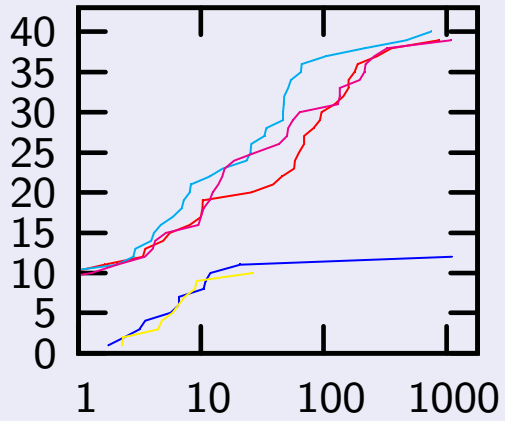
Social Golfer



Bnd Span Tree



Blocked Queens



Key

MXG+MXC

clasp

DLV

Smodels

MidL

Outline

Practical Performance Needs

Logicians: done when we have a correct axiomatization.

Customers: people want instances solved, not axiomatized.

To solve hard search problems, we must help the solver!

Axiomatization Techniques for Performance

Some basic techniques:

- Eliminating Symmetric Clauses
- Solution Symmetry Breaking
- Redundant Axioms
- Auxiliary Predicates
- Inductive Definitions as a pre-processing technique
- Bounded Quantifiers and Ordering
- Cardinality Constraints

Note: With cardinality we usually use MXC, a SAT+Cardinality solver.

Outline

Larger Example: "Phylogenetic Reconstruction"

- Given species data, construct an evolutionary tree that explains it
- Many variants – most NP-Complete
- Declarative solutions would be very useful

The Binary Camin-Sokal Problem (BCS)

INSTANCE:

- Set S of n distinct vectors from $\{0,1\}^m$;
- Natural number B.

QUESTION: Is there a directed tree $T = (V, E)$, such that:

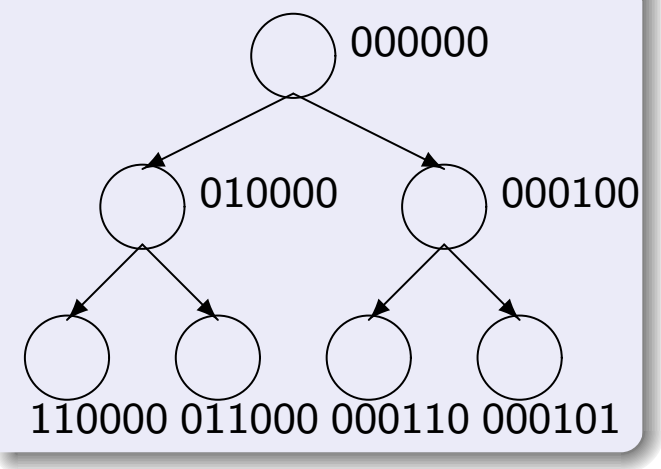
- T is rooted at 0^m;
- $S \subseteq V \subseteq \{0,1\}^m$;
- Every leaf of T is in S;
- For each directed edge $(v_1, v_2) \in E$, v_1 and v_2 differ in exactly one character, which is 0 at v_1 and 1 at v_2;
- $|V| \leq B$.

The Binary Camin-Sokal Problem (BCS)

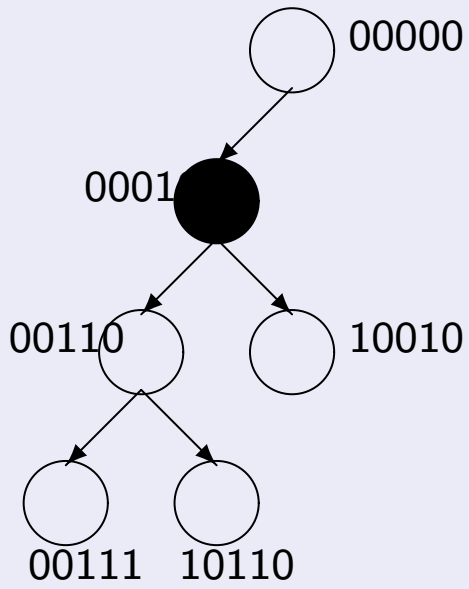
Perfect Phylogeny, each character mutates *exactly once*

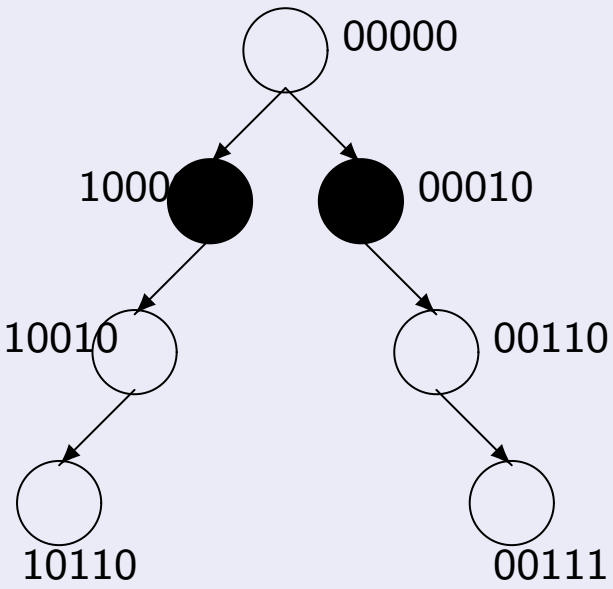
(# Nodes = # Characters + 1)

Perfect Phylogeny



Optimal and Sub-Optimal Trees





Conflict: 4 species covering $\{00, 01, 10, 11\}$ for some pair of characters

Requires one species appear in two distinct sub-trees

Basic MX Axiomatization of BCS

```
Given:
    type   Char State Vertex;
    A(Vertex, Char, State)   // Species Data
    NSpecies: Vertex
    NEdges: Vertex

Find:
    Edge(Vertex, Vertex)   // The Tree
    Vector(Vertex, Char)    // Vectors labeling vertices

Satisfying:
```

Basic MX Axiomatization of BCS

Continued

Each edge has exactly one mutation, which is $0 \longrightarrow 1$

$\forall$ u v : Edge(u,v) $\supset$ ($\exists$c : ($\neg$Vector(u,c) $\wedge$ Vector(v,c)))

$\forall$ u v : UB(1; c; (Edge(u,v) $\wedge$ $\neg$Vector(u,c) $\wedge$ Vector(v,c)))

$\forall$ u v : UB(0; c; (Edge(u,v) $\wedge$ Vector(u,c) $\wedge$ $\neg$Vector(v,c)))

Edge represents a tree rooted at $\vec{0}$

CARD(NEdges; v, u; Edge(v,u))

$\forall$ v $>$ MIN : CARD(1; u; Edge(u, v))

First *n* vertices labeled with species data

$\forall$ s $\leq$ NSpecies c : (A(s,c,MAX) $\Leftrightarrow$ Vector(s, c))

Measuring Progress

Instances:

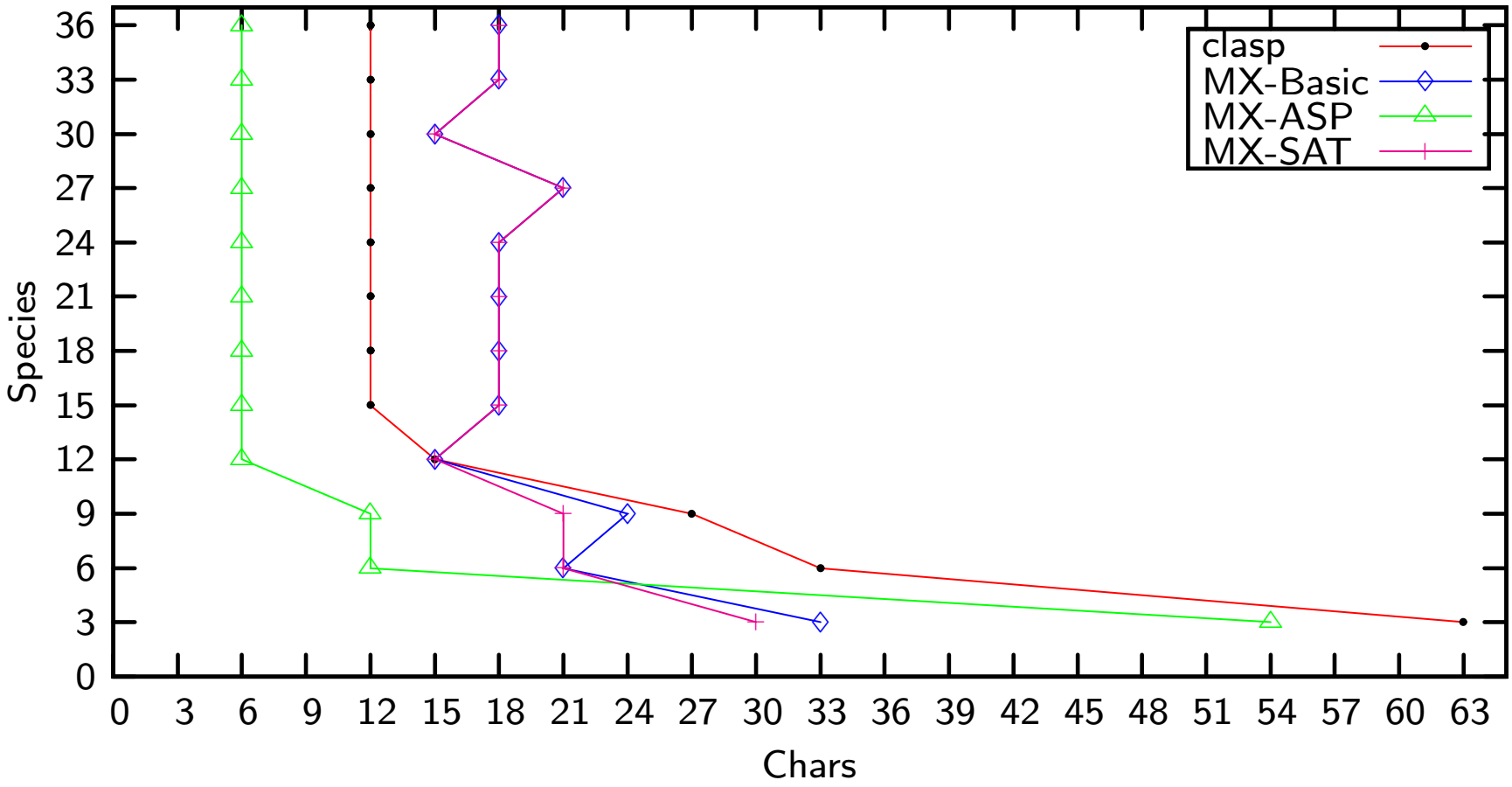
- Real Data Set: 37 Species, 65 Characters, 2 States
- Graduated instances:
  - subsets with first k species; first l characters (for k, l multiples of 3).
  - Let $\vec{0}$ be the first species; put other in reverse lexicographic order. (Species 2 has many 1's.)

Measuring Performance:

- Solve to Optimality (solve a sequence of instances)
- Plot maximum size solved to optimality in 2 hours

MXG *vs* Previous ASP Solution

Answer Set Programming: Specification with Logic Programming Rules
clasp: Very fast ASP solver



Improving Performance: Redundant Axioms (Depth)

FACT: In a BCS tree, a vector with k 1's must be at depth k
Make the property explicit with redundant axioms:

Each vertex has a unique depth

$\forall$ u : CARD(1; d; VtxDepth(u, d))

Depth difference between a child and a parent is 1

$\forall$ u v d1 d2 : ((Edge(u,v) $\wedge$ VtxDepth(u, d1) $\wedge$ VtxDepth(v,d2))
$\supset$ SUCC(d1, d2))

A species must be at it's correct depth

$\forall$ u$\leq$NSpecies d: (VtxDepth(u,d) $\Leftrightarrow$ SpcDepth(u,MAX,d))

Further Axioms (DepthPlus)

Only the root is at depth 0

$\forall$ u>MIN : $\neg$ VtxDepth(u,MIN)

"extra" vertices are ordered by depth (Symmetry Breaking Axiom)

$\forall$ u>NSpecies v>u d1 d2 : ((VtxDepth(u,d1) $\wedge$ VtxDepth(v,d2))
$\supset$ d2 $\geq$ d1)

"extra" vertices are not leaves (not quite redundant – but very helpful)

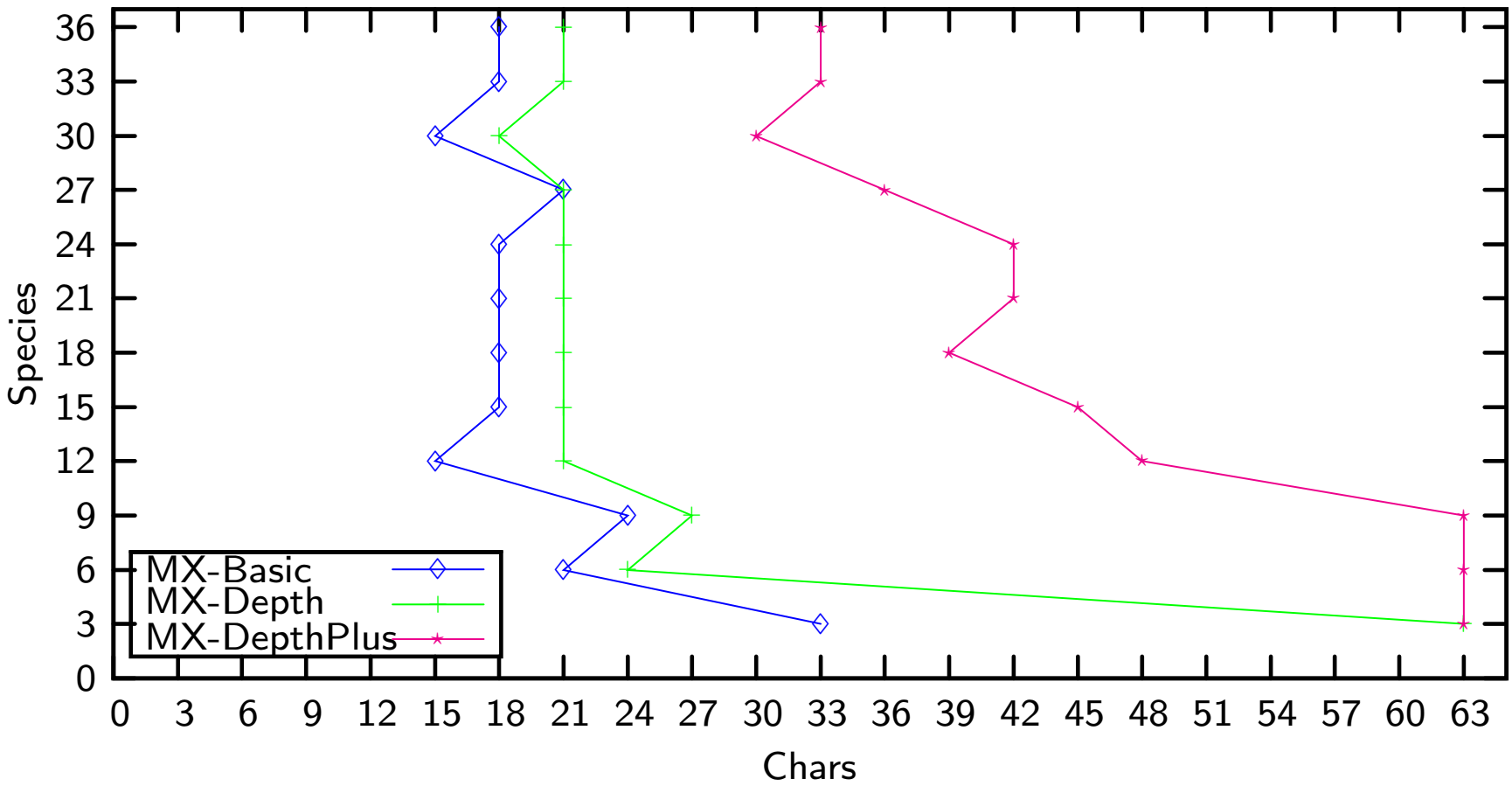
$\forall$ u>NSpecies : $\exists$ v : Edge(u,v)

Redundant Axioms - Pre-Computing Depth

MXG Computes this Definition while grounding
It determines the depth of each species vector

$$
\begin{aligned}
\{\ & \mathrm{SpcDepth(u,c,d)} \leftarrow \mathrm{c{=}MIN} \wedge \mathrm{d{=}MIN} \wedge \mathrm{s} = \mathrm{MIN} \wedge \mathrm{A(u,c,s)} \\
& \mathrm{SpcDepth(u,c,d)} \leftarrow \mathrm{c{=}MIN} \wedge \mathrm{SUCC(MIN,\ d)} \\
& \qquad\qquad \wedge\ \mathrm{s} = \mathrm{MAX} \wedge \mathrm{A(u,c,s)} \\
& \mathrm{SpcDepth(u,c,d)} \leftarrow \mathrm{SpcDepth(u,c1,\ d)} \wedge \mathrm{SUCC(c1,\ c)} \\
& \qquad\qquad \wedge\ \mathrm{A(u,c,s)} \wedge \mathrm{s{=}MIN} \\
& \mathrm{SpcDepth(u,c,d)} \leftarrow \mathrm{SpcDepth(u,c1,\ d1)} \wedge \mathrm{SUCC(c1,c)} \\
& \qquad\qquad \wedge\ \mathrm{SUCC(d1,\ d)} \wedge \mathrm{A(u,c,s)} \wedge \mathrm{s{=}MAX} \\
\}
\end{aligned}
$$

Performance of Refined Axiomatizations

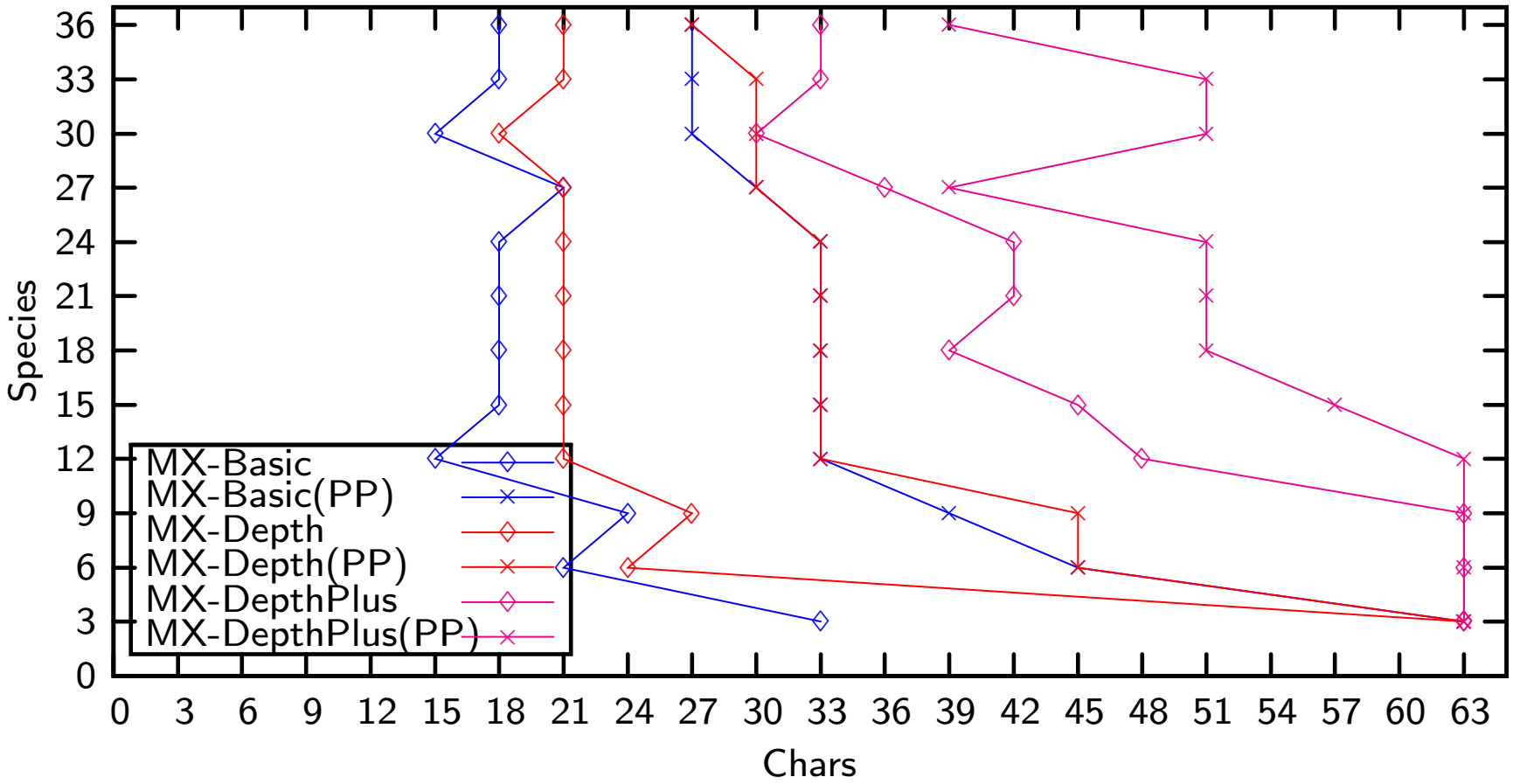


Instance Pre-Processing

We recursively applied the following rules:

- Delete any all-zero column.
- Delete any column having exactly one 1.
- Delete any column having exactly one 0.
- Delete any duplicate species.
- Delete s_2 for any pair s_1, s_2 of species such that:
  - $s_1 \subset s_2$, i.e., every character that is 1 for s_1 is also 1 for s_2;
  - $|s2 - s1| = k$, i.e., s_2 has k more 1's than s_1;
  - $\forall s_3 \notin \{s_1, s_2\}, |s_2 \backslash s_3| \geq k$.

Performance of Refined Axiomatizations



These changes reflect many orders of magnitude difference in run times.

Outline

Conclusions

Our experience to date indicates:

- FO(ID)-MX and our grounding/solving method are feasible approach to NP Search.
- MXG is competitive with high-quality ASP solvers.
- We can refine performance of MXG by:
  - usual techniques of declarative programming (at specification level),
  - specific features of MXG.
- Using a modelling language front end can be effective:
  - quickly produce reasonable solutions
  - efficiently try multiple representations
  - effectively refine to obtain good performance
  - de-bugging is more efficient than with standard methodology
- Good performance can be obtained:
  - with logic-based tools
  - without ad-hoc restrictions on language
- Poly-time instance pre-processing can be done declaratively
- We also found MXG a useful SAT solver debugging aid!

Future Work

- Language Extensions (i.e., arithmetic, strings, matrices)
- Improved Grounding (i.e., generalize query optimization)
- Specification Pre-processing (i.e., symmetry detection)
- Other Ground Solvers (i.e., SMT, PB, ILP)
- Extend to Optimization Problems
- Handle unrestricted inductive definitions
- Further Evaluation:
  - wider range of problems
  - other competing technologies (CLP, CSP, ILP, ...)

Credits

MX Project Principals:

- Eugenia Ternovska,
- David G. Mitchell

MXG Implementation and Demonstration:

- Raheleh Mohebali,
- Faraz Hach

MXC SAT Solver:

- David Bregman

Others:

- Yongmei Liu, Arvind Gupta, Murray Patterson, Jonathan Kavanagh, Jano Manuch, Sharon Xiao, Nhan Nguyen, Brendan Guild, Nikolay Pelov, ...

Links and References

- MX Project Web Site: www.cs.sfu.ca/research/groups/mxp/
  (painfully out of date)
- Abstraction and Expressiveness in Essence

  M., Ternovska — Constraints (to appear)
- Declarative Programming for Search Problems with Built-in Arithmetic

  Ternovska, M.
- Faster Phylogenetic Inference with MXG

  M., Hach, Mohebali — LPAR'07
- Constructing Camin-Sokal Phylogenies via Answer Set Programming

  Kavanagh, M., Ternovska, Manuch, Zhao, Gupta — LPAR'06
- A Framework for Modelling and Solving Search Problems

  M., Ternovska, Hach, Mohebali — SFU CS TR 2006-24
- A Framework for Representing Search Problems

  M., Ternovska — AAAI'05