Constraint Programming for Secondary Structure Prediction

Johannes Waldmann [1]

October 5, 2012

[1] Fakultät IMN, HTWK Leipzig, Germany

Secondary Structure Prediction

input: primary structure (RNA sequence)

```
GGGAAAUGGACUGAGCGGCGCCGACCGCCAAACAACCGGCA
```

output: encoding of secondary structure (base pairs)

```
: [ [ : : : ( ( ( : ] ] : : : ( ( ( : [ [ [ [ : ) ) ) ) ) ) : : : : : : ] ] ] ] :
```

value: sum of stack lengths

```
        1 +  2    +    2 +  3           = 8
```

This is a constraint satisfaction problem (if lower value bound is given),
a constrained optimization problem (if value is to be maximized).

Approaches for solving

- complete enumeration (hopeless)
- restrict to underlying models with efficient algorithms,
  e.g., (multiple) context-free grammar and CYK (tabled) parsing
- (this talk): handle the constraint satisfaction problem as-is
- slogan: don't fear NP-completeness, hail Minisat ($=$ efficient solver for Boolean satisfiability problems)

Constraint Program (Example)

$P, Q, R, S \in \mathbb{Z}$,

$0 < P \wedge 0 \leq Q \wedge 0 < R \wedge 0 \leq S \wedge PS + Q > RQ + S$

Textual representation (SMT2 standard)

```
(set-logic QF_NIA)
(set-option :produce-models true)
(declare-fun P () Int) (declare-fun Q () Int)
(declare-fun R () Int) (declare-fun S () Int)
(assert (and (< 0 P) (<= 0 Q) (< 0 R) (<= 0 S)))
(assert (> (+ (* P S) Q) (+ (* R Q) S)))
(check-sat)(get-value (P Q R S))
```

Solver (`research.microsoft.com/projects/z3/`)

```
$ z3 con-exp.smt2
sat  ((P 14) (Q 9) (R 11) (S 7))
```

Constraint Programming

- constraint program: a formula P in predicate calculus, containing
  - predefined functions and relations from some domain (e.g., linear or polynomial equalities or inequalities)
  - free variables (unknowns) $v_1, \dots$
- solution: an assignment σ (mapping from variables to values) such that $P\sigma$ is true
- constraint solver: computes σ from P
- the *application* programmer benefits from the highly sophisticated *domain*-specific search algorithms in the solvers (e.g., Gauss, Simplex, Qepcad, Nelson-Oppen, DPLL(T))

Boolean Constraints (Example)

$x_1, x_2, x_3, x_4 \in \mathbb{B}$

$x_3 \leftrightarrow (x_1 \oplus x_2) \;\wedge\; x_4 \leftrightarrow (x_1 \wedge x_2) \;\wedge\; x_4$

equivalent conjunctive normal form:

$(x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$
$\wedge (\overline{x}_4 \vee x_1) \wedge (\overline{x}_4 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_4) \wedge x_4$

textual representation (DIMACS file format)

```
p cnf 4 8
1 -2 3 0  -1 2 3 0  1 2 -3 0  -1 -2 -3 0
-4 1 0  -4 2 0  -1 -2 4 0  4 0
```

Solver (`http://minisat.se/`)

```
$ minisat sat-exp.dimacs /dev/stdout
SAT  1 2 -3 4 0
```

Boolean Constraints (SAT)

- domain (for values and variables): $\mathbb{B} = \{0, 1\}$
- deciding satisfiability of Boolean formulas is NP-complete
  unless $P = NP$, there is no algorithm that is efficient in *all* cases
- DPLL (Davis-Putnam-Logemann-Loveland) with CDCL (conflict driven clause learning) is surprisingly efficient in *a lot* of cases.
- industrial-strength solvers (used in verification of hardware and software), SAT competitions, ...
- finite domain constraint problems can be solved by transformation to SAT.

Finite Domain (FD) Constraints

- SAT: unknowns are Booleans $\mathbb{B} = \{0, 1\}$
- FD: unknowns from some finite set, e.g., $\text{Colour} = \{\text{empty}, \text{black}, \text{white}\}$
- unary encoding: Colour $\hookrightarrow \mathbb{B}^3$
  $\text{empty} = (1, 0, 0), \text{black} = (0, 1, 0), \text{wh.} = (0, 0, 1)$
- binary encoding: Colour $\hookrightarrow \mathbb{B}^3$
  $\text{empty} = (0, 0), \text{black} = (0, 1), \text{white} = (1, 0)$
- can be used directly for graph (colouring) problems, parsing problems, etc.
- for problems with infinite (or large) domain, try to find some FD approximation
  (represent numbers in some fixed bit width)

SAT Coding Expl.: State Transitions

```
[0,0,2,2,2,2,1,1,1,1]
[1,1,2,2,2,2,1,0,0,1]
[1,1,2,2,0,0,1,2,2,1]
[1,0,0,2,1,2,1,2,2,1]
[1,2,1,2,1,2,1,2,0,0]
```

- unknowns: $x_{t,p}$ where $t =$ time, $p =$position
- obvious initial/final condition, transitions:

$$\bigwedge_t \bigvee_{a,b} \bigwedge_p \left(\begin{array}{l} p \notin \{a, a+1, b, b+1\} \Rightarrow x_{t,p} = x_{t+1,p} \\ \wedge\, x_{t,a} = x_{t+1,b} \wedge x_{t,a+1} = x_{t+1,b+1} \\ \wedge\, x_{t+1,a} = 0 \wedge x_{t+1,a+1} = 0 \\ \wedge\, x_{t,b} = 0 \wedge x_{t,b+1} = 0 \end{array} \right)$$

improve to $\bigwedge_t \exists r, s \in \{1, 2\} \, (\bigvee_a \bigwedge_p \dots) \wedge (\bigvee_b \bigwedge_p \dots)$

complete source code (100 lines) `http://dfa.imn.htwk-leipzig.de/cgi-bin/gitweb.cgi?p=biosat.git;a=blob;f=rewriting/C.hs;hb=HEAD`

SAT encoding for Sec. Struc. Pred.

model: disjoint circular matchings in graphs
input: $G = (V, E)$ where $V =$ positions in RNA string, $E =$ set of all possible base pairs; number $k \in \mathbb{N}$
output: sequence $M_1, \ldots, M_k$ with $M_i \subseteq E$
such that

- $M := \bigcup_i M_i$ is a matching (each $v \in V$ is incident to at most one edge in M)
- each M_i is circular (no crossing edges w.r.t. the ordering on V)

each M_i is an edge set, thus a relation, thus a boolean matrix $M_i : V \times V \to \mathbb{B}$
the unknowns of the constraint system are the entries of these matrices.

Related Work

- Unyanee Poolsap, Yuki Kato, and Tatsuya Akutsu: *Prediction of RNA secondary structure with pseudoknots using integer programming*, BMC Bioinformatics. 2009; 10(Suppl 1): S38.
- Ganesh et al.: *Lynx: A Programmatic SAT Solver for the RNA-Folding Problem*, SAT'12 using the direct encoding (l^4 clauses) for the non-crossing condition

Encoding details

- union: $M = \bigcup_i M_i$
  $M(p,q) := \bigvee_i M_i(p,q)$
- possible base pairs $M \subseteq E$:
  $\bigwedge\{\neg M(p,q) \mid (w[p], w[q]) \notin \{AU, UA, CG, GC, GU, UG\}\}$
- M is matching:
  $\bigwedge\{\neg(M(p,q) \wedge M(q,r)) \mid p \neq r\}$
- M_i is circular (non-crossing):
  $\bigwedge\{\neg(M_i(p,q) \wedge M_i(r,s)) \mid p < r < q < s\}$
- number of variables: $l^2 k$, formula size: $\Theta(l^4 k)$.

Encoding for CYK parsing

…to reduce the l^4 formula size
use table (relation) T with specification
$T(p,q) \iff w[p..q]$ is correctly parenthesized:

- $T(p,p) \iff p \notin \text{domain}\, M \cup \text{range}\, M$
- $T(p,q) \iff$
  $(M(p,q) \wedge T(p+1, q-1)) \vee \bigvee_h T(p,h) \wedge T(h+1, q)$
- $M(p,q) \Rightarrow T(p,q)$, $M(1,l)$

l^2 variables, l^3 formula size

source code: `http://dfa.imn.htwk-leipzig.de/cgi-bin/gitweb.cgi?p=biosat.git;a=blob;f=ssp/code/SSP/Graph/Encode.hs;hb=HEAD` Note: cannot apply CYK to the original problem, since we need to guess the type of parentheses. (this parsing problem is NP-hard)

Encoding Numeric Valuation

For valuation of $(M_1, \dots, M_k)$, consider *stacks* (groups of parallel edges in $M = \bigcup_i M_i$)

- Define $S : V \to \mathbb{B}$ by
  $S(p) := \bigvee_q M(p, q) \wedge M(p + 1, q - 1)$,
- count number of 1 in $(S(1), \dots, S(l))$
  by repeated binary addition
  (using half adder/full adder circuits represented as constraint systems)
- compare with a given bound
  $v \geq B \iff \exists d : v = B + d$

Solving the Optimization Problem

- write the constraint system $C(P, S, V) =$ "S is an admissible solution for problem P with value $\geq V$"
- to find $\max\{V \mid \exists S : C(P, S, V)\}$, determine a finite feasible range for V (e.g., $0 \ldots$ length of input)
- use iteration $V = 0, 1, 2, \ldots$ or bisection $V = l/2, 3l/4, \ldots$

Prototype Implementation

of Secondary Structure Prediction with fixed number of parenthesis types
is proof-of-concept, as a basis for experimentation:

- source: `git://dfa.imn.htwk-leipzig.de/srv/git/biosat`
- using Haskell library Satchmo `https://github.com/jwaldmann/satchmo` to generate SAT constraint system and decode result
- solver: `https://github.com/niklasso/minisat`

Program Inversion

Constraint system $C(P, S, V) =$

"S is an admissible solution for problem P with value $\geq V$"

can be used for:

- given P, V, determine S
  e.g., RNA parsing (sec. struct. pred.)
- given S, V, determine P
  e.g., RNA design

Conclusion/Claims

- constraint programming is *easy*: especially for non-programmers, since it is *declarative*
- constraint programming is *powerful*:
  use generic *domain-specific* solver for
  application-specific program/problem
- constraint programming is *flexible*:
  easily add/remove/change/invert constraints
  (much easier than change an
  application-specific algorithm)
- write the constraint program in an EDSL
  (*embedded domain specific language*)
  that takes care of encoding and decoding