Encoding Hard String Problems with Answer Set Programming

Dominik Köppl

Uni Muenster, Germany

warning:
Although I mostly work on algorithms and data structures, we here use artificial intelligence tools for problem solving

problem setting

- only a tiny fraction of problems are efficiently solvable
- infinitely many problems are NP-hard (NP-hard is closed under union/intersection/concatenation)
- but sometimes we need really to solve a problem, for which no efficient solution exists

What can we do?

- use heuristics: approximation algorithms, probabilistic tree search, evolutionary algorithm, etc.
- but may not work if we want the exact solution!

On what problems we want to look at?

exemplary: Closest String

Problem Closest String

Input

- set of m strings $\mathcal{S} = \{S_1, \ldots, S_m\}$ on an alphabet Σ of size σ
- $|S_j| = n \quad \forall j \in [1..m]$

Task: find string T with

- $|T| = n$
- $\max_{x \in [1..m]} \mathrm{dist}_{\mathrm{ham}}(S_x, T)$ is minimal

where $\mathrm{dist}_{\mathrm{ham}}(S_x, T) := |\{i \in [1..n] : S_x[i] \neq T[i]\}|$ is Hamming distance between T and S_x.

- problem is NP-hard for $\sigma \geq 2$ in n and m! Frances,Litman'97
- fortunately: already exist efficient solutions for this problem (ILP solver, etc.)

example

	1	2	3	4	5	6	7	8	9	10	11	12	13
$S_1 =$	l	n	e	e	p	l	e	s	s	n	e	l	s
$S_2 =$	s	l	e	e	p	s	l	s	s	n	e	s	n
$S_3 =$	n	l	e	l	p	l	e	s	s	n	s	s	s
$S_4 =$	s	n	e	e	p	l	e	l	s	n	s	s	s
$S_5 =$	s	l	l	e	e	l	e	s	s	n	s	s	s

example

	1	2	3	4	5	6	7	8	9	10	11	12	13
$S_1 =$	l	n	e	e	p	l	e	s	s	n	e	l	s
$S_2 =$	s	l	e	e	p	s	l	s	s	n	e	s	n
$S_3 =$	n	l	e	l	p	l	e	s	s	n	s	s	s
$S_4 =$	s	n	e	e	p	l	e	l	s	n	s	s	s
$S_5 =$	s	l	l	e	e	l	e	s	s	n	s	s	s
$T =$	s	l	e	e	p	l	e	s	s	n	e	s	s

why this problem?

- well-studied:
  - 31 conference papers
  - 22 journal papers
- it is a string problem, and we love strings!

yet. . .

do we have any implementation of a solution available so far?

> *"We do not compare with the algorithm in [6], because its code is not available."*

Shota Yuasa, Zhi-Zhong Chen, Bin Ma, Lusheng Wang:
Designing and Implementing Algorithms for the Closest String Problem.
Proc. FAW 2017, LNCS 10336, pages 79-90

Of course, the authors also did not publish their code. . .

So is there any implementation available at all?

```
The algorithm is explained in detail in the
following article:


https://example.com
```

https://github.com/kirilenkobm/BDCSP (accessed: 30th of April 2023)

Other Half-Baken Code Repositories

- "A challenge to make this basic closest-strings program more efficient. "
  last update: 3 years ago (2020)

  `https://github.com/robertvunabandi/closest-strings-challenge`
- "Swarm Intelligence project: Closest string problem"
  last update: 6 years ago (2017)

  `https://github.com/arnomoonens/closest-string-problem`
- ⋮

Looks like some unfinished student projects. So:

- will the code run? maybe
- will it produce correct results? unknown: there are (mostly) no tests

our aim

exact search:

- brute-force, exhaustive search : easy to program, but combinatorial explosion prevents from working even on small input sizes
- Integer linear programming (ILP) or MAX-SAT formulation: burden on the implementation!

want to have: tool for fast prototyping

- easy implementation
- speed should be reasonable
- goals:
  - fast problem solving
  - usable for testing coding-intensive implementations at an early stage

introduction to answer set programming (ASP)

- Prolog-like declarative language
- most classic problems like traveling salesman problem can be expressed in a few lines of code, but still performant on small instance sizes
- current standard: ASP-Core-2 Calimeri+'19
- standard reference implementation: `clingo`
    - in active development at `https://potassco.org/clingo/` (University of Potsdam) by Torsten Schaub
    - shipped with common Linux distributions such as Ubuntu/Debian: `adb install gringo`

how to solve Closest String with ASP?

with seven lines of code:

```
mat(X,I) :- s(X,I,_).
1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
c(X,I) :- t(I,C), s(X,I,A), C != A.
cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,_).
mcost(M) :- M = #max {C : cost(_,C)}.
#minimize {M : mcost(M)}.
#show t/2. #show mcost/1. #show cost/2.
```

how does the input look like?

transform texts

- $S_1 =$ `lneeplessnels`
- $\vdots$
- $S_5 =$ `slleelessnsss`

write $S_j[i]$ as $s(j, i, rank(S_j[i]))$, where *rank* is the ASCII rank of the symbol

- `l` $\mapsto 108$
- `n` $\mapsto 110$
- `e` $\mapsto 101$
- `s` $\mapsto 115$

ASP input

```
s(0, 0, 108).
s(0, 1, 110).
s(0, 2, 101).
...
s(4, 10, 115).
s(4, 11, 115).
s(4, 12, 115).
```

modelling the input

- so we have at startup tuples $s(i, j, S_i[j])$
- next we create a boolean matrix `mat` that specifies whether $S_i[j]$ exists

```
mat(X,I) :- s(X,I,_).
1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
c(X,I) :- t(I,C), s(X,I,A), C != A.
cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,
      _).
mcost(M) :- M = #max {C : cost(_,C)}.
#minimize {M : mcost(M)}.
#show t/2. #show mcost/1. #show cost/2.
```

but how do we get to the closest substring of that?

Restriction of Optimal Solution

Lemma (Kelsey, Kotthoff'11)

There exists an optimal solution T with $T[i] \in \{S_1[i], \ldots, S_m[i]\}$.

Proof.

- if $T[i] \notin \{S_1[i], \ldots, S_m[i]\}$, then T mismatches with all input strings at position i
- if $T[i] = S_j[i]$, then the distance to at least S_j is better, so it does not worsen the distance

□

Definition

define $\Sigma_i := \{S_1[i], \ldots, S_m[i]\}$ effective alphabet for position $i \in [1..n]$

modelling T

- model $T[i]$ as a boolean matrix $T_{i,c} = 1 \Leftrightarrow T[i] = c$
- state that $T[i] = S_x[i]$, i.e., only one $T_{i,c}$ is set:

$$\forall i \in [1..n] : \sum_{c \in \Sigma_i} T_{i,c} = 1$$

$$[\mathcal{O}(n), \mathcal{O}(\min(m, \sigma))]$$

```
mat(X,I) :- s(X,I,_).
1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
c(X,I) :- t(I,C), s(X,I,A), C != A.
cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,
    _).
mcost(M) :- M = #max {C : cost(_,C)}.
#minimize {M : mcost(M)}.
#show t/2. #show mcost/1. #show cost/2.
```

complexity (x,y):

- x : # clauses
- y : # variables per clause

modelling costs

- define $C_{i,x} \in \{0,1\}$: $\forall i \in [1..n], x \in [1..m]$ with $C_{i,x} = 1$ if $T[i] \neq S_x[i]$.
- then $\mathrm{dist}_{\mathrm{ham}}(T, S_x) = \sum_{i\in[1..n]} C_{i,x}$ is Hamming distance between T and S_x

$$\forall i \in [1..n], c \in \Sigma_i, x \in [1..m] : \quad T_{i,c} \wedge S_x[i] \neq c \implies C_{i,x}$$

$[\mathcal{O}(nm\sigma), \mathcal{O}(1)]$

```
mat(X,I) :- s(X,I,_).
1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
c(X,I) :- t(I,C), s(X,I,A), C != A.
cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,
    _).
mcost(M) :- M = #max {C : cost(_,C)}.
#minimize {M : mcost(M)}.
#show t/2. #show mcost/1. #show cost/2.
```

maximum of summed costs

- add helper variables $\text{cost}_x := \sum_{i \in [1..n]} C_{i,x} = \mathrm{dist}_{\mathrm{ham}}(T, S_x)$
- and compute the maximum value $\text{mcost} := \max\{\text{cost}_1, \ldots, \text{cost}_m\}$

```
mat(X,I) :- s(X,I,_).
1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
c(X,I) :- t(I,C), s(X,I,A), C != A.
cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,_).
mcost(M) :- M = #max {C : cost(_,C)}.
#minimize {M : mcost(M)}.
#show t/2. #show mcost/1. #show cost/2.
```

setting the objective

- statement for setting $C_{i,x}$ to false is not needed: optimizer will do so if it does not violate Line 3
- for that, our objective is:

$$\text{minimize} \quad \max_{x \in [1..m]} \sum_{i \in [1..n]} C_{i,x}$$

$[\mathcal{O}(1), \mathcal{O}(mn)]$

```
1 mat(X,I) :- s(X,I,_).
2 1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
3 c(X,I) :- t(I,C), s(X,I,A), C != A.
4 cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,
      _).
5 mcost(M) :- M = #max {C : cost(_,C)}.
6 #minimize {M : mcost(M)}.
7 #show t/2. #show mcost/1. #show cost/2.
```

specifying the output

output T, mcost, and cost

```
mat(X,I) :- s(X,I,_).
1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
c(X,I) :- t(I,C), s(X,I,A), C != A.
cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,
    _).
mcost(M) :- M = #max {C : cost(_,C)}.
#minimize {M : mcost(M)}.
#show t/2. #show mcost/1. #show cost/2.
```

complexities

- $\mathcal{O}(n\sigma)$ selectable variables ($T_{i,c}$)
- $\mathcal{O}(nm)$ helper variables ($C_{i,x}$),
- $\mathcal{O}(nm\sigma)$ clauses (Line 3).

```
1 mat(X,I) :- s(X,I,_).
2 1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).
3 c(X,I) :- t(I,C), s(X,I,A), C != A.
4 cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,_).
5 mcost(M) :- M = #max {C : cost(_,C)}.
6 #minimize {M : mcost(M)}.
7 #show t/2. #show mcost/1. #show cost/2.
```

interpreting output

- since mcost $= 3$, we have at most three errors at each text position
- (actually we have exactly three errors at all positions when looking at cost for this solution)
- by remapping ASCII ranks to characters from $\texttt{t}(i, \mathit{rank}(T[i]))$, we obtain $T = \texttt{sleeplessness}$

```
mcost(3)
cost(0,3) cost(1,3) cost(2,3)
cost(3,3) cost(4,3)
t(0,115) t(1,108) t(2,101) t(3,101)
t(4,112) t(5,108) t(6,101) t(7,115)
t(8,115) t(9,110) t(10,101)
t(11,115) t(12,115)
```

works in practice

freely available at `https://github.com/koeppl/aspstring`

- python wrapper around ASP/clingo calls
- input and output: plain string(s)
- framework for working with strings: easy to write code for other string-related problems

evaluation with brute-force approach (test every possible value for $T[1..n]$)

evaluation on random datasets

		ASP				brute-force	
file	x	rules	vars	choices	[s]	choices	[s]
s05m07n009i0	6	1025	264	673	0.01	327 680	2.19
s05m07n009i1	6	1002	262	608	0.01	172 800	1.15
s05m07n009i2	6	977	253	589	0.01	98 304	0.66
s05m08n009i0	6	1122	290	605	0.01	230 400	1.74
s05m08n009i1	6	1123	290	975	0.01	216 000	1.64
s05m08n009i2	6	1136	291	716	0.01	288 000	2.17
s05m09n009i0	6	1288	321	725	0.01	640 000	5.47
s05m09n009i1	7	1258	319	1723	0.02	409 600	3.48
s05m09n009i2	7	1273	320	1828	0.02	512 000	4.33
s06m07n009i0	6	1039	265	974	0.01	384 000	2.57
s06m07n009i1	7	1078	268	1767	0.02	768 000	5.12
s06m07n009i2	6	1002	262	569	0.01	172 800	1.15
s06m08n009i0	6	1191	295	1074	0.01	750 000	5.67
s06m08n009i1	7	1248	299	2378	0.02	1 800 000	13.63
s06m08n009i2	7	1248	299	2128	0.02	1 800 000	13.61
s06m09n009i0	7	1303	322	1837	0.02	800 000	6.81
s06m09n009i1	7	1396	328	1849	0.02	2 700 000	22.97
s06m09n009i2	6	1336	324	1874	0.02	1 080 000	9.07

`s05m07n009i0` denotes

- $\sigma = 5$
- $m = 7$
- $n = 9$
- $i =$ 0-th sample (iteration)

columns:

- $x =$ mcost
- $[s]$: time in seconds

observation:

- $\#$ choices correlates with time
- ASP has much fewer to check

but wait. . .

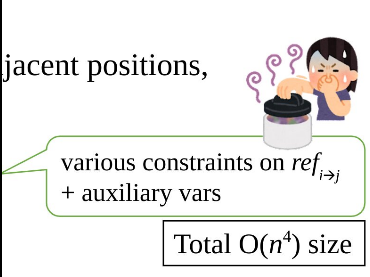
. . . if there are good solutions like with ILP for CLOSEST STRING, why bother?

maybe you work on a variation: CLOSEST STRING ⇒ CLOSEST SUBSTRING

- fewer references, much fewer implementations
- hard to adapt ILP/MAX-SAT implementations to this variation
- but easy with ASP! (see code repository)

weakness

- ASP is slower than good MAX-SAT implementation, e.g.: string attractor
- Bannai+'22: MAX-SAT for string attractor in pysat, 566 line of code
- but ASP for string attractor in 5 line of code:



Bannai+'22

```
{ in(1..n) }.
sub_str(S,E) :- cover(S,E,_).
:- not 1 { in(P) : cover(S,E,P) }, sub_str(S,E).
#minimize { 1,P : in(P) }.
#show in/1.
```

conclusion

introduction of ASP to hard string problems

- fast prototyping
- actual code available for comparison, benchmarks, etc.
- framework to implement new code easily
- usually faster than naive implementations but slower than sophisticated ones

https://github.com/koeppl/aspstring
happy coding ♪

P.S.: Initial code has already been improved by an anonymous CPM reviewer. Thanks for that!

does it work with ChatGPT? (asked by Pierre Peterlongo)

seems (un)fortunately hard:

Can you encode the closest string problem in the answer set programming (ASP) language such that I can run the code with the program clingo? The code should accept a list of input strings S1,...,S5. All strings have exactly the same length. The solution is a string that has also the same length. Your encoding must output the solution string. As an example, can you encode the input strings as S1 = ”lneeplessnels”, S2 = ”sleepslssnesn”, S3 = ”nlelplessnsss”, S4 = ”sneeplelsnsss”, S5 = ”slleelessnsss”? The code must not include the solution, and should work without a parameter specifying the maximal allowed distance. Note that neither the syntax ”#int” nor ”#len” is valid in ASP.

output of ChatGPT:

```
% Define the input strings
input_string("lneeplessnels").
input_string("sleepslssnesn").
input_string("nlelplessnsss").
input_string("sneeplelsnsss").
input_string("slleelessnsss").

% Define the length of the input strings
len(N) :- input_string(S), len(S, N).
...
```

Not even the first line, the definition of `len(S,N)` works. . .