warning:
Although I mostly work on algorithms and data structures, we here use artificial intelligence tools for problem solving
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 \( S = \{S_1, \ldots, S_m\} \) on an alphabet \( \Sigma \) of size \( \sigma \)
- \( |S_j| = n \quad \forall j \in [1..m] \)

**Task:** find string \( T \) with

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

where \( \text{dist}_{\text{ham}}(S_x, T) := |\{ i \in [1..n] : S_x[i] \neq T[i] \}| \) 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

\[
\begin{align*}
S_1 &= \text{lineepleessnells} \\
S_2 &= \text{sleeepsllssnnessn} \\
S_3 &= \text{nnleelpplessnssss} \\
S_4 &= \text{sneeepleelssnssss} \\
S_5 &= \text{ssllleelssnssss}
\end{align*}
\]
example

$S_1 = l n e e p l e s s n e l s$
$S_2 = s l e e p s l l s s n e s n$
$S_3 = n l e l p l e s s n s s s s$
$S_4 = s n e e p l e l s n s s s s$
$S_5 = s l l e e l e s s n s s s s$
$T = s l e e p l e s s n e s s$

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.”


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
- 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:

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.
how does the input look like?

transform texts

- $S_1 = \text{lneepllessnels}$
- ...
- $S_5 = \text{slleelessnssss}$

write $S_j[i]$ as $s(j, i, \text{rank}(S_j[i]))$, where $\text{rank}$ is the ASCII rank of the symbol

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

ASP input

1 $s(0, 0, 108)$. 
2 $s(0, 1, 110)$. 
3 $s(0, 2, 101)$. 
4 ...
5 $s(4, 10, 115)$. 
6 $s(4, 11, 115)$. 
7 $s(4, 12, 115)$. 

11 / 20
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

```
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.
```

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 $i \in [1..n]$
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:

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

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

complexity $(x,y)$:

$x$ : # clauses

$y$ : # variables per clause
modelling costs

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

\[
\forall i \in [1..n], c \in \Sigma, x \in [1..m] : \quad T_{i,c} \land S_x[i] \neq c \implies C_{i,x} \quad [\mathcal{O}(nm\sigma), \mathcal{O}(1)]
\]

1. \( \text{mat}(X,I) :- \text{s}(X,I,\_). \)
2. \( 1 \{ \text{t}(I,C) : \text{s}(\_,I,C) \} 1 :- \text{mat}(\_,I). \)
3. \( \text{c}(X,I) :- \text{t}(I,C), \text{s}(X,I,A), C \neq A. \)
4. \( \text{cost}(X,C) :- C = \#\text{sum} \{1,I : \text{c}(X,I)\}, \text{mat}(X,\_). \)
5. \( \text{mcost}(M) :- M = \#\text{max} \{C : \text{cost}(\_,C)\}. \)
6. \#\text{minimize} \{M : \text{mcost}(M)\}.
7. \#\text{show} t/2. \#\text{show} mcost/1. \#\text{show} cost/2.
maximum of summed costs

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

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

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

\[ [O(1), O(mn)] \]
specifying the output

output \( T \), \( mcost \), and \( cost \)

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

- $O(n\sigma)$ selectable variables ($T_{i,c}$)
- $O(nm)$ helper variables ($C_{i,x}$),
- $O(nm\sigma)$ 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 \( t(i, \text{rank}(T[i])) \), we obtain \( T = \text{sleeplessness} \)

\[
\begin{array}{cccc}
\text{mcost}(3) & \text{cost}(0,3) & \text{cost}(1,3) & \text{cost}(2,3) \\
\text{cost}(3,3) & \text{cost}(4,3) \\
\end{array}
\]

\[
\begin{array}{cccccccc}
t(0,115) & t(1,108) & t(2,101) & t(3,101) \\
t(4,112) & t(5,108) & t(6,101) & t(7,115) \\
t(8,115) & t(9,110) & t(10,101) \\
t(11,115) & t(12,115) \\
\end{array}
\]
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 $T[1..n]$)
### Table: Performance Comparison for Random Datasets

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

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

**Columns:**
- $x = m$-cost
- [s]: time in seconds

**Additional Notes:**
- The number of 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 $\Rightarrow$ 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:

```prolog
1 { in(1..n) }.
2 sub_str(S,E) :- cover(S,E,\_).
3 :- not 1 { in(P) : cover(S,E,P) }, sub_str(S,E).
4 #minimize { 1,P : in(P) }.
5 #show in/1.
```

© by Mutsunori Banbara’22
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!
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 out-
put 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 maxi-
mal allowed distance. Note that neither the syn-
tax "#int" nor "#len" is valid in ASP.

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