warning:
Although I mostly work on theoretical stuff, we here get actual code to run!
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?
best matching score

What is the music score that has the fewest maximal mismatches with each of the given scores?
Solution has exactly three errors with each input score!

There is no solution with fewer errors.
reduction to \textbf{Closest String}

Problem \textbf{Closest String}

Input
- set of \(m\) strings \(S = \{S_1, \ldots, S_m\}\) on an alphabet \(\Sigma\) of size \(\sigma\)
- \(|S_j| = n\) \(\forall j \in [1..m]\)

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 \(m!\) \cite{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

\[
\begin{align*}
S_1 &= \text{lineeplessnles} \\
S_2 &= \text{sleepslessness} \\
S_3 &= \text{nleeplessness} \\
S_4 &= \text{sneeplessness} \\
S_5 &= \text{sleellessnness} \\
T &= \text{sleeplessness}
\end{align*}
\]

actually same problem and solution with the scores!
why this problem?

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

from https://upload.wikimedia.org/wikipedia/commons/a/a0/Stringed_Instruments.jpg
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

“\n“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 program 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:

```prolog
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 \neq A.
4 cost(X,C) :- C = \#\text{sum} \{1,I : c(X,I)\}, mat(X,\_).
5 mcost(M) :- M = \#\text{max} \{C : cost(_,C)\}.
6 #\text{minimize} \{M : mcost(M)\}.
7 #\text{show} t/2. #\text{show} mcost/1. #\text{show} cost/2.
```
how does the input look like?

transform texts

- $S_1 = \text{lneeplellness}$
- ...
- $S_5 = \text{sllleeelnessss}$

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. $s(4, 12, 115)$. 
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

```prolog
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 = #\text{sum} \{1,I : c(X,I)\}, mat(X, _).
5  mcost(M) :- M = #\text{max} \{C : cost(_,C)\}.
6  #\text{minimize} \{M : mcost(M)\}.
7  #\text{show} t/2. #\text{show} mcost/1. #\text{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 $T_{i,c} = 1 \iff T[i] = c$
- state that $T[i] = S_x[i]$, i.e., only one $T_{i,c}$ is set:

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

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

complexity $(x,y)$:
- $x$ : # clauses
- $y$ : # variables per clause

1. `mat(X,I) :- s(X,I,_)`.
2. `1 \{t(I,C) : s(_,I,C)\} 1 :- mat(_,I).` (highlighted)
3. `c(X,I) :- t(I,C), s(X,I,A), C \neq 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.`
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 $S_x$

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

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 \neq 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.
maximum of summed costs

- add helper variables
  \[ \text{cost}_x := \sum_{i \in [1..n]} C_{i,x} \]
- and compute the maximum value
  \[ \text{mcost} := \max\{\text{cost}_1, \ldots, \text{cost}_m\} \]

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mat(X,I) :- s(X,I,_).</td>
</tr>
<tr>
<td>2</td>
<td>1 {t(I,C) : s(_,I,C)} 1 :- mat(_,I).</td>
</tr>
<tr>
<td>3</td>
<td>c(X,I) :- t(I,C), s(X,I,A), C != A.</td>
</tr>
<tr>
<td>4</td>
<td>cost(X,C) :- C = #sum {1,I : c(X,I)}, mat(X,_).</td>
</tr>
<tr>
<td>5</td>
<td>mcost(M) :- M = #max {C : cost(_,C)}.</td>
</tr>
<tr>
<td>6</td>
<td>#minimize {M : mcost(M)}.</td>
</tr>
<tr>
<td>7</td>
<td>#show t/2. #show mcost/1. #show cost/2.</td>
</tr>
</tbody>
</table>
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:

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

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

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

1. $\text{mat}(X,I) :- \text{s}(X,I,\_)$.
2. $\{t(I,C) : \text{s}(_,I,C)\} 1 :- \text{mat}(_,I)$.
3. $\text{c}(X,I) :- t(I,C), \text{s}(X,I,A), C \neq A$.
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. $\#\text{show } t/2. \#\text{show } \text{mcost}/1. \#\text{show } \text{cost}/2.$
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  {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 \( T = \text{sleeplessness} \)
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

<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>

s05m07n009i0 denotes:
- $\sigma = 5$
- $m = 7$
- $n = 9$
- $i = 0$-th sample (iteration)

**Columns:**
- $x = \text{mcost}$
- $[s]: \text{time in seconds}$

**Observation:**
- \# choices correlates with time
- ASP has much fewer to check
but wait...

... if there are good solutions like 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!
Closest Substring

- parameter $\lambda$: length of the output string $T$: $|T| = \lambda$
- objective: minimize $\max_{x \in [1..m]} \text{dist}_\lambda(S_x, T)$

where $\text{dist}_\lambda(S_x, T) := \min_{i \in [1..n-\lambda+1]} \text{dist}_{\text{ham}}(S_x[i..i+\lambda-1], T)$: alignment score
Closest Substring example for $\lambda = 4$

\begin{align*}
S_1 &= s l e s n l e s s p e s s \\
S_2 &= s n e l p e l l n e s s s \\
S_3 &= s s s s s s s l p s p e s s \\
S_4 &= p s e l n e s e e l s e s \\
S_5 &= n e s s s s l s n e l e s s \\
\end{align*}

- task: compute a solution for $\lambda = 4$
- idea: shift $S_j$ and compute \textsc{closest substring} for the first $\lambda$ characters

Gramm+’03
Closest Substring example for $\lambda = 4$

$S_1 = s l e s n l e s s p e s s$

$S_2 = s n e l p e l l n e s s s$

$S_3 = s s s s s s l p s p e s s$

$S_4 = p s e l n e s e e l s e s$

$S_5 = n e s s s l s n e l e s s$

$T = s n e s$

Output has distance 1 to all input strings

https://upload.wikimedia.org/wikipedia/commons/2/24/Super_Nintendo_Entertainment_System-USA.jpg
modelling input

same startup, but also need to set $\lambda = 4$ via la(4).

```
1 mat(X,I) :- s(X,I,_).
2 1 {d(X,D) : D = 0..n-la} 1 :- mat(X,0).
3 si(I,C) :- s(X,J,C), d(X,D), J-D>=0, I=J-D.
4 1 {t(I,C) : si(I,C)} 1 :- mat(_,I), I < la.
5 c(X,I) :- t(I,C), s(X,J,A), d(X,D), I+D==J, I < la, A != C.
6 cost(X,C) :- C=#sum {1,I:c(X,I)}, mat(X,_).
7 mcost(M) :- M = #max {C : cost(_,C)}.
8 #minimize {M : mcost(M)}.
9 #show t/2. #show mcost/1. #show cost/2.
```

for space reasons: $\sigma(gma) = \sigma$, $\lambda(mbda) = \lambda$
modelling shifts

- select shifts \(d_x \in [0..n - \lambda]\) of each input string \(S_x\) such that the CSP of \(\{S_1[1 + d_1..\lambda + d_1], \ldots, S_m[1 + d_m..\lambda + d_m]\}\) is a solution of CSS if we take the minimum distance over all shifts \(d_x\)

- represent the shifts by a matrix of selectable Boolean variables of size \(O(m(n - \lambda))\)

\[
\begin{align*}
1 & \text{mat}(X,I) :- \text{s}(X,I,\_). \\
2 & \{d(X,D) : D = 0..n-\lambda\} 1 :- \text{mat}(X,0). \\
3 & \text{si}(I,C) :- \text{s}(X,J,C), d(X,D), J-D>=0, I=J-D. \\
4 & \{t(I,C) : \text{si}(I,C)\} 1 :- \text{mat}(\_,I), I < \lambda. \\
5 & \text{c}(X,I) :- t(I,C), \text{s}(X,J,A), d(X,D), I+D==J, I < \lambda, A != C. \\
6 & \text{cost}(X,C) :- C=\#\text{sum} \{1,I:\text{c}(X,I)\}, \text{mat}(X,\_). \\
7 & \text{mcost}(M) :- M = \#\text{max} \{C : \text{cost}(\_,C)\}. \\
8 & \#\text{minimize} \{M : \text{mcost}(M)\}. \\
9 & \#\text{show} t/2. \#\text{show} \text{mcost}/1. \#\text{show} \text{cost}/2.
\end{align*}
\]

for space reasons: \(\text{si}(\text{gma}) = \sigma, \lambda(\text{mbda}) = \lambda\)
modelling alphabet

redefine the alphabet for the
i-th character to be

$$\Sigma_i := \{S_1[i+d_1], \ldots, S_m[i+d_m]\}$$

```prolog
1 mat(X,I) :- s(X,I,_).
2 1 \{d(X,D) : D = 0..n-la\} 1 :- mat(X,0).
3 si(I,C) :- s(X,J,C), d(X,D), J-D>=0, I=J-D.
4 1 \{t(I,C) : si(I,C)\} 1 :- mat(_,I), I < la.
5 c(X,I) :- t(I,C), s(X,J,A), d(X,D), I+D==J,
        I < la, A != C.
6 cost(X,C) :- C=#sum \{1,I:c(X,I)\}, mat(X,_).
7 mcost(M) :- M = #max \{C : cost(_,C)\}.
8 #minimize \{M : mcost(M)\}.
9 #show t/2. #show mcost/1. #show cost/2.
```

for space reasons: si(gma) = \sigma, la(mbda) = \lambda
modelling output $T$

- define variable $T_{i,c}$ as before
- but $\#$ clauses is $O(\lambda)$ since $|T| = \lambda$ (before $|T| = n$)

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

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

1. $\text{mat}(X,I) :- \text{s}(X,I,\_).$
2. $1 \{\text{d}(X,D) : D = 0..n-\lambda\} 1 :- \text{mat}(X,0).$
3. $\text{si}(I,C) :- \text{s}(X,J,C), \text{d}(X,D), J-D\geq0, I=J-D.$
4. $1 \{t(I,C) : \text{si}(I,C)\} 1 :- \text{mat}(\_,I), I < \lambda.$
5. $\text{c}(X,I) :- t(I,C), \text{s}(X,J,A), \text{d}(X,D), I+D==J, I < \lambda, A \neq C.$
6. $\text{cost}(X,C) :- C=\#\text{sum} \{1,I:\text{c}(X,I)\}, \text{mat}(X,\_).$
7. $\text{mcost}(M) :- M = \#\text{max} \{C : \text{cost}(\_,C)\}.$
8. $\#\text{minimize} \{M : \text{mcost}(M)\}.$
9. $\#\text{show} t/2. \#\text{show mcost}/1. \#\text{show} \text{cost}/2.$

for space reasons: $\text{si}(\text{gma}) = \sigma$, $\text{la}(\text{mbda}) = \lambda$
modelling costs

for costs we need to take shifts into consideration

\[ \forall i \in [1..\lambda], c \in \Sigma_i, x \in [1..m] : \]
\[ T_{i,c} \land S_x[i + d_x] \neq c \implies C_{i,x} \]

[\( O(\lambda nm\sigma), O(1) \)]

additional \( n \)-term in #clauses because offsets \( d_x \in [1..n] \) given by two-dimensional binary array \( D[x, \ell] = 1 \iff d_x = \ell \)

1 mat(X,I) :- s(X,I,_).
2 1 \{d(X,D) : D = 0..n-la} 1 :- mat(X,0).
3 si(I,C) :- s(X,J,C), d(X,D), J-D>=0, I=J-D.
4 1 \{t(I,C) : si(I,C)} 1 :- mat(_,I), I < la.
5 c(X,I) :- t(I,C), s(X,J,A), d(X,D), I+D==J, I < la, A != C.
6 cost(X,C) :- C=#sum \{1,I:c(X,I)}, mat(X,_).
7 mcost(M) :- M = #max \{C : cost(_,C)}.
8 #minimize \{M : mcost(M)}.
9 #show t/2. #show mcost/1. #show cost/2.

for space reasons: \( si(gma) = \sigma, la(mbda) = \lambda \)
complexity

- $\mathcal{O}(\lambda \sigma + m(n - \lambda))$
  - selectable variables: $T_{i,c}$ and $d_x$
- $\mathcal{O}(\lambda m)$ helper variables: $C_{i,x}$
- $\mathcal{O}(\lambda mn\sigma)$ clauses
- largest clause size: $\mathcal{O}(\lambda m)$

for space reasons: $\text{si}(\text{gma}) = \sigma$, $\text{la}(\text{mbda}) = \lambda
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:

```
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.
```
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/asparagus

happy coding 🎵