Natural Language Generation

for Spoken Dialogue Systems

Ondřej Dušek

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University in Prague

May 14th, 2015
Overview

Outline of this talk

1. Introduction to NLG
   a) Textbook NLG pipeline
   b) How real systems differ

2. Examples of real NLG systems

3. Our NLG system
   a) Structure
   b) Experiments
   c) How to improve?
Introduction

Objective of NLG
Given (whatever) input and a communication goal, create a natural language string that is well-formed and human-like.

- Desired properties: variation, simplicity, trainability (?)

Usage

- Spoken dialogue systems
- Machine translation
- Short texts: Personalized letters, weather reports …
- Summarization
- Question answering in knowledge bases
Standard NLG Pipeline (*Textbook*)

[Input]
Standard NLG Pipeline (*Textbook*)

**[Input]**

↓ Content/text planning (“what to say”)
- Content selection, basic ordering

**[Content plan]**
Standard NLG Pipeline (*Textbook*)

**[Input]**

↓ Content/text planning ("what to say")

- Content selection, basic ordering

**[Content plan]**

↓ Sentence planning/microplanning ("middle ground")

- aggregation, lexical choice, referring…

**[Sentence plan(s)]**
Standard NLG Pipeline (*Textbook*)

**[Input]**

↓ Content/text planning ("what to say")

  • Content selection, basic ordering

**[Content plan]**

↓ Sentence planning/microplanning ("middle ground")

  • aggregation, lexical choice, referring…

**[Sentence plan(s)]**

↓ Surface realization ("how to say it")

  • linearization according to grammar

**[Text]**
Standard NLG Pipeline (*Textbook*)

**Inputs**

- Communication goal (e.g. “inform user about search results”)
- Knowledge base (e.g. list of matching entries in database, weather report numbers etc.)
- User model (constraints, e.g. user wants short answers)
- Dialogue history (referring expressions, repetition)
Standard NLG Pipeline (*Textbook*)

**Inputs**

- Communication goal (e.g. “inform user about search results”)
- Knowledge base (e.g. list of matching entries in database, weather report numbers etc.)
- User model (constraints, e.g. user wants short answers)
- Dialogue history (referring expressions, repetition)

**Content planning**

- Content selection according to communication goal
- Basic structuring (ordering)
Standard NLG Pipeline (*Textbook*)

**Sentence planning (micro-planning)**

- Word and syntax selection (e.g. choose templates)
- Dividing content into sentences
- Aggregation (merging simple sentences)
- Lexicalization
- Referring expressions
Standard NLG Pipeline (*Textbook*)

**Sentence planning (micro-planning)**
- Word and syntax selection (e.g. choose templates)
- Dividing content into sentences
- Aggregation (merging simple sentences)
- Lexicalization
- Referring expressions

**Surface realization**
- Creating linear text from (typically) structured input
- Ensuring syntactic correctness
Real NLG Systems

Few systems implement the whole pipeline

- Systems focused on content planning with trivial surface realization
- Surface-realization-only, word-order-only systems
- One-step (holistic) approaches
- SDS: content planning done by dialogue manager
Real NLG Systems

Few systems implement the whole pipeline

- Systems focused on content planning with trivial surface realization
- Surface-realization-only, word-order-only systems
- One-step (holistic) approaches
- SDS: content planning done by dialogue manager

Approaches

- Templates, Grammars, Rules, Statistics, or a mix thereof
Real NLG Systems

Few systems implement the whole pipeline

- Systems focused on content planning with trivial surface realization
- Surface-realization-only, word-order-only systems
- One-step (holistic) approaches
- SDS: content planning done by dialogue manager

Approaches

- Templates, Grammars, Rules, Statistics, or a mix thereof

Data representations

- Varied, custom-tailored, non-compatible
Trainable Sentence Planning: **SPoT**

- Spoken Dialogue System in the flight information domain
- Handcrafted generator + overgeneration
- Statistical reranker (RankBoost) trained on hand-annotated sentence plans

<table>
<thead>
<tr>
<th>Alt</th>
<th>Realization</th>
<th>H</th>
<th>RB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>What time would you like to travel on September the 1st to Dallas from Newark?</td>
<td>5</td>
<td>.85</td>
</tr>
<tr>
<td>5</td>
<td>Leaving on September the 1st. What time would you like to travel from Newark to Dallas?</td>
<td>4.5</td>
<td>.82</td>
</tr>
<tr>
<td>8</td>
<td>Leaving in September. Leaving on the 1st. What time would you, traveling from Newark to Dallas, like to leave?</td>
<td>2</td>
<td>.39</td>
</tr>
</tbody>
</table>
Trainable Sentence Planning: Parameter Optimization

- Requires a flexible handcrafted planner
- No overgeneration
- Adjusting its parameters “somehow”
Trainable Sentence Planning: Parameter Optimization

- Requires a flexible handcrafed planner
- No overgeneration
- Adjusting its parameters “somehow”

Examples

- *Paiva&Evans*: linguistic features annotated in corpus generated with many parameter settings, correlation analysis
- *PERSONAGE-PE*: personality traits connected to linguistic features via machine learning
Grammar-based Realizers (90's): \textit{KPML, FUF/SURGE}

\textbf{KPML}

- General purpose, multilingual
- Systemic Functional Grammar

\texttt{(EXAMPLE}
\begin{verbatim}
:NAME   EX-SET-1
:TARGETFORM  "It is raining cats and dogs."
:LOGICALFORM
  (A / AMBIENT-PROCESS :LEX RAIN
   :TENSE PRESENT-CONTINUOUS :ACTEE
   (C / OBJECT :LEX CATS-AND-DOGS :NUMBER MASS))
\end{verbatim}
\texttt{)}
Grammar-based Realizers (90's): KPML, FUF/SURGE

KPML
- General purpose, multilingual
- Systemic Functional Grammar

FUF/SURGE
- General purpose
- Functional Unification Grammar

EXAMPLE NLG Systems

Surface Realization

Input Specification ($I_1$):

Output Sentence ($S_1$): “She hands the draft to the editor”
Grammar-based Realizer: OpenCCG

- General purpose, multi-lingual
- Combinatory Categorial Grammar
- Used in several projects
- With statistical enhancements

\[
\begin{align*}
(>) & \quad X/Y \quad Y \quad \Rightarrow \quad X \\
(<) & \quad Y \quad X/Y \quad \Rightarrow \quad X \\
(>B) & \quad X/Y \quad Y/Z \quad \Rightarrow \quad X/Z \\
(<B) & \quad Y/Z \quad X/Y \quad \Rightarrow \quad X/Z \\
(>T) & \quad X \quad \Rightarrow \quad Y/(Y/X) \\
(<T) & \quad X \quad \Rightarrow \quad Y/(Y/X)
\end{align*}
\]

\[
\begin{align*}
\text{man} & \vdash n \\
\text{that} & \vdash (n\,n)/(s_{\text{form}=\text{fin}}/\text{np}) \\
\text{Bob} & \vdash \text{np} \\
\text{saw} & \vdash (s_{\text{tense}=\text{past},v\text{form}=\text{fin}}/\text{np})/\text{np}
\end{align*}
\]

Example NLG Systems

Surface Realization

\[
\begin{align*}
\text{be} & \quad [\text{tense=pres},\,\text{info=rh},\,\text{id=n1}] \\
<\text{Arg}> & \quad \text{flight} \quad [\text{num=sg},\,\text{det=the},\,\text{info=th},\,\text{id=f2}] \\
& \quad <\text{HasProp}> \quad \text{cheapest} \quad [\text{kon=}+\,\text{id=n2}] \\
<\text{Prop}> & \quad \text{has-rel} \quad [\text{id=n3}] \\
& \quad <\text{Of}>\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,\,...
\end{align*}
\]
Procedural Realizer: SimpleNLG

- General purpose
- English, adapted to several other languages
- Java implementation (procedural)

```java
Lexicon lexicon = new XMLLexicon("my-lexicon.xml");
NLGFactory nlgFactory = new NLGFactory(lexicon);
Realiser realiser = new Realiser(lexicon);

SPhraseSpec p = nlgFactory.createClause();
p.setSubject("Mary");
p.setVerb("chase");
p.setObject("the monkey");
p.setFeature(Feature.TENSE, Tense.PAST);

String output = realiser.realiseSentence(p);
System.out.println(output);

>>> Mary chased the monkey.
```
Trainable Realizers: Overgenerate and Rank

- Require a handcrafted realizer, e.g. CCG realizer
- Input underspecified → more outputs possible
- Overgenerate
- Then use a statistical reranker
Trainable Realizers: Overgenerate and Rank

- Require a handcrafted realizer, e.g. CCG realizer
- Input underspecified → more outputs possible
- Overgenerate
- Then use a statistical reranker
- Ranking according to:
  - $n$-gram models (*NITROGEN, HALOGEN*)
  - Tree models (XTAG grammar – *FERGUS*)
  - Predicted Text-to-Speech quality (*Nakatsu and White*)
  - Personality traits (extraversion, agreeableness… – *CRAG*)
  + alignment (repeating words uttered by dialogue counterpart)
Trainable Realizers: Overgenerate and Rank

- Require a handcrafted realizer, e.g. CCG realizer
- Input underspecified → more outputs possible
- Overgenerate
- Then use a statistical reranker
- Ranking according to:
  - $n$-gram models (NITROGEN, HALOGEN)
  - Tree models (XTAG grammar – FERGUS)
  - Predicted Text-to-Speech quality (Nakatsu and White)
  - Personality traits (extraversion, agreeableness… – CRAG)
    + alignment (repeating words uttered by dialogue counterpart)
- Provides variance, but at a greater computational cost
Trainable Realizers: Syntax-Based

- *StuMaBa*: general realizer based on SVMs
- Pipeline:
  - Deep syntax/semantics
  - surface syntax
  - linearization
  - morphologization
Holistic NLG

- Only one stage – no distinction
- “Good enough” for limited domains, also in SDS
Holistic NLG

- Only one stage – no distinction
- “Good enough” for limited domains, also in SDS

Template-based systems

- Most common, also in commercial NLG systems
- Simple, straightforward, reliable (custom-tailored for domain)
- Lack generality and variation, difficult to maintain
- Enhancements for more complex utterances: rules
Example: Templates

- Just filling variables into slots
- Possibly a few enhancements, e.g. articles

```plaintext
inform(pricerange="{pricerange}"):
'It is in the {pricerange} price range.'

affirm()&inform(task="find")
    &inform(pricerange="{pricerange}"):
'Ok, you are looking for something in the'
    + ' {pricerange} price range.'

affirm()&inform(area="{area}"):
'Ok, you want something in the {area} area.'

affirm()&inform(food="{food}")
    &inform(pricerange="{pricerange}"):
'Ok, you want something with the {food} food'
    + ' in the {pricerange} price range.'

inform(food="None"):
'I do not have any information'
    + ' about the type of food.'
```

Facebook templates

Alex (English restaurant domain)
Statistical Holistic NLG

- Limited domain
- Based on supervised learning
  (typically: MR + sentence + alignment)
- Typically: phrase-based
Statistical Holistic NLG

- Limited domain
- Based on supervised learning (typically: MR + sentence + alignment)
- Typically: phrase-based

Examples

- **BAGEL**: Bayesian networks
  - semantic stacks, ordering
- **Angeli et al.**: log-linear model
  - records \( \downarrow \) fields \( \downarrow \) templates
- **WASP\(^{-1}\)**: Synchronous CFGs
  - noisy channel, similar to MT
Our experiments: Two-Step NLG for SDS

Learning from unaligned data

• Typical NLG training:
  a) requires detailed alignments of MR elements and words/phrases
  b) uses a separate alignment step

\[
\text{inform(name=X, type=placetoeat, eattype=restaurant, area=riverside, food=Italian)}
\]

\[
\text{MR}
\]

\[
\text{alignment}
\]

\[
X \text{ is an italian restaurant in the riverside area .}
\]

\[
\text{text}
\]
Our experiments: Two-Step NLG for SDS

Learning from unaligned data

• Typical NLG training:
  a) requires detailed alignments of MR elements and words/phrases
  b) uses a separate alignment step

• Our generator learns alignments jointly
  • (with sentence planning)
  • training from pairs: MR + sentence

MR
inform(name=X, type=place to eat, eattype=restaurant, area=riverside, food=Italian)

X is an italian restaurant in the riverside area.

text
Overall workflow of our generator

- **Input**: a MR
  - here – dialogue acts: “inform” + slot-value pairs
  - other formats possible
Overall workflow of our generator

- **Input:** a MR
  - here – dialogue acts: “inform” + slot-value pairs
  - other formats possible

- **Step 1.** – sentence planning
  - statistical, our main focus
Overall workflow of our generator

- **Input**: a MR
  - here – dialogue acts: “inform” + slot-value pairs
  - other formats possible

- **Step 1.** – sentence planning
  - statistical, our main focus

- **Sentence plan**: deep-syntax dependency trees
  - based on *TectoMT's* t-layer, but very simplified
  - two attributes per tree node: *t-lemma* + *formeme*
  - using surface word order

- **Step 2.** – surface realization
  - reusing *Treex/TectoMT* English synthesis (rule-based)
Overall workflow of our generator

• **Input**: a MR
  • here – dialogue acts: “inform” + slot-value pairs
  • other formats possible

• **Step 1.** – sentence planning
  • statistical, our main focus

• **Sentence plan**: deep-syntax dependency trees
  • based on *TectoMT*’s t-layer, but very simplified
  • two attributes per tree node: t-lemma + formeme
  • using surface word order

• **Step 2.** – surface realization
  • reusing *Treex/TectoMT* English synthesis (rule-based)

• **Output**: plain text sentence
Data structures used

```
inform(name=X, type=placetoeat, eattype=restaurant, area=riverside, food=Italian)
```

X is an italian restaurant in the riverside area.
Why we keep the two-step approach

- It makes the 1st – statistical – task simpler
  - no need to worry about morphology
  - this will be more important for Czech (and similar)
Why we keep the two-step approach

• It makes the 1st – statistical – task simpler
  • no need to worry about morphology
  • this will be more important for Czech (and similar)

• The 2nd step – rule based – can ensure grammatical correctness
  • or at least it's more straightforward to fix when it doesn't
Why we keep the two-step approach

- It makes the 1st – statistical – task simpler
  - no need to worry about morphology
  - this will be more important for Czech (and similar)

- The 2nd step – rule based – can ensure grammatical correctness
  - or at least it's more straightforward to fix when it doesn't

- The realizer is (relatively) easy to implement and domain-independent
  - + why not use it if we have it already in Treex/TectoMT
Downside of the two-step approach

- We need to analyze training sentences into deep trees
Downside of the two-step approach

- We need to analyze training sentences into deep trees
  - but we can do it easily using Treex
    - t-layer analysis implemented for several languages
  - automatic annotation is good enough
Sentence planner – overall

- Two main components:
  - **candidate generator:**
    - churning out more and more sentence plan trees
  - **scorer/ranker** for the candidates
Sentence planner – overall

• Two main components:
  • **candidate generator:**
    – churning out more and more sentence plan trees
  • **scorer/ranker** for the candidates

• A*-style search
  • incrementally finding the path
    • from an empty tree
    • to a full sentence plan tree which contains all information
Sentence planner – overall

- Two main components:
  - **candidate generator:**
    - churning out more and more sentence plan trees
  - scorer/ranker for the candidates

- A*-style search
  - incrementally finding the path
    - from an empty tree
    - to a full sentence plan tree which contains all information
  - using open_set, close_set – heaps sorted by score
Sentence planner – workflow

- Init: open_set = {empty tree}, close_set = ∅
Sentence planner – workflow

• Init: \texttt{open\_set} = \{empty tree\}, \texttt{close\_set} = \emptyset
• Loop:
  1. get top-scoring \( C \leftarrow \texttt{open\_set} \)
     put \( C \rightarrow \texttt{close\_set} \)
Sentence planner – workflow

- **Init:** \( \text{open}_\text{set} = \{\text{empty tree}\} \), \( \text{close}_\text{set} = \emptyset \)
- **Loop:**
  1. get top-scoring \( C \leftarrow \text{open}_\text{set} \)
     put \( C \rightarrow \text{close}_\text{set} \)
  2. \( C = \text{candidate generator successors}(C) \)
     - viable trees, \( C \) + some node(s)
     - \( C \) may be empty
Sentence planner – workflow

- **Init:** open_set = \{empty tree\}, close_set = ∅
- **Loop:**
  1. get top-scoring \( C \leftarrow \text{open_set} \)
     put \( C \rightarrow \text{close_set} \)
  2. \( C = \text{candidate generator successors}(C) \)
     - viable trees, \( C + \text{some node(s)} \)
     - \( C \) may be empty
  3. score \( C' \land C' \in C \)
     put \( C' \rightarrow \text{open_set} \)
Sentence planner – workflow

- **Init:** $\text{open\_set} = \{\text{empty tree}\}$, $\text{close\_set} = \emptyset$
- **Loop:**
  1. get top-scoring $C \leftarrow \text{open\_set}$
     put $C \rightarrow \text{close\_set}$
  2. $C =$ candidate generator successors($C$)
     - viable trees, $C + \text{some node(s)}$
     - $C$ may be empty
  3. score $C' \forall C' \in C$
     put $C' \rightarrow \text{open\_set}$
  4. check if top score($\text{open\_set}$) > top score($\text{close\_set}$)

- **Stop if:**
  a) $\text{close\_set}$ has better top score than $\text{open\_set}$ for $d$ consecutive iterations
  b) there's nothing left on the open list (unlikely)
Candidate generator

• Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)
Candidate generator

- Given a candidate plan tree, generate its successors by adding 1 node (at every possible place)
Candidate generator – limiting the space

- Number of candidates very high even for small domains
- We need to lower the number of “possible” successors
Candidate generator – limiting the space

- Number of candidates very high even for small domains
- We need to lower the number of “possible” successors
- Limiting by things seen in training data:
  1. t-lemma + formeme combination
  2. parent–child combination
  3. number of children
  4. tree size
     - + at depth levels
     - + given input MR
  5. “weak” compatibility with input MR:
     - nodes seen with current slot-values
  6. “strong” compatibility with input MR:
     - required slot-values for each node
       (minimum seen in training data)
Scorer

- a function:
  
  sentence plan tree $t$, MR $m \rightarrow$ real-valued score
  
  - describes the fitness of $t$ for $m$
Scorer

- a function:

\[ \text{sentence plan tree } t, \text{ MR } m \rightarrow \text{ real-valued score} \]
- describes the fitness of \( t \) for \( m \)

Basic perceptron scorer

- \( \text{score} = \mathbf{w}^\top \cdot \text{feat}(t, m) \)
Scorer

• a function:
  sentence plan tree $t$, MR $m$ \to real-valued score
  • describes the fitness of $t$ for $m$

Basic perceptron scorer

• score $= \mathbf{w}^\top \cdot \text{feat}(t, m)$

• Training:
  • given $m$, generate the best tree $t_{top}$ with current weights
  • update weights if $t_{top} \neq t_{gold}$ (gold-standard)
Scorer

- a function:
  sentence plan tree $t$, MR $m \rightarrow$ real-valued score
  - describes the fitness of $t$ for $m$

Basic perceptron scorer

- $score = \mathbf{w}^\top \cdot \text{feat}(t, m)$
- Training:
  - given $m$, generate the best tree $t_{top}$ with current weights
  - update weights if $t_{top} \neq t_{gold}$ (gold-standard)
- Update: $\mathbf{w} = \mathbf{w} + \alpha \cdot (\text{feat}(t_{gold}, m) - \text{feat}(t_{top}, m))$
Differing subtree updates

- Features are global → bigger trees score better
  - need to promote “promising” incomplete trees
Differing subtree updates

- Features are global → bigger trees score better
  - need to promote “promising” incomplete trees
  - → promoting subtrees of gold-standard trees
  - + demoting subtrees of wrong generation outputs
Differing subtree updates

- Features are global $\rightarrow$ bigger trees score better
  - need to promote “promising” incomplete trees
  - $\rightarrow$ promoting subtrees of gold-standard trees
  - $+$ demoting subtrees of wrong generation outputs

- Update: find common subtree, start from it and update using pairs of subtrees $t^i_{\text{gold}}, t^i_{\text{top}}$
Differing subtree updates

- Features are global → bigger trees score better
  - need to promote “promising” incomplete trees
  - → promoting subtrees of gold-standard trees
  - + demoting subtrees of wrong generation outputs
- Update: find common subtree, start from it and update using pairs of subtrees $t^i_{gold}$, $t^i_{top}$
Differing subtree updates

- Features are global → bigger trees score better
  - need to promote “promising” incomplete trees
  - promoting subtrees of gold-standard trees
  - + demoting subtrees of wrong generation outputs

- Update: find common subtree, start from it and update using pairs of subtrees $t^i_{gold}$, $t^i_{top}$

```
Gold standard ($t_{gold}$):
```
```
Top generated $t_{top}$:
```

```
Common subtree ($t_c$)
```

```
Differing subtrees for update
```
```
t_1
```
```
t_{top}
```
```
t_{gold}
```
Differing subtree updates

- Features are global → bigger trees score better
  - need to promote “promising” incomplete trees
  - promoting subtrees of gold-standard trees
  - + demoting subtrees of wrong generation outputs

- Update: find common subtree, start from it and update using pairs of subtrees $t_{gold}^i$, $t_{top}^i$

Gold standard ($t_{gold}$):

- moderate adj:attr

Top generated $t_{top}$:

- cheap adj:attr
- italian adj:attr

Differing subtrees for update
Differing subtree updates

- Features are global → bigger trees score better
  - need to promote “promising” incomplete trees
  - → promoting subtrees of gold-standard trees
  - + demoting subtrees of wrong generation outputs

- Update: find common subtree, start from it and update using pairs of subtrees $t_{gold}^i$, $t_{top}^i$

Gold standard ($t_{gold}$):
- t-tree
- n:subj be v:fin restaurant
- n:obj range
- n:in+X
- n:attr moderate
- adj:attr

Top generated $t_{top}$:
- t-tree
- n:subj be v:fin restaurant
- n:obj cheap
- adj:attr
- n:attr italian
- adj:attr

+ regular full update
Future promise estimate

- Further boost for incomplete trees
Future promise estimate

- Further boost for incomplete trees
- Using expected number of children $E_c(n)$ of a node
Future promise estimate

- Further boost for incomplete trees
- Using expected number of children $E_c(n)$ of a node
- Future promise: “how many children are missing to meet the expectation”

$$fc = \sum_{n \in t} \max\{0, E_c(n) - c(n)\}$$
Future promise estimate

- Further boost for incomplete trees
- Using expected number of children \( E_c(n) \) of a node
- Future promise:
  “how many children are missing to meet the expectation”

\[
fc = \sum_{n \in t} \max\{0, E_c(n) - c(n)\}
\]

- over the whole tree
- + multiplied by feature sum
- + weighted
Future promise estimate

- Further boost for incomplete trees
- Using expected number of children $E_c(n)$ of a node
- Future promise:
  “how many children are missing to meet the expectation”

$$fc = \sum_{n \in t} \max\{0, E_c(n) - c(n)\}$$

- over the whole tree
- + multiplied by feature sum
- + weighted

- used on the open_set, not close_set
  - not for perceptron updates, not for stopping generation
Surface realizer overview

- English synthesis pipeline from *Treex/TectoMT*
  - domain-independent
Surface realizer overview

- English synthesis pipeline from Treex/TectoMT
  - domain-independent
- Mostly simple, single-purpose, rule-based modules (blocks)
  - Word inflection: statistical (Flect)
Surface realizer overview

- English synthesis pipeline from Treex/TectoMT
  - domain-independent
- Mostly simple, single-purpose, rule-based modules (blocks)
  - Word inflection: statistical (Flect)
- Gradual transformation of deep trees into surface dependency trees
  - Surface trees are then simply linearized
Surface realizer overview

- English synthesis pipeline from Treex/TectoMT
  - domain-independent
- Mostly simple, single-purpose, rule-based modules (blocks)
  - Word inflection: statistical (Flect)
- Gradual transformation of deep trees into surface dependency trees
  - Surface trees are then simply linearized
- Works OK: analysis → synthesis on our data = 89.79% BLEU
Surface realization example

- Realizer steps (simplified):
Surface realization example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
Surface realization example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
Surface realization example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions

Our System  Surface realizer

Natural Language Generation
Surface realization example

• Realizer steps (simplified):
  • Copy the deep tree (sentence plan)
  • Determine morphological agreement
  • Add prepositions and conjunctions
  • Add articles

Our System
Surface realizer

Natural Language Generation
Surface realization example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions
  - Add articles
  - Compound verb forms (add auxiliaries)
Surface realization example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions
  - Add articles
  - Compound verb forms (add auxiliaries)
  - Punctuation
Surface realization example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions
  - Add articles
  - Compound verb forms (add auxiliaries)
  - Punctuation
  - Word inflection
Surface realization example

- Realizer steps (simplified):
  - Copy the deep tree (sentence plan)
  - Determine morphological agreement
  - Add prepositions and conjunctions
  - Add articles
  - Compound verb forms (add auxiliaries)
  - Punctuation
  - Word inflection
  - Capitalization
Experiments – data set

- Restaurant recommendations from the BAGEL generator
  - restaurant location, food type, etc.
- 404 utterances for 202 input dialogue acts (DAs)
  - two paraphrases for each DA
Experiments – data set

• Restaurant recommendations from the BAGEL generator
  • restaurant location, food type, etc.
• 404 utterances for 202 input dialogue acts (DAs)
  • two paraphrases for each DA
• Alignment provided, but we don't use it
Experiments – data set

• Restaurant recommendations from the BAGEL generator
  • restaurant location, food type, etc.
• 404 utterances for 202 input dialogue acts (DAs)
  • two paraphrases for each DA
• Alignment provided, but we don't use it
• “Non-enumerable” information replaced by “X” symbol
  • restaurant names, postcodes, phone numbers etc.
Experiments – features

• Tailored for the input MR format
Experiments – features

• Tailored for the input MR format
• Basic feature types:
  • tree properties (size, depth…)
  • tree + input DA (nodes per slot-value pair…)
  • node features
  • input DA features (slots, values, pairs of slots)
  • node + input DA features
  • repeat features (repeated nodes/slots/values)
  • dependency features (parent-child)
  • siblings features (+DA)
  • bigram features (+DA)
Experiments – features

- Tailored for the input MR format
- Basic feature types:
  - tree properties (size, depth…)
  - tree + input DA (nodes per slot-value pair…)
  - node features
  - input DA features (slots, values, pairs of slots)
  - node + input DA features
  - repeat features (repeated nodes/slots/values)
  - dependency features (parent-child)
  - siblings features (+DA)
  - bigram features (+DA)
- Typical case: counts over whole tree
  - normalized
Results

- Using 10-fold cross-validation, measuring BLEU/NIST
  - training DAs never used for testing
  - using 2 paraphrases for BLEU/NIST measurements
Results

- Using 10-fold cross-validation, measuring BLEU/NIST
  - training DAs never used for testing
  - using 2 paraphrases for BLEU/NIST measurements

<table>
<thead>
<tr>
<th>Setup</th>
<th>BLEU</th>
<th>NIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic perceptron</td>
<td>54.24</td>
<td>4.643</td>
</tr>
<tr>
<td>+ diff-tree updates</td>
<td>58.70</td>
<td>4.876</td>
</tr>
<tr>
<td>+ future promise</td>
<td>59.89</td>
<td>5.231</td>
</tr>
</tbody>
</table>
Results

• Using 10-fold cross-validation, measuring BLEU/NIST
  • training DAs never used for testing
  • using 2 paraphrases for BLEU/NIST measurements

<table>
<thead>
<tr>
<th>Setup</th>
<th>BLEU</th>
<th>NIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic perceptron</td>
<td>54.24</td>
<td>4.643</td>
</tr>
<tr>
<td>+ diff-tree updates</td>
<td>58.70</td>
<td>4.876</td>
</tr>
<tr>
<td>+ future promise</td>
<td>59.89</td>
<td>5.231</td>
</tr>
</tbody>
</table>

• less than BAGEL's ~ 67% BLEU
Results

- Using 10-fold cross-validation, measuring BLEU/NIST
  - training DAs never used for testing
  - using 2 paraphrases for BLEU/NIST measurements

<table>
<thead>
<tr>
<th>Setup</th>
<th>BLEU</th>
<th>NIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic perceptron</td>
<td>54.24</td>
<td>4.643</td>
</tr>
<tr>
<td>+ diff-tree updates</td>
<td>58.70</td>
<td>4.876</td>
</tr>
<tr>
<td>+ future promise</td>
<td>59.89</td>
<td>5.231</td>
</tr>
</tbody>
</table>

- less than *BAGEL's* ~ 67% BLEU
- But:
  - we do not use alignments
  - our generator has to know when to stop (whether all information is already included)
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, eattype=restaurant, near=X-near, food=Continental, food=French)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a French and continental restaurant near X.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a French and continental restaurant near X.</td>
</tr>
</tbody>
</table>
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, eattype=restaurant, near=X-near, food=Continental, food=French)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a French and continental restaurant near X.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a French and continental restaurant near X.</td>
</tr>
<tr>
<td>Input DA</td>
<td>inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)</td>
</tr>
<tr>
<td>Reference</td>
<td>X restaurant is near X on the riverside.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is a restaurant in the riverside area near X.</td>
</tr>
</tbody>
</table>
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>Reference</th>
<th>Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inform(name=X-name, type=placetoeat, eattype=restaurant, near=X-near, food=Continental, food=French)</code></td>
<td><code>X is a French and continental restaurant near X.</code></td>
<td><code>X is a French and continental restaurant near X.</code></td>
</tr>
<tr>
<td><code>inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)</code></td>
<td><code>X restaurant is near X on the riverside.</code></td>
<td><code>X is a restaurant in the riverside area near X.</code></td>
</tr>
<tr>
<td><code>inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)</code></td>
<td><code>X is a moderately priced restaurant in X.</code></td>
<td><code>X is a restaurant in the X area.</code></td>
</tr>
</tbody>
</table>
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>Reference</th>
<th>Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inform(name=X-name, type=placetoeat, eattype=restaurant, near=X-near, food=Continental, food=French)</code></td>
<td>X is a French and continental restaurant near X.</td>
<td>X is a French and continental restaurant near X.</td>
</tr>
<tr>
<td><code>inform(name=X-name, type=placetoeat, area=riverside, near=X-near, eattype=restaurant)</code></td>
<td>X restaurant is near X on the riverside.</td>
<td>X is a restaurant in the riverside area near X.</td>
</tr>
<tr>
<td><code>inform(name=X-name, type=placetoeat, area=X-area, pricerange=moderate, eattype=restaurant)</code></td>
<td>X is a moderately priced restaurant in X.</td>
<td>X is a restaurant in the X area.</td>
</tr>
<tr>
<td><code>inform(name=X-name, type=placetoeat, eattype=restaurant, area=riverside, food=French)</code></td>
<td>X is a French restaurant on the riverside.</td>
<td>X is a French restaurant in the riverside area which serves French food.</td>
</tr>
</tbody>
</table>
### Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>inform(name=X-name, type=placetoeat, eattype=restaurant, pricerange=moderate, area=X-area, food=Contemporary, food=English)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>X is a moderately priced English contemporary restaurant in X.</td>
</tr>
<tr>
<td>Generated</td>
<td>X is an English restaurant in the X area which serves expensive food in the moderate price range located in X.</td>
</tr>
</tbody>
</table>
Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>Reference Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inform(name=X-name, type=placetoeat, eattype=restaurant, pricerange=moderate, area=X-area, food=Contemporary, food=English)</code></td>
<td>X is a moderately priced English contemporary restaurant in X. X is an English restaurant in the X area which serves expensive food in the moderate price range located in X.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input DA</th>
<th>Reference Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inform(name=X-name, type=placetoeat, eattype=restaurant, area=citycentre, near=X-near, food=&quot;Chinese takeaway&quot;, food=Japanese)</code></td>
<td>X is a Chinese takeaway and Japanese restaurant in the city centre near X. X is a Japanese restaurant in the centre of town near X and X.</td>
</tr>
</tbody>
</table>
## Example Outputs

<table>
<thead>
<tr>
<th>Input DA</th>
<th>Reference</th>
<th>Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>inform(name=X-name, type=placetoate, eattype=restaurant, pricerange=moderate, area=X-area, food=Contemporary, food=English)</td>
<td>X is a moderately priced English contemporary restaurant in X.</td>
<td>X is an English restaurant in the X area which serves expensive food in the moderate price range located in X.</td>
</tr>
<tr>
<td>inform(name=X-name, type=placetoate, eattype=restaurant, area=citycentre, near=X-near, food=“Chinese takeaway”, food=Japanese)</td>
<td>X is a Chinese takeaway and Japanese restaurant in the city centre near X.</td>
<td>X is a Japanese restaurant in the centre of town near X and X.---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inform(name=X-name, type=placetoate, pricerange=moderate, eattype=restaurant)</td>
<td>X is a restaurant that offers moderate price range.</td>
<td>X is a restaurant in the moderate price range.---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
Results

- The outputs are mostly fluent and meaningful/relevant
  - Sometimes identical to reference
  - More often original (unseen) paraphrases
Results

- The outputs are mostly fluent and meaningful/relevant
  - Sometimes identical to reference
  - More often original (unseen) paraphrases
- Alignment can be learnt together with sentence planning
Results

• The outputs are mostly fluent and meaningful/relevant
  • Sometimes identical to reference
  • More often original (unseen) paraphrases

• Alignment can be learnt together with sentence planning

• Differing tree updates + future promise bring significant improvements
Results

- The outputs are mostly fluent and meaningful/relevant
  - Sometimes identical to reference
  - More often original (unseen) paraphrases
- Alignment can be learnt together with sentence planning
- Differing tree updates + future promise bring significant improvements
- Errors:
  - information missing
  - information is repeated
  - irrelevant information
- \(\rightarrow\) Scoring should be improved (?)
What to do to make it better?

- Larger training set – better weight estimates
- Refine features?
- Using neural networks
  - no need for sophisticated features
  - probably will be faster
What to do to make it better?

- Larger training set – better weight estimates
- Refine features?
- Using neural networks
  - no need for sophisticated features
  - probably will be faster
- Any suggestions?
What to do to make it better?

• Larger training set – better weight estimates
• Refine features?
• Using neural networks
  • no need for sophisticated features
  • probably will be faster
• Any suggestions?

Thank you for your attention
Contact me:
odusek@ufal.mff.cuni.cz, office 424
References


BAGEL
Mairesse, F. et al. 2010. Phrase-based statistical language generation using graphical models and active learning. *ACL*

CRAG
Isard, A. et al. 2006. Individuality and alignment in generated dialogues. *INLG*

FERGUS

Flect

FUF/SURGE

HALOGEN
Langkilde-Geary, I. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. *INLG*

KPML

OpenCCG
White, M. and Baldrige, J. 2003. Adapting Chart Realization to CCG. *ENLG*

Nakatsu & White
Nakatsu, C. and White, M. 2006. Learning to say it well: reranking realizations by predicted synthesis quality. *COLING-ACL*

NITROGEN
References


RL-NLG  Rieser, V. and Lemon, O. 2010. Natural language generation as planning under uncertainty for spoken dialogue systems. *EMNLP*


SPoT  Walker, M. et al. 2001. SPoT: A trainable sentence planner. *NAACL*

StuMaBa  Bohnet, B. et al. 2010. Broad coverage multilingual deep sentence generation with a stochastic multi-level realizer. *COLING*

TectoMT  Žabokrtský, Z. et al. 2008. TectoMT: highly modular MT system with tectogrammatics used as transfer layer. *WMT*


http://ufal.cz/treex


Further Links

C. DiMarco's slides: https://cs.uwaterloo.ca/~jchampai/CohenClass.en.pdf
J. Moore's NLG course: http://www.inf.ed.ac.uk/teaching/courses/nlg/
NLG Systems Wiki: http://www.nl-g-wiki.org