Chapter 4: Features and Constraints

Posing a Constraint Satisfaction Problem

A CSP is characterized by

- A set of variables $V_1, V_2, \ldots, V_n$.
- Each variable V_i has an associated domain $\mathbf{D}_{V_i}$ of possible values.
- There are hard constraints on various subsets of the variables which specify legal combinations of values for these variables.
- A solution to the CSP is an assignment of a value to each variable that satisfies all the constraints.

Example: scheduling activities

- Variables: A, B, C, D, E that represent the starting times of various activities.
- Domains: $\mathbf{D}_A = \{1, 2, 3, 4\}$, $\mathbf{D}_B = \{1, 2, 3, 4\}$, $\mathbf{D}_C = \{1, 2, 3, 4\}$, $\mathbf{D}_D = \{1, 2, 3, 4\}$, $\mathbf{D}_E = \{1, 2, 3, 4\}$
- Constraints:

$$
\begin{aligned}
(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge \\
(C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge \\
(E < C) \wedge (E < D) \wedge (B \neq D).
\end{aligned}
$$

Example: scheduling activities

- Variables: A, B, C, D, E that represent the starting times of various activities.
- Domains: $\mathbf{D}_A = \{1,2,3,4\}$, $\mathbf{D}_B = \{1,2,3,4\}$, $\mathbf{D}_C = \{1,2,3,4\}$, $\mathbf{D}_D = \{1,2,3,4\}$, $\mathbf{D}_E = \{1,2,3,4\}$
- Constraints:

$$
\begin{aligned}
&(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge \\
&\quad (C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge \\
&\quad (E < C) \wedge (E < D) \wedge (B \neq D).
\end{aligned}
$$

- Other problems that can be recasted as features and their admissible value assignments?

Solution procedures

- Generate-and test
- Graph search
- Domain and arc consistency
- Variable elimination

Generate-and-Test Algorithm

- Generate the assignment space $\mathbf{D} = \mathbf{D}_{V_1} \times \mathbf{D}_{V_2} \times \ldots \times \mathbf{D}_{V_n}$. Test each assignment with the constraints.
- Example:

$$\begin{aligned}\mathbf{D} &= \mathbf{D}_A \times \mathbf{D}_B \times \mathbf{D}_C \times \mathbf{D}_D \times \mathbf{D}_E \\ &= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \\ &\quad \times \{1,2,3,4\} \times \{1,2,3,4\} \\ &= \{\langle 1,1,1,1,1\rangle, \langle 1,1,1,1,2\rangle, ..., \langle 4,4,4,4,4\rangle\}.\end{aligned}$$

- How many assignments need to be tested for n variables each with domain size d?

Backtracking Algorithms

- Systematically explore **D** by instantiating the variables one at a time
- evaluate each constraint predicate as soon as all its variables are bound
- any partial assignment that doesn't satisfy the constraint can be pruned.

Example Assignment $A = 1 \wedge B = 1$ is inconsistent with constraint $A \neq B$ regardless of the value of the other variables.

CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment of values to some of the variables.
- Suppose node N is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable Y that isn't assigned in N.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.

Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent?

Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent?
  $\mathbf{D}_B = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.

Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent?
  $\mathbf{D}_B = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.
- What should we do, if a variable is not domain consistent?

Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the node is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent?

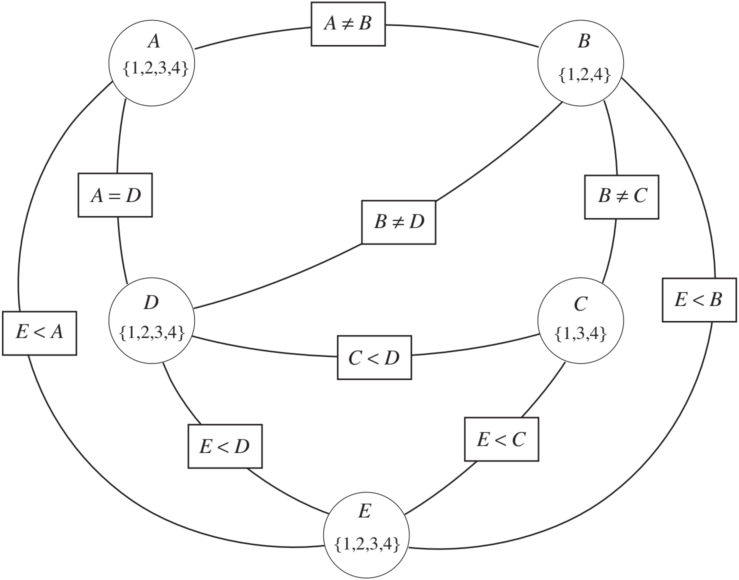
  $\mathbf{D}_B = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.
- What should we do, if a variable is not domain consistent?

  Remove all the values from the domain that violate some constraint.

Constraint Network

- There is a oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each constraint that involves X.

Example Constraint Network



Arc Consistency

- An arc $\langle X, r(X, \overline{Y})\rangle$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied.
- A network is arc consistent if all its arcs are arc consistent.
- What should we do, if arc $\langle X, r(X, \overline{Y})\rangle$ is *not* arc consistent?

Arc Consistency

- An arc $\langle X, r(X, \overline{Y}) \rangle$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied.
- A network is arc consistent if all its arcs are arc consistent.
- What should we do, if arc $\langle X, r(X, \overline{Y}) \rangle$ is *not* arc consistent?

  All values of X in $dom(X)$ for which there is no corresponding value in $dom(\overline{Y})$ can be deleted from $dom(X)$ to make the arc $\langle X, r(X, \overline{Y}) \rangle$ consistent.

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

  An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

  An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty
  - Each domain has a single value
  - Some domains have more than one value

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

  An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Longrightarrow$ no solution
  - Each domain has a single value
  - Some domains have more than one value

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

  An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Longrightarrow$ no solution
  - Each domain has a single value $\Longrightarrow$ unique solution
  - Some domains have more than one value

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

  An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Longrightarrow$ no solution
  - Each domain has a single value $\Longrightarrow$ unique solution
  - Some domains have more than one value $\Longrightarrow$ there may or may not be a solution

Finding solutions when AC finishes

- If some domains have more than one element $\Longrightarrow$ search
- Split a domain, then recursively solve each half.
  - It is often best to split a domain in half.
- Eliminate the variables one-by-one passing their constraints to their neighbours
  - Solve the simplified problem
  - Reintegrate the eliminated variable

Variable elimination

- If there is only one variable, return the intersection of the (unary) constraints that contain it
- select a variable X
- compute all binary relations R_1 ... R_n of that variable with its neighbouring variables X_1 ... X_n in the constraint graph
- join these relations $R = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- project the join to the remaining variables $R' = \pi_X R$
- call variable elimination recursively without X
- join the result with R

Variable elimination

Example:

- Variables: A, B, C, D
- Domains: $\mathbf{D}_A = \mathbf{D}_B = \mathbf{D}_C = \mathbf{D}_D = \{1, 2, 3, 4, 5\}$
- Constraints: $(A < B) \wedge (B < C) \wedge (C < D)$

Variable elimination

Eliminating B

R_{AB}

A	B
1	2
1	3
1	4
1	5
2	3
2	4
2	5
3	4
3	5
4	5

$\bowtie$

R_{BC}

B	C
1	2
1	3
1	4
1	5
2	3
2	4
2	5
3	4
3	5
4	5

$=$

R_{ABC}

A	B	C
1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

$\pi_B(R_{ABC}) =$

R_{AC}

A	C
1	3
1	4
1	5
2	4
2	5
3	5

Variable elimination

Combining with the remaining constraints (which do not involve B)

R_{AC}

A	C
1	3
1	4
1	5
2	4
2	5
3	5

$\bowtie$

R_{CD}

C	D
1	2
1	3
1	4
1	5
2	3
2	4
2	5
3	4
3	5
4	5

$=$

R_{ACD}

A	C	D
1	3	4
1	3	5
1	4	5
2	4	5

Variable elimination

Re-integrating the eliminated variable

R_{ACD}

A	C	D
1	3	4
1	3	5
1	4	5
2	4	5

$\bowtie$

R_{ABC}

A	B	C
1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

$=$

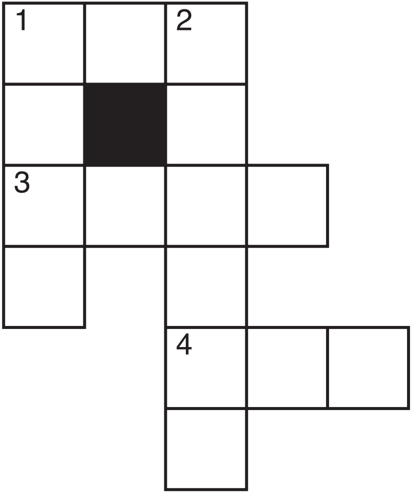
R_{ABCD}

A	B	C	D
1	2	3	4
1	2	3	5
1	2	4	5
1	3	4	5
2	3	4	5

Variable elimination

- If any join is empty: no solution exists
- If only a single solution is needed, an arbitrary tuple of the join can be returned
- The efficiency of the algorithm depends on the order in which the variables are selected
  - finding the optimal elimination sequence is NP hard
- Heuristics: always select the variable
  - which results in the smallest relation, or
  - which adds the smallest number of arcs to the constraint network
- variable elimination can be combined with arc consistency

How to formulate a problem as a CSP?



Words:

ant, big, bus, car, has
book, buys, hold,
lane, year
beast, ginger, search,
symbol, syntax

Hard and Soft Constraints

- Given a set of variables, assign a value to each variable that either
  - satisfies some set of constraints: satisfiability problems — "hard constraints"
  - minimizes some cost function, where each assignment of values to variables has some cost: optimization problems — "soft constraints"
- Many problems are a mix of hard and soft constraints (called constrained optimization problems).

Local Search

Local Search (Greedy Descent):

- Maintain an assignment of a value to each variable.
- Repeat:
  - Select a variable to change
  - Select a new value for that variable
- Until a satisfying assignment is found

Local Search for CSPs

- Aim: find an assignment with zero unsatisfied constraints.
- Given an assignment of a value to each variable, a conflict is an unsatisfied constraint.
- The goal is an assignment with zero conflicts.
- Heuristic function to be minimized: the number of conflicts.

Greedy Descent Variants

How to choose a variable to change and its new value?

- Find a variable-value pair that minimizes the number of conflicts
- Select a variable that participates in the most conflicts.
  Select a value that minimizes the number of conflicts.
- Select a variable that appears in any conflict.
  Select a value that minimizes the number of conflicts.
- Select a variable at random.
  Select a value that minimizes the number of conflicts.
- Select a variable and value at random; accept this change if it doesn't increase the number of conflicts.

Complex Domains

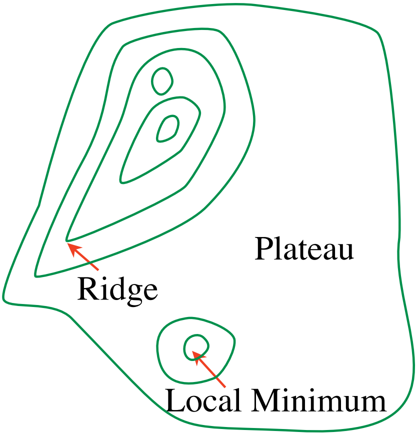
- When the domains are small or unordered, the neighbors of an assignment corresponds to choosing another value for any of the variables.
- When the domains are large and ordered, the neighbors of an assignment are the adjacent values for one of the variables.
- If the domains are continuous, Gradient descent changes each variable proportionally to the gradient of the heuristic function in that direction.
  The value of variable X_i is updated according to

$$v_i' = v_i - \eta \frac{\partial h}{\partial X_i}$$

  η is the step size.

Problems with Greedy Descent

- a local minimum that is not a global minimum
- a plateau where the heuristic values are uninformative
- an inverse ridge (i.e. an alley) which leads the search in the wrong direction
  $\rightarrow$ Ignorance of the peak



Randomized Algorithms

- Consider two methods to find a minimum value:
  - Greedy descent, starting from some position, keep moving down & report minimum value found
  - Pick values at random & report minimum value found
- Which do you expect to work better to find a global minimum?
- Can a mix work better?

Randomized Greedy Descent

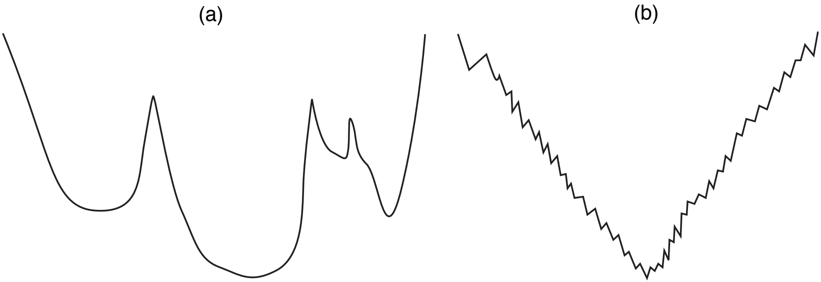
As well as downward steps we can allow for:

- Random steps: move to a random neighbor.
- Random restart: reassign random values to all variables.

Which is more expensive computationally?

1-Dimensional Ordered Examples

Two 1-dimensional search spaces; step right or left:



- Which method would most easily find the global minimum?
- What happens in hundreds or thousands of dimensions?
- What if different parts of the search space have different structure?

Stochastic Local Search

Stochastic local search is a mix of:

- Greedy descent: move to a lowest neighbor
- Random walk: taking some random steps
- Random restart: reassigning values to all variables

Random Walk

Variants of random walk:

- When choosing the best variable-value pair, randomly sometimes choose a random variable-value pair.
- When selecting a variable then a value:
  - Sometimes choose any variable that participates in the most conflicts.
  - Sometimes choose any variable that participates in any conflict (a red node).
  - Sometimes choose any variable.
- Sometimes choose the best value and sometimes choose a random value.

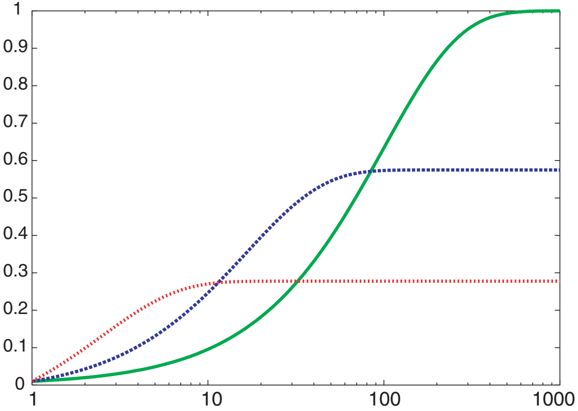
Comparing Stochastic Algorithms

- How can you compare three algorithms when
  - one solves the problem 30% of the time very quickly but doesn't halt for the other 70% of the cases
  - one solves 60% of the cases reasonably quickly but doesn't solve the rest
  - one solves the problem in 100% of the cases, but slowly?

Comparing Stochastic Algorithms

- How can you compare three algorithms when
  - one solves the problem 30% of the time very quickly but doesn't halt for the other 70% of the cases
  - one solves 60% of the cases reasonably quickly but doesn't solve the rest
  - one solves the problem in 100% of the cases, but slowly?
- Summary statistics, such as mean run time, median run time, and mode run time don't make much sense.

- Plots runtime (or number of steps) and the proportion (or number) of the runs that are solved within that runtime.



Variant: Simulated Annealing

- Pick a variable at random and a new value at random.
- If it is an improvement, adopt it.
- If it isn't an improvement, adopt it probabilistically depending on a temperature parameter, T.
  - With current assignment n and proposed assignment n' we move to n' with probability $e^{(h(n')-h(n))/T}$
- Temperature can be reduced.

Variant: Simulated Annealing

- Pick a variable at random and a new value at random.
- If it is an improvement, adopt it.
- If it isn't an improvement, adopt it probabilistically depending on a temperature parameter, T.
  - With current assignment n and proposed assignment n' we move to n' with probability $e^{(h(n')-h(n))/T}$
- Temperature can be reduced.

Probability of accepting a change:

Temperature	1-worse	2-worse	3-worse
10	0.91	0.81	0.74
1	0.37	0.14	0.05
0.25	0.02	0.0003	0.000006
0.1	0.00005	2×10^{-9}	9×10^{-14}

Tabu lists

- To prevent cycling we can maintain a tabu list of the k last assignments.
- Don't allow an assignment that is already on the tabu list.
- If $k = 1$, we only don't allow the immediate reassignment of the same value to the variable chosen.
- We can implement it more efficiently than as a list of complete assignments.
- It can be expensive if k is large.

Parallel Search

A total assignment is called an individual.

- Idea: maintain a population of k individuals instead of one.
- At every stage, update each individual in the population.
- Whenever an individual is a solution, it can be reported.
- Like k restarts, but uses k times the minimum number of steps.

Beam Search

- Like parallel search, with k individuals, but choose the k best out of all of the neighbors.
- When $k = 1$, it is greedy descent.
- When $k = \infty$, it is breadth-first search.
- The value of k lets us limit space and parallelism.

Stochastic Beam Search

- Like beam search, but it probabilistically chooses the k individuals at the next generation.
- The probability that a neighbor is chosen is proportional to its heuristic value.
- This maintains diversity amongst the individuals.
- The heuristic value reflects the fitness of the individual.
- Like asexual reproduction: each individual mutates and the fittest ones survive.

Genetic Algorithms

- Like stochastic beam search, but pairs of individuals are combined to create the offspring:
- For each generation:
  - Randomly choose pairs of individuals where the fittest individuals are more likely to be chosen.
  - For each pair, perform a cross-over: form two offspring each taking different parts of their parents:
  - Mutate some values.
- Stop when a solution is found.

Crossover

- Given two individuals:

$$X_1 = a_1, X_2 = a_2, \ldots, X_m = a_m$$

$$X_1 = b_1, X_2 = b_2, \ldots, X_m = b_m$$

- Select i at random.
- Form two offspring:

$$X_1 = a_1, \ldots, X_i = a_i, X_{i+1} = b_{i+1}, \ldots, X_m = b_m$$

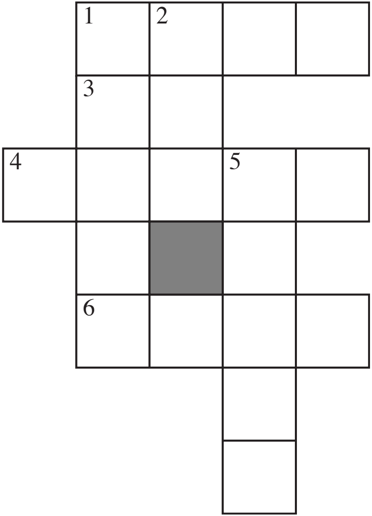
$$X_1 = b_1, \ldots, X_i = b_i, X_{i+1} = a_{i+1}, \ldots, X_m = a_m$$

- The effectiveness depends on the ordering of the variables.
- Many variations are possible.

Constraint satisfaction revisited

- A Constraint Satisfaction problem consists of:
  - a set of variables
  - a set of possible values, a domain for each variable
  - a set of constraints amongst subsets of the variables
- The aim is to find a set of assignments that satisfies all constraints, or to find all such assignments.

Example: crossword puzzle



at, be, he, it, on,
eta, hat, her, him, one,
desk, dove, easy, else,
help, kind, soon, this,
dance, first, fuels, given,
haste, loses, sense, sound,
think, usage

Dual Representations

Two ways to represent the crossword as a CSP

- First representation:
  - nodes represent word positions: 1-down…6-across
  - domains are the words
  - constraints specify that the letters on the intersections must be the same.
- Dual representation:
  - nodes represent the individual squares
  - domains are the letters
  - constraints specify that the words must fit

Representations for image interpretation

- First representation:
  - nodes represent the chains and regions
  - domains are the scene objects
  - constraints correspond to the intersections and adjacency
- Dual representation:
  - nodes represent the intersections
  - domains are the intersection labels
  - constraints specify that the chains must have same marking

Natural Language Processing as Constraint Satisfaction

- Agreement
- Linear Order and Optionality
- Structural Interpretation

Agreement

- In many languages word forms have to agree with respect to different morpho-syntactic features

ein kleiner Baum	der kleine Baum	die kleinen Bäume
eine kleine Blume	die kleine Blume	die kleinen Blumen
ein kleines Gras	das kleine Gras	die kleinen Gräser

- Usually the assignment of feature values is highly ambiguous

	number	gender	case
die	sing $\vee$ plur	masc $\vee$ fem $\vee$ neutr	nom $\vee$ acc
großen	sing $\vee$ plur	masc $\vee$ fem $\vee$ neutr	nom $\vee$ gen $\vee$ dat $\vee$ acc
Teller	sing $\vee$ plur	masc	nom $\vee$ gen $\vee$ dat $\vee$ acc

Agreement

- Lexical constraints: Only some of the possible feature value combinations are valid ones
- Lexical constraints can be extensionally specified

die	$\langle$sing, fem, nom$\rangle$	$\vee$	$\langle$sing, fem, acc$\rangle$	$\vee$
	$\langle$plur, masc, nom$\rangle$	$\vee$	$\langle$plur, masc, acc$\rangle$	$\vee$
	$\langle$plur, fem, nom$\rangle$	$\vee$	$\langle$plur, fem, acc$\rangle$	$\vee$
	$\langle$plur, neutr, nom$\rangle$	$\vee$	$\langle$plur, neutr, acc$\rangle$	
großen	$\langle$sing, masc, gen$\rangle$	$\vee$	$\langle$sing, masc, dat$\rangle$	$\vee$
	$\langle$sing, fem, gen$\rangle$	$\vee$	$\langle$sing, fem, dat$\rangle$	$\vee$
	$\langle$sing, neutr, gen$\rangle$	$\vee$	$\langle$sing, neutr, dat$\rangle$	$\vee$
	$\langle$plur, masc, nom$\rangle$	$\vee$	$\langle$plur, masc, gen$\rangle$	$\vee$
	$\langle$plur, masc, dat$\rangle$		$\vee$ ...	
Teller	$\langle$sing, masc, nom$\rangle$	$\vee$	$\langle$sing, masc, dat$\rangle$	$\vee$
	$\langle$sing, masc, acc$\rangle$	$\vee$	$\langle$plur, masc, nom$\rangle$	$\vee$
	$\langle$plur, masc, gen$\rangle$	$\vee$	$\langle$plur, masc, acc$\rangle$	

Agreement

- ... or as a single logical expression

die	fem ∧ sing ∧ (nom ∨ acc) ∨ plur ∧ (masc ∨ fem ∨ neutr) ∧ (nom ∨ acc)
großen	sing ∧ (gen ∨ dat) ∧ (masc ∨ fem ∨ neutr) ∨ plur ∧ (masc ∨ fem ∨ neutr) ∧ (nom ∨ gen ∨ dat ∨ acc)
Teller	masc ∧ (sing ∧ (nom ∨ dat ∨ acc) ∨ plur ∧ (nom ∨ gen ∨ acc)))

Agreement

- … or as separate constraints

die	$masc \vee fem \vee neutr$
	$sing \vee plur$
	$nom \vee acc$
	$sing \rightarrow fem$
großen	$nom \vee gen \vee dat \vee acc$
	$masc \vee fem \vee neutr$
	$sing \vee plur$
	$sing \wedge masc \rightarrow gen \vee dat \vee acc$
	$sing \wedge (fem \vee neutr) \rightarrow gen \vee dat$
Teller	$masc$
	$sing \vee plur$
	$sing \rightarrow nom \vee dat \vee acc$
	$plur \rightarrow nom \vee gen \vee acc$

Agreement

- Agreement constraints: require two or more word forms to share the same feature value

- Agreement is imposed in certain structural contexts, e.g. in German

  - noun phrases: determiner, adjective, noun

    features: number, gender, case

    der kluge Hund, des klugen Hunds, dem klugen Hund, den klugen Hund, die klugen Hunde, ...

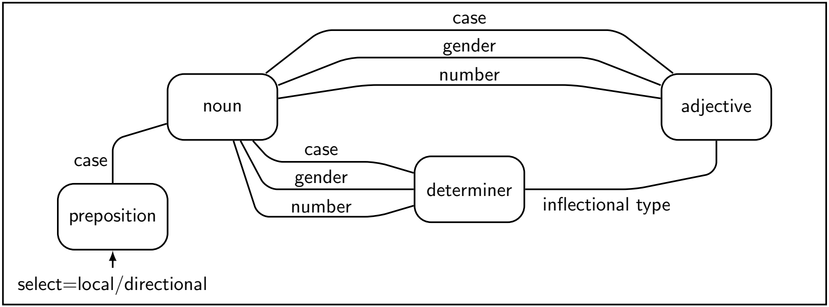
  - clause-level: subject-verb(-reflexive pronoun):

    features: person, number

    Ich freue mich. Du freust dich. Er freut sich. ...

- Checking for agreement is a (simple) constraint satisfaction problem

Agreement


case
gender
number
noun
adjective
case
case
gender
number
determiner
inflectional type
preposition
select=local/directional

Linear Order and Optionality

- Partial order, e.g. German prepositional phrase
  - Examples

    auf das Haus
    auf das kleine Haus
    auf das ziemlich kleine Haus
    aufs Haus

  - Constraints

    Preposition $<$ Determiner
    Contracted Preposition $<$ Graduating Particle
    Determiner $<$ Graduating Particle
    Graduating Particle $<$ Adjective
    Adjective $<$ Noun

Linear Ordering and Optionality

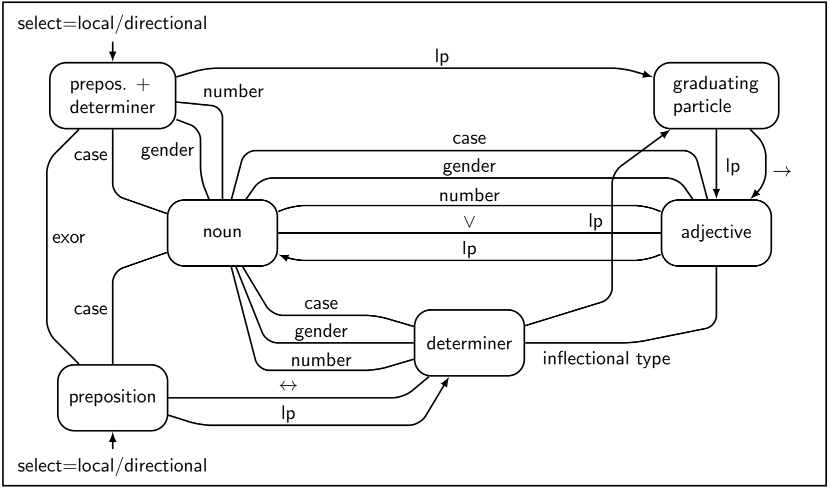
- Co-occurence constraints, e.g. German prepositional phrase
  - Examples

    auf das Haus
    auf das kleine Haus
    auf das ziemlich kleine Haus
    aufs Haus

  - Constraints

preposition $\leftrightarrow$ determiner	*auf Tisch
$\neg$ (contracted_preposition $\leftrightarrow$ preposition)	*in im Bett
graduating_particle $\rightarrow$ adjective	*das sehr Auto
adjective $\vee$ noun	*wegen der

German Prepositional Phrase

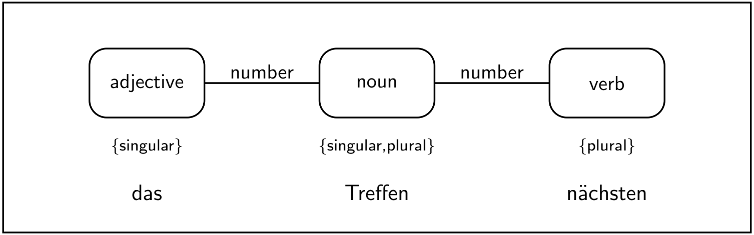


Diagnosis as Constraint Propagation

- Constraint Satisfaction fails in case of ill-formed input
- By *retracting* constraints the global consequences of error hypotheses can be investigated
- Searching for *minimal* error hypotheses ...
- ... by successively increasing the number of retracted constraints

Diagnosis as Constraint Propagation

- Highly precise error explanations can be derived
- Different *views* on the error are supported
  - rule violations
  - missing lexical knowledge
- alternative error interpretations can be found



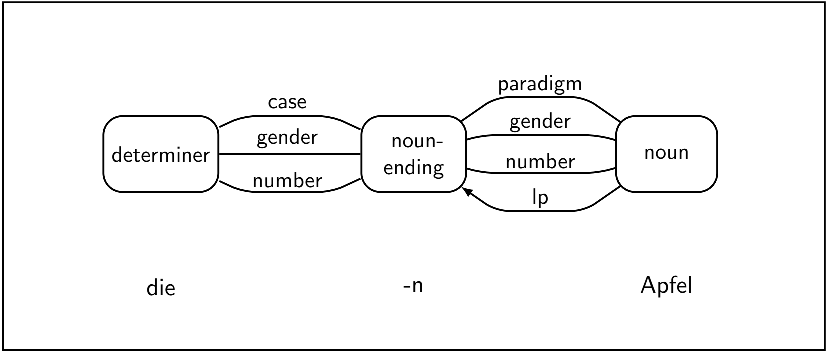
selecting an appropriate one according to the communicative context

Diagnosis as Constraint Propagation

- different feedback levels can be supported
  - error detection
  - error localization
  - error explanation
  - correction proposal

Diagnosis as Constraint Propagation

- Diagnosis of word formation errors



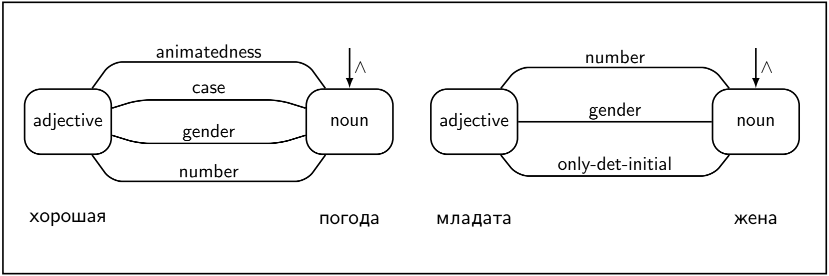
Diagnosis as Constraint Propagation

- error explanations for non-words become available

die Schachtel	der Apfel	
die Schachteln	*die Apfeln	$\rightarrow$ Apfel is masculine not feminine
mit den Schachteln	*mit den Apfeln	$\rightarrow$ the plural of Apfel requires umlaut

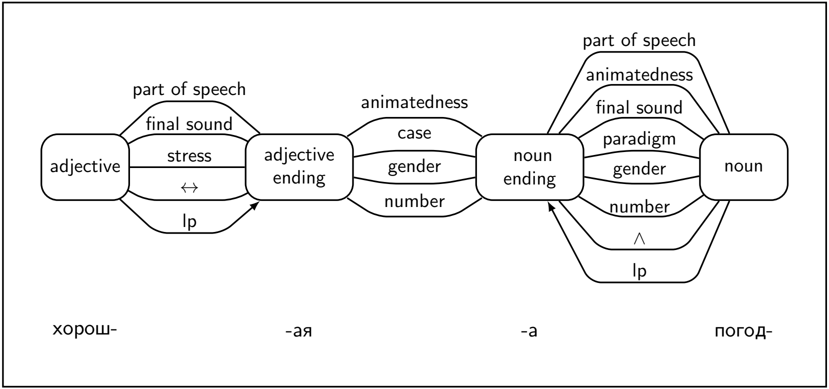
Diagnosis as Constraint Propagation

- Diagnosis in morphologically rich languages with full forms



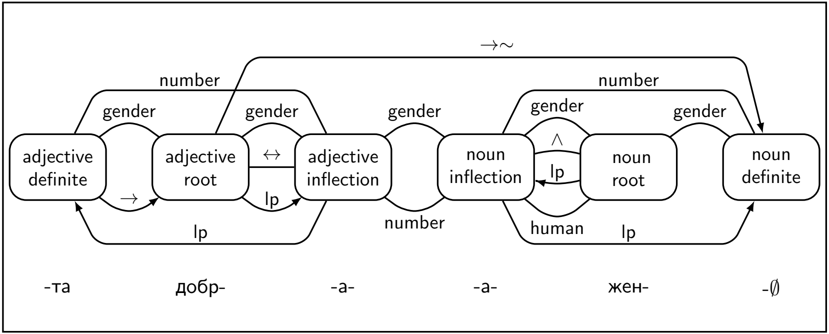
Diagnosis as Constraint Propagation

Morph-based diagnosis in Russian



Diagnosis as Constraint Propagation

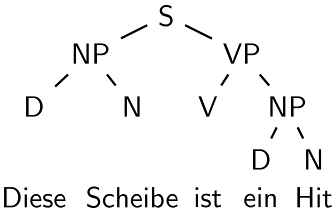
Morph-based diagnosis in Bulgarian

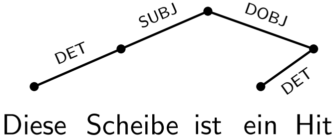


Structural Interpretation

- Parsing a natural language utterance means solving two tasks
  - finding structural descriptions
  - selecting the most plausible one
- Heuristic search in a large search space
- two different kinds of structural descriptions

phrase structure trees vs. dependency trees





Parsing as Constraint Satisfaction

- Labeled word-to-word are dependencies licensed by constraints
- Word forms correspond to the variables of a constraint satisfaction problem:
  - find the "correct" lexical reading
  - find the "correct" attachment point
  - find the "correct" label
- Parsing as structural disambiguation:
  find a variable assignment which satisfies all constraints

Hypothesis space

root/nil	root/nil	root/nil	root/nil	root/nil
det/2	det/1	det/1	det/1	det/1
det/3	det/3	det/2	det/2	det/2
det/4	det/4	det/4	det/3	det/3
det/5	det/5	det/5	det/5	det/4
subj/2	subj/1	subj/1	subj/1	subj/1
subj/3	subj/3	subj/2	subj/2	subj/2
subj/4	subj/4	subj/4	subj/3	subj/3
subj/5	subj/5	subj/5	subj/5	subj/4
dobj/2	dobj/1	dobj/1	dobj/1	dobj/1
dobj/3	dobj/3	dobj/2	dobj/2	dobj/2
dobj/4	dobj/4	dobj/4	dobj/3	dobj/3
dobj/5	dobj/5	dobj/5	dobj/5	dobj/4
Diese	*Scheibe*	*ist*	*ein*	*Hit*
1	2	3	4	5

Parsing as Constraint Satisfaction

- Constraints license meaningful linguistic structures
- Natural language regularities do not depend on word positions
  $\rightarrow$ Constraints have to hold between arbitrary variables

$\{X\}$: DetNom : Det : 0.0 :
    $X{\downarrow}cat{=}det \rightarrow X{\uparrow}cat{=}noun \wedge X.label{=}\mathrm{DET}$
$\{X\}$: SubjObj : Verb : 0.0 :
    $X{\downarrow}cat{=}noun$
    $\rightarrow X{\uparrow}cat{=}vfin \wedge X.label{=}\mathrm{SUBJ} \vee X.label{=}\mathrm{DOBJ}$
$\{X\}$: Root : Verb : 0.0 :
    $X{\downarrow}cat{=}vfin \rightarrow X{\uparrow}cat{=}nil$
$\{X,Y\}$: Unique : General : 0.0 :
    $X{\uparrow}id{=}Y{\uparrow}id \rightarrow X.label{\neq}Y.label$
$\{X,Y\}$: SubjAgr : Subj : 0.0 :
    $X.label{=}\mathrm{SUBJ} \wedge Y.label{=}\mathrm{DET} \wedge X{\downarrow}id{=}Y{\uparrow}id$
    $\rightarrow Y{\uparrow}case{=}Y{\downarrow}case{=}nom$

Preferences

- Natural language grammar is not fully consistent
- Many conflicting requirements
  - e.g. minimizing distance: verb bracket vs. reference

Sie hört sich die Scheibe, die ein Hit ist, an.

Preferences

- Natural language grammar is not fully consistent
- Many conflicting requirements
  - e.g. minimizing distance: verb bracket vs. reference

Sie hört sich die Scheibe, die ein Hit ist, an.

Sie hört sich die Scheibe an, die ein Hit ist.

Conflicts

Conflicts occur

- between levels of conceptualization
  e.g. syntax, information structure and semantics
- between different processing components
  e.g. tagger, chunker, PP-attacher
- between the model and the utterance
  e.g. modelling errors, not well-formed input
- between the utterance and the background knowledge
  e.g. misconceptions, lies
- across modalities
  e.g. seeing vs. hearing

Goal: achieve robustness and develop diagnostic capabilities

Conflicts

Why should we care about conflicts?

- they are pervasive
- they provide valuable information
  - for improving the system:
    e.g. through manual grammar development or reinforcement learning
  - about the proficiency of the speaker/writer:
    e.g. to derive remedial feedback
  - about the intentions of the speaker/writer:
    e.g. attention focussing by means of topicalization
  - for guiding the parser

Weighted Constraints

- conflict resolution requires weighted constraints
  - weights describe the importance of the constraint
  - how serious it is to violate the constraint
- differently strong constraints
  - hard constraints, must always be satisfied
  - strong constraints: agreement, word order, ...
  - weak constraints: preferences, defaults, ...

Weighted Constraints

- weighted constraints are defeasible
- preferential reasoning can be applied
  - global optimization problem
  - based on local scores
  - scores are derived from constraint violations (penalties)

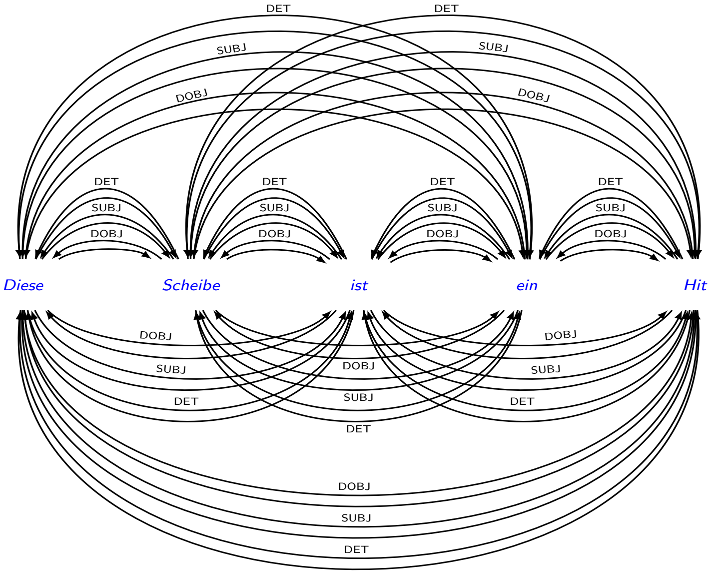
Global Constraints

- Most constraints are local ones (unary, binary)
- Sometimes global requirements need to be checked
  - existence/non-existence requirements (e.g. valencies)
  - conditions in a complex verb group
- Local search supports the application of global constraints
  - always a complete value assignment (i.e. a dependency tree) is available
- Three kinds of global constraints
  - *has*: downwards tree traversal
  - *is*: upwards path traversal
  - recursive constraints: can call other constraints to be checked elsewhere in the tree

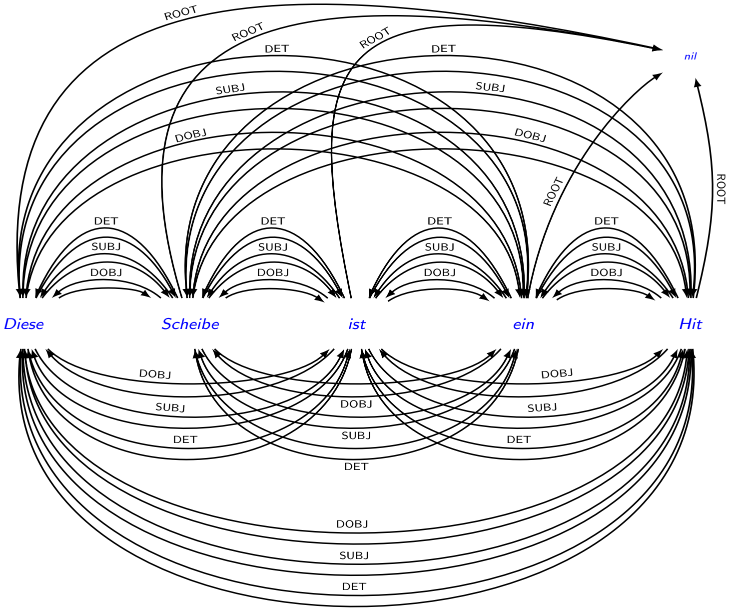
Weighted Constraints

- Different solution procedures are available
  - pruning
  - systematic search
  - local search, guided local search (transformation-based)
- strong quality requirements
  - a single prespecified solution has to be found (gold standard)
  - sometimes the gold standard differs from the optimal solution
  - modelling errors vs. search errors
- The best method found so far:
  - local search with value exchange (frobbing)
  - gradient descent heuristics
  - with a tabu list
  - with limits (similar to branch and bound)
  - increasingly accepting degrading value selections to escape from local minima

Frobbing

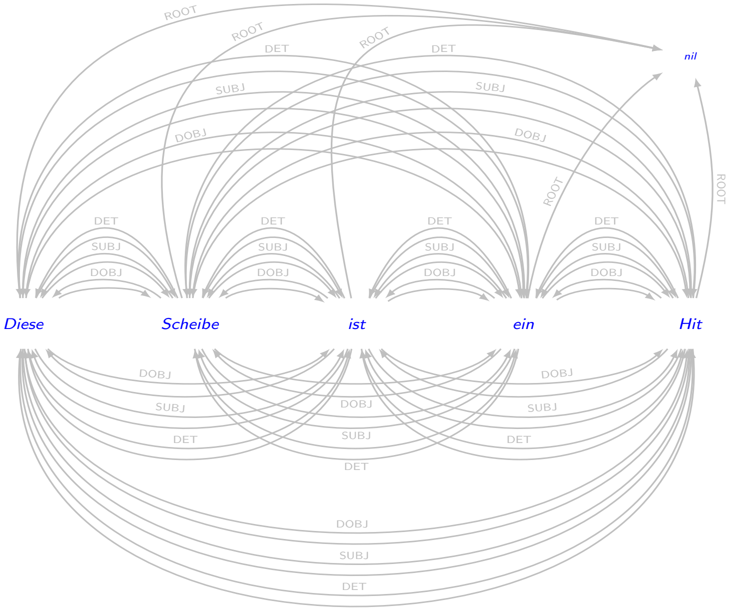

DET
DET
SUBJ
SUBJ
DOBJ
DOBJ
DET
DET
DET
DET
SUBJ
SUBJ
SUBJ
SUBJ
DOBJ
DOBJ
DOBJ
DOBJ
Diese
Scheibe
ist
ein
Hit
DOBJ
DOBJ
SUBJ
DOBJ
SUBJ
DET
SUBJ
DET
DET
DOBJ
SUBJ
DET

Frobbing



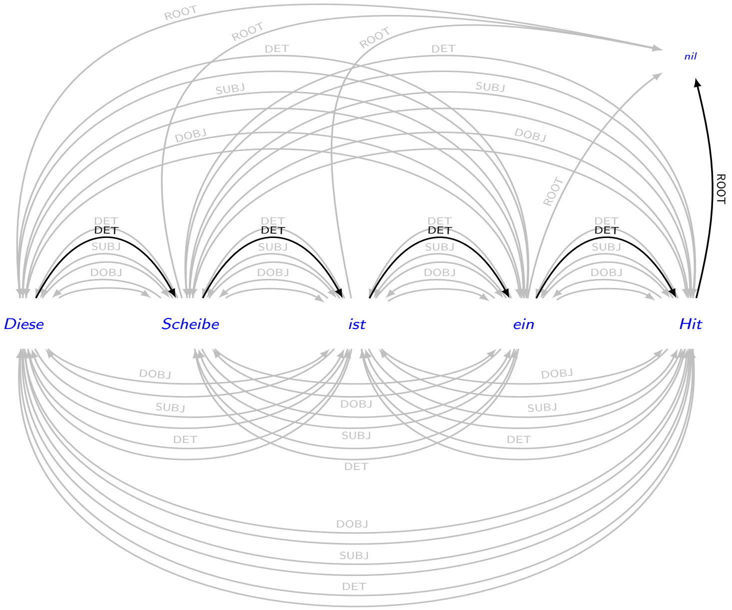
ROOT
ROOT
ROOT
nil
DET
DET
SUBJ
SUBJ
DOBJ
DOBJ
ROOT
ROOT
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
Diese
Scheibe
ist
ein
Hit
DOBJ
DOBJ
SUBJ
DOBJ
SUBJ
DET
SUBJ
DET
DET
DET
DOBJ
SUBJ
DET

Frobbing

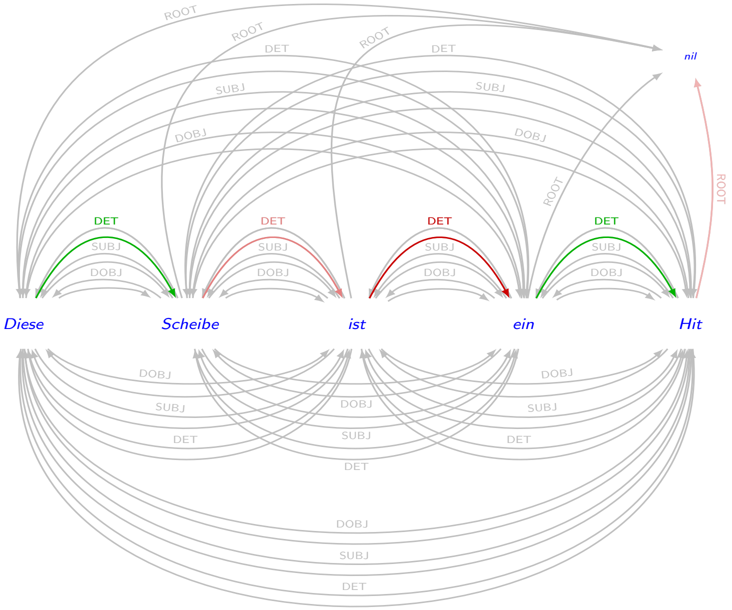


nil
ROOT
ROOT
ROOT
ROOT
ROOT
DET
DET
SUBJ
SUBJ
DOBJ
DOBJ
DET
SUBJ
DOBJ
Diese
Scheibe
ist
ein
Hit
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET

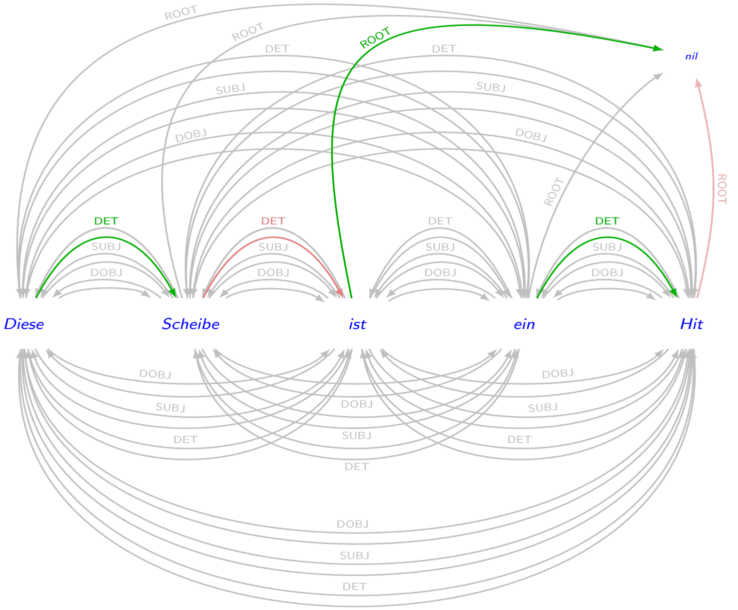
Frobbing



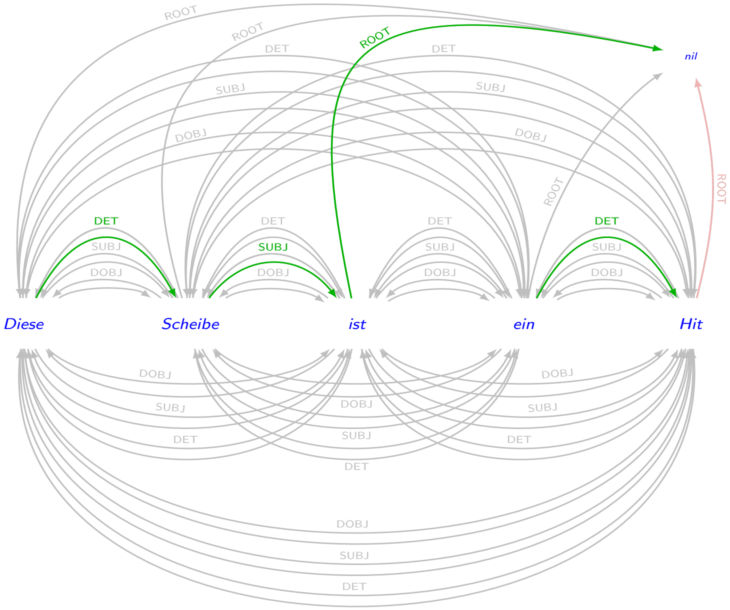
Frobbing


nil
ROOT
ROOT
ROOT
ROOT
ROOT
DET
DET
SUBJ
SUBJ
DOBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
Diese
Scheibe
ist
ein
Hit
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET

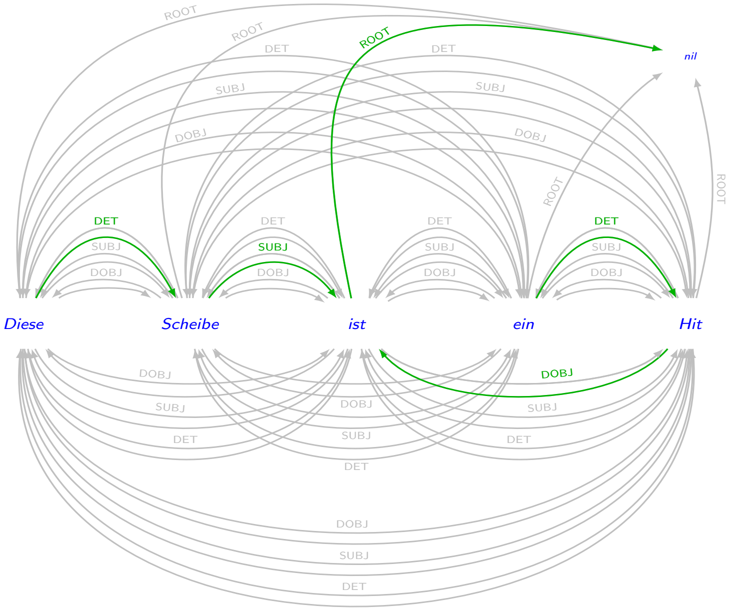
Frobbing


ROOT
ROOT
ROOT
nil
DET
DET
SUBJ
SUBJ
DOBJ
DOBJ
ROOT
ROOT
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
Diese
Scheibe
ist
ein
Hit
DOBJ
DOBJ
SUBJ
SUBJ
DOBJ
DET
SUBJ
DET
DET
DET
DOBJ
SUBJ
DET

Frobbing



Frobbing



ROOT
ROOT
ROOT
DET
DET
SUBJ
SUBJ
DOBJ
DOBJ
ROOT
ROOT
nil
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
DET
SUBJ
DOBJ
Diese
Scheibe
ist
ein
Hit
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET
DOBJ
SUBJ
DET

Non-local Transformations

- usually local transformations result in inacceptable structures
- sequences of repair steps have to be considered.
- e.g. swapping SUBJ and DOBJ

a)	syntax	...
diese_1	det/2	...
scheibe_2	**dobj/3**	...
ist_3	root/nil	...
ein_4	den/5	...
hit_5	**subj/3**	...

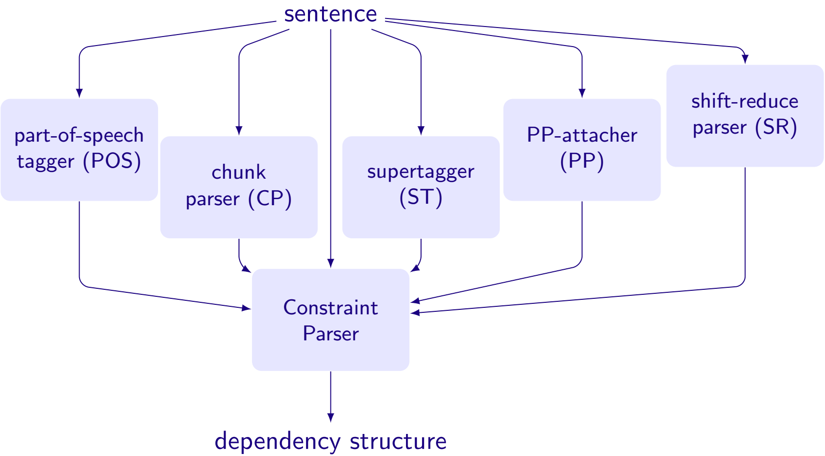
$\Longrightarrow$

b)	syntax	...
diese_1	det/2	...
scheibe_2	**subj/3**	...
ist_3	root/nil	...
ein_4	det/5	...
hit_5	**dobj/5**	...

Hybrid parsing

- The bare constraint-based parser itself is weak
- But: constraints turned out to provide an ideal interface to external predictor components
- predictors might be inherently unreliable
  $\rightarrow$ can their information still be useful?
- using several predictors $\rightarrow$ consistency cannot be expected

Hybrid parsing



Hybrid parsing

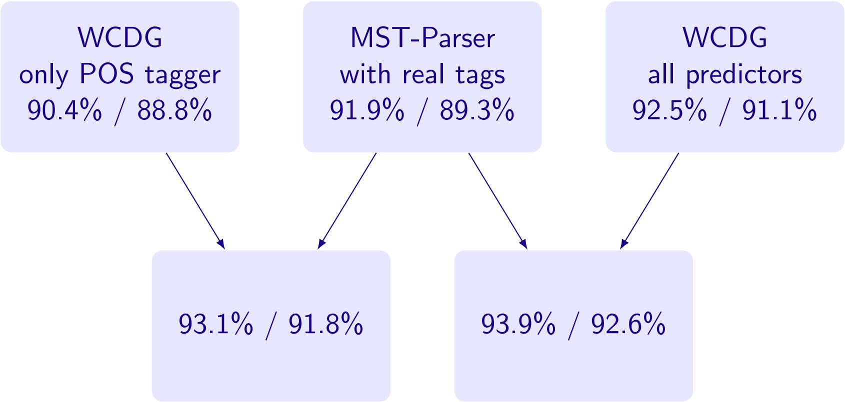
- results on a 1000 sentence newspaper testset (Foth 2006)

	accuracy	
Predictors	unlabelled	labelled
0: none	72.6%	68.3%
1: POS only	89.7%	87.9%
2: POS+CP	90.2%	88.4%
3: POS+PP	90.9%	89.1%
4: POS+ST	92.1%	90.7%
5: POS+SR	91.4%	90.0%
6: POS+PP+SR	91.6%	90.2%
7: POS+ST+SR	92.3%	90.9%
8: POS+ST+PP	92.1%	90.7%
9: all five	92.5%	91.1%

- net gain although the individual components are unreliable

Hybrid Parsing

- What happens if the predictor becomes superior? (KHMYLKO 2007)



Parsing as Constraint Satisfaction

Current research

- Incremental parsing
  - Language unfolds over time
  - Decisions about the optimal interpretation have to be taken in a timely manner
  - Local search has an ideal anytime behaviour: fully interruptable
- Parsing in a multimodal environment
  - Mapping visual stimuli onto linguistic constructions
  - Using language to guide the visual attention
- Using dynamic predictions
  - The world changes over time as the utterance unfolds
  - How does the behaviour of the parser depends on when an external information becomes available

Summary

Constraint satisfaction techniques ...

- simplify search problems
- provide diagnostic information
- can contribute attractive anytime properties

Weighted constraint satisfaction ...

- helps to solve hard optimization problems
- deals with conflicting regularities
- facilitates information fusion in hybrid architectures
- maintains the diagnostic abilities

Major challenge:
The search problem has to be recast in terms of a set of variables and their compatible value assignments.