Beam search strategy for IPS: Integration of a structure-based beam search

Cathy Berthouzoz and Paola Merlo
Département de linguistique - LATL
Université de Genève

1 Introduction

Differently from programming languages, natural languages are nondeterministic, since they are ambiguous. When dealing with natural language processing one of the main tasks is to resolve ambiguity at the lexical, syntactic, semantic and pragmatic levels in order to produce naturally preferred analyses, *i.e.* preferred interpretations, of any given input.

A provably complete approach to the solution of this problem consists in pursuing all possible analyses in parallel, thus simulating nondeterminism. Full parallelism – all the alternatives are processed from any point of ambiguity – is computationally expensive and unsupported by current findings in human sentence processing. To avoid the combinatorial explosion of alternatives that full parallelism would generate, limited non-deterministic parsing algorithms have been elaborated (see for review Dowty, Karttunen, and Zwicky 1985; Reyle and Rohrer 1988): they limit the number of hypotheses that are pursued at any point of ambiguity.

Beside the inherent ambiguity of the language, the form of the underlying grammar may add what is called *parasitic ambiguity*, ambiguity resulting from the underspecification of the grammar. For instance, the IPS parser developed at LATL (Wehrli 1992; Wehrli 1993 among others) used as testbed for our project, employs a *GB-style* grammar. Since this kind of grammar is underconstrained, it generates a large amount of parasitic hypotheses, namely it generates many types of temporary ambiguities even for unambiguous sentences. Using an efficient pruning technique to discard a certain number of alternatives is thus very important.

The IPS system integrates a filtering device to prune the generated hypotheses in the course of parsing, and a ranking device to score the complete analyses of a given sentence and retrieve the most preferred one at the end of the parse. The filtering device on the one hand prunes hypotheses considered as less preferable according to structure-based heuristics. These heuristics are based on the attachment type of the given constituent. As described elsewhere (Berthouzoz and Merlo 1996), they can be more or less efficient, depending on the point in the analysis from which they are invoked. At the end of the parse, other heuristics are used to prune complete analyses whose type mismatches the default type deduced from the final punctuation mark[1].The ranking device on the other hand ranks the complete analyses according to their cost, computed as described in section 2.1 below.

We describe in this paper the integration of a structure-based beam search algorithm into the IPS parser. The beam is used both as filtering and ranking device. The same strategy and the same data structure will serve for both tasks. This structure-based algorithm is the first

[1]An interrogative sentence is supposed to be ended with an interrogative mark, an exclamative sentence with an exclamation mark and an assertion with a colon respectively.

step towards the development of a probabilistic beam search algorithm that will be tested against this one.

2 The beam search

The recovery of the correct analysis for a sentence can be described as the retrieval of the correct or preferred path in a space of hypotheses generated by the grammar, represented as a graph.

Let a tree-representation of an algorithmic search space be given. In the basic **beam search** algorithm used in AI (Winston 1981) space, only a predefined number w – the beam width – of best nodes are kept for further processing at each level of the search tree. This means that for a tree of degree b, a maximum of wb nodes only are searched at each level. The metric to establish which nodes are the best is a measure of the distance of the node to the goal.

This tree-representation can be applied to the search space of a parallel parsing algorithm. Take for instance the sentence given in (1) below, whose lexical ambiguity is very high since almost every word can be either noun or verb. Suppose we are processing the word *hope* while (1b) is the left context. The parser tries to combine the current word with its context. *hope* can be a noun – in which case the parser constructs the complex noun phrase (1c) – or a verb – in which case it constructs the sentence (1d). A parallel parser constructs both alternatives. At the following step, when the parser processes the word *tourists*, the left context contains the structures given in (1c) and (1d), which can both combine with the new word. The same process goes on until the end of the sentence is reached.

The search space of the parser can thus be seen as a tree of undetermined degree. The structures hypothesized by the parser at each step of the parsing process constitute the nodes of such a tree. The number of daughters of each node depends on the degree of ambiguity — lexical ambiguity or ambiguity of attachment to its context — the next word presents. The leaves formed by complete structures — they include all the words of the input sentence — indicate successful analyses. A path going from the root to the leaves encodes the different steps performed by the parser to generate the complete analysis.

(1)a. The league's promoters hope tourists will join fans like Paul.

b. [$_{\text{DP}}$ the league's promoters]

c. [$_{\text{DP}}$ the league's promoters hope]

d. [$_{\text{CP}}$ [$_{\text{TP}}$ [$_{\text{DP}}$ the league's promoters] [$_{\text{T'}}$ hope]]]

2.1 The beam metric

The beam search algorithm is used in natural language processing to prune dispreferred analyses. The metric to determine the width of the beam can be symbolic. For instance, Gibson (1991) measures what he called processing load units (PLUs). PLUs are abstract units measuring the cost of predetermined properties of the linguistic representations of an input sentence. Gibson defines the following properties: Thematic Reception, Lexical Requirement, Thematic Transmission and Recency Preference. As processing resources are limited, there

is a maximum load that the human processor can tolerate at any state of the parse. The difference in load between competing structures determines the width of the beam.

The metric can also be based on a probability model. Collins (1996) defines a probabilistic parser based on bigram dependencies, which uses a beam search to prune very low probability analyses. The width of the beam is a fixed threshold below the best analysis, defined in terms of probabiltiy of the derivation up to that point in parsing.

The beam search can be applied several times to the same input, using differnt sources of information to determine the parse and the beam. processing text requires the use of several knowledge sources – lexical information, syntax, semantics, pragmatics, which can be symbolic or probabilistic – while speech processing needs specific knowledge such as acoustic model, language model, prosody. The hypotheses generated by the NLP system must be consistent with all the knowledge sources. Computing the preferred hypothesis can be therefore very time-consuming. For instance, in order to parse the noun phrase *the state of being away or of not being present*, Briscoe and Carroll (1993) report times of 7.9, 6.0 and 5.8 CPU seconds on a DEC 3100 Unix workstation for a bottom-up chart parser, a semi-automatic LR parser and a nondeterministic LR parser respectively. The IPS system parses the very lexically ambiguous sentence *The league's promoters hope tourists will join fans like Paul.* in 2.6 CPU seconds on a DEC 3000/M800 AXP workstation. In parsing, the search space for a sentence of 30 words may contain many hundreds, if not thousands of paths. In order to tackle this problem, the N-best algorithm was defined to find the N best parsing hypotheses, using a measure of distance from the most likely sentence to determine the width of the beam (Chow and Swartz 1989). The output of the parser consists in a list of ranked sentences, where the few top sentences contain the correct or preferred one. The input is processed in two passes: during the first pass the most efficient, but cheapest knowledge sources are used to select the N best hypotheses, which are then re-scored with the remaining knowledge sources in a second pass. For instance, the first set of knowledge sources can include a statistical grammar while the second includes semantic and pragmatic components.

The search space of the IPS system – an incremental bottom-up parser – is represented by a chart (Kay 1967; Kaplan 1973). A chart is an acyclic directed weighted graph whose vertices are the positions between the words of the input sentence and the edges are the well-formed constituents hypothesized by the parser. A complete analysis of the input sentence is an edge from the first to the last vertex.

Edges are weighted by a *structural cost*. Before being stored into the chart, each new constituent resulting from the attachment of the current constituent to its context receives a cost computed by the parser on the basis of several factors. One factor is the type and site of attachment, as defined in tables 1 and 2 below, which implement the attachment hierarchy and the locality constraints developed in Walther (1993). Other factors are the cost of the current constituent and the cost of the site of attachment.

Attachment	Cost
functional complement	0
noun modifier	5
subcategorized complement	10
subcategorized complement of predicate	15
adjunct	20
trace of subcategorized complement	10
trace of subcategorized adverb	20
trace of specifier	15
trace of adjunct	20

Table 1: Cost attributed to the attachment types

Attachment	Cost
the first PP in the domain of a PP	1
subsequent PP in the domain of a PP	20
a PP to a verb	10
a PP to a verb, after a predicative complement	30
the first PP not to a verb and not in the domain of a PP	20
subsequent PP not to a verb and not in the domain of a PP	30
subcategorized complement or adjunct not to a verb	10
subcategorized complement or adjunct after a CP	+30

Table 2: Cost attributed to the locality of the attachment

At the end of the parse, the system ranks the completed hypotheses by adding to the structural cost described above the cost of the grammar violations, if any, as defined in 3. It then retrieves the best one, *i.e.* the one with the minimum cost.

The structural cost will serve as the metric for the beam search algorithm developed here: at any step of the parse, only the constituents with a cost lying within a given bound from the lowest cost will be kept for further processing.

2.2 The beam width

The beam width can be set as a number of hypotheses – at each level, the N constituents with the lowest costs are kept – or as a threshold – at each level, all the constituents with a cost between $cost_{min}$ and $cost_{min} + N$ are kept.

When dealing with natural language, and in particular if one uses an overgenerating grammar such as a GB-style grammar, it would be too constraining to set a maximum number of hypotheses to be kept at each level. If the number is too low, some sentences would not be correctly analyzed if they involve complex structures or highly ambiguous words. On the other hand, setting the number too high would generate too many parasitic hypotheses for unambiguous or simpler sentences.

We thus adopt the strategy of setting the beam width as a threshold. For the algorithm to work efficiently the right value of the threshold has to be determined. This value will be set empirically, as described in section 3 below.

Recoverable violations	Actual Cost
incomplete DP	20
thetaless subject	30
subjectless TP	30
caseless subject	40
complementless to	50
incomplete PP	60
incomplete passive	100
Non recoverable violations	
no determiner	10
sentential subject	10
no lexical match	15
heavy NP-shift	20
wrong punctuation	30
WH in situ	30
Fatal violation	
open abar chain	100

Table 3: Weight of the recoverable, non recoverable and fatal grammar violations.

2.3 The beam data structures

As mentioned above, the search space is represented by a chart (Kay 1967; Kaplan 1973) in the IPS system . The parser stores the well-formed constituents it generates in this structure. We use a chart because it is efficient, as it represents parallel structures compactly. The parser attempts to keep a connected graph of constituents whenever possible. So, for instance, for every new word, it generates adequate constituents and attempts to attach them to the left context. In many cases of left branching, however, no attachement is possible. For instance, cases of discontinuous structures are found when the system processes the example (2). When *John* is encountered, the parser cannot attach it to the left context, the conjunction *while*. The same occurs when *Mary* is encountered, as the parser cannot attach it as complement of the verb *sleeping*. Only when the verb *was* is processed, can the parser attach the new constituent – a clause whose subject is *John* – as complement of the conjunction *while* to form a subordinate clause. And only when it processes the verb *read*, can the processor attach the subordinate clause as modifier of the new constituent – a clause whose subject is *Mary*.[2]

(2) While John was sleeping Mary read a book.

In general, before attaching phrasal complements or modifiers of full sentences, the first constituent – the subjects of the subordinate and of the main sentence in the example above– cannot be attached to their left context until the verb is recognized, as their mother node has not been encountered yet.

The beam seach algorithm looks for a path into a tree or a lattice, which are continuous data structures. A chart, on the contrary, can contain discontinuous structures, thus it does not always implement the search space that the beam search algorithm requires. We observe

[2] Attachment of modifier and subject is called specifier attachment and is done on the left of a structure while attachment of a complement is done on the right of a structure.

however, that in IPS we do not use the beam search for finding a path in the chart at the end of the parse, but for pruning bad constituents during the parse and for ranking complete analyses that are retrieved from the chart. Thus we resolve the apparent contradiction by defining a new data structure, which the beam search algorithm manipulates.

In order to limit overlap with the chart, while guaranteeing completeness, we choose a doubly-linked list, whose nodes are 4-tuples *(const, cost, level, keep)* described below.

- const: the constituent, in fact, a pointer to the constituent, resulting from the combination of the current word with its left context;
- cost: the cost of the constituent, computed as described in section 2.1 above;
- level: the beginning vertex of the constituent in the chart;
- keep: a boolean value used for pruning: if true the constituent is kept for further processing, if false, the constituent is pruned.

The minimal set of operations defined on the proposed data structure is as follows.

- `Insert(current,const,cost,level)` Inserts in order `const` with `cost` in the beam pointed at by `current` at `level`.
- `GetMin(current)` Returns the minimum cost of the current level in the beam pointed at by `current`.
- `Retrieve(current,threshold,complete,best,worst)` Looks into the beam pointed at by `current` and returns two lists.
  1. `best` : the list of constituents that have a cost below `threshold`. The field `keep` of these constituents is set to true.
  2. `worst` : the list of constituents that have a cost above `threshold`. The field `keep` of these constituents is set to false.

  If `complete` is true, only the constituents which start at vertex 1 of the chart are kept. This flag is used at the end of the parse to rank complete analyses of the given sentence.

2.4 Beam search algorithm

Different strategies of selection based on heuristics have been described in Berthouzoz and Merlo (1996). To summarize, the selection procedure can take place at different points, *i.e.* 1) after each word sense, the selected constituents are inserted into the chart, 2) after each word, the discarded constituents are deleted from the chart and 3) after both. The best results appear to occur when selection is called after each word. Therefore, the beam search is invoked as a filter on the generated hypotheses of each new word and the discarded constituents are deleted from the chart. This strategy appear to be good for computational reasons – we avoid too many manipulations of heavy data structures – and for practical reasons – the beam search is not run twice on the same set of hypotheses, as in case 3) above, nor is it run by chunks of hypotheses at each level, as in case 1).

The same algorithm is used for ranking the complete analyses. Therefore, it will also be invoked after the whole sentence is processed. Informally, the the beam search algorithm proceeds as follow:

1. *Set the level to 1;*
2. *After each word,*
   (a) *Insert each constituent resulting from the combination of the current word with its context (in increasing order of cost) into the beam data structure;*
   (b) *Calculate the minimum cost of that level.*
   The level is the group of constituents generated by combination of the current word to its context. It is assigned the same number as the beginning vertex of the current word in the chart;
   (c) *Retrieve the constituents which are within the threshold of that level;*
   the threshold is set by adding the beam width to the minimum cost of that level;
   (d) *Remove from the chart the constituents whose cost is higher than the threshold.*
3. *At the end of the parse,*
   (a) *Calculate the minimum cost of the last level;*
   (b) *Retrieve the complete analyses (starting at vertex 1);*
   (c) *Compute the final cost of the complete analyses.*
   Namely add the cost of the grammar violations (if any) to the structural cost.
   (d) *add 1 to the current level;*
   The same algorithm and the same data structure are used to rank the complete analyses. As for completing the analysis we need N+1 levels (number of vertices of the chart), we need an additional level for ranking.
   (e) *Insert (in increasing order of cost) the constituents with the final cost at the current level;*
   (f) *Retrieve the complete analyses (if any) which are within the threshold of that level.*
   The first analysis is the preferred one as it has the minimum cost.

Let's reformulate the algorithm in more formal terms. First, let's define the variables and auxiliary data structures used.

- `keep`: a boolean value used for pruning: if true, the constituent is kept for further processing, if false, it is deleted from the chart.
- `complete`: a boolean value used for discriminating between pruning (complete is false) and ranking (complete is true);
- `width`: an integer value specifying the beam width;
- `vertex`: an integer specifying the current vertex in the chart; that is the ending vertex of the last word processed;
- `level`: an integer specifying the current level in the parse;
- `cost`: an integer specifying the structural cost of a constituent;
- `minCost`: an integer specifying the minimum cost at a given level;

- `constituent`: a pointer to a data structure, which encodes the structural, syntactic and semantic properties of a word or a combination of words;
- `candidateList`: a list of all the constituents generated by attachment of the last word to its left context;
- `bestList`: a ranked list of the constituents whose cost is below a given threshold;
- `worstList`: a list of the constituents whose cost is over a given threshold;
- `completeList`: a list of all the complete analyses;
- `rankedList`: a ranked list of the best complete analyses;
- `beamPtr`: a pointer to a doubly-linked list whose nodes are the 4-tuples (constituent, cost, level, keep).

The algorithm is then defined in the following terms:

- `level` ← `1;`
- **For** each word processed **do**
  1. Retrieve from the chart the list of all the constituents whose ending vertex is `vertex` and store them in `candidateList`
  2. **For** each `constituent` in `candidateList` **do**
     - Compute cost of constituent and store it in `cost`
     - `Insert(beamPtr,constituent,cost,level);`
  3. `minCost` ← `GetMin(level);`
  4. `complete` ← `false;`
  5. `Retrieve(level,minCost + width,complete,bestList,worstList);`
  6. **For** each constituent in `worstList` **do**
     - Remove `constituent` from the chart;
  7. `level` ← `level + 1;`
- After all words are processed
  1. `tempWidth` ← `MAX(INTEGER);`[3]
  2. `minCost` ← `GetMin(level);`
  3. `complete` ← `true;`
  4. `Retrieve(level - 1,minCost + tempWidth,complete,bestList, worstList);`
  5. **For** each constituent in `bestList` **do**
     - Compute the cost of the violations and add it to the structural cost
     - `Insert(beamPtr,constituent,cost,level);`
  6. `minCost` ← `GetMin(level);`
  7. `complete` ← `true;`

[3]MAX(INTEGER) represents the maximum value an integer number can take. Specifying a cost of this value allows us to retrieve all the analyses, whatever their cost is.

8. `Retrieve(level,minCost + width, complete, best,worst);`
9. `level` $\leftarrow$ `1;`

At that point, `best` contains the most probable analyses in decreasing order of cost, which means in increasing order of preference.

3 Setting the beam width: experiments and results

The beam width has to be set carefully in order for the system to run satisfactorily. As we could not establish in advance what the best value was, we performed some experiments, which are reported below.

The system – with different beam widths – was run over the test sentences given in appendix A.[4] Its performance was evaluated with the two standard measures of precision and recall, measures of accuracy and completeness respectively. In our system, *precision* measures the number of analyses that the parser reports as correct which are actually correct according to manual checking, while *recall* measures the number of correct analyses that the parser actually finds. The number of expected correct analyses was also determined independently by the authors.

These measures are defined as follow:

$$precision = \frac{\text{number of checked analyses}}{\text{number of correct analyses}}$$

$$recall = \frac{\text{number of checked analyses}}{\text{number of expected analyses}}$$

In the formulae above, the *correct* analyses are the analyses that the parser generates as free of any grammar violation, the *checked* analyses are the analyses produced as correct by the parser that we also find correct, and the *expected* analyses are the correct analyses that the parser should produce (1 for unambiguous sentences, more than 1 for ambiguous ones).

For the standard system, which produces one analysis for each sentence – irrespective of the degree of ambiguity the input presents – precision and recall are accidentally equivalent, because the number of analyses returned as correct by the parser is equal to the number of expected analyses (we have two incorrect sentences, number 33 and 35, but the parser did not produce an analysis for sentences number 37 and 53):

$$precision = \frac{53}{59} = 0.898$$

$$recall = \frac{53}{59} = 0.898$$

Although these results look quite good, these figures do not provide an accurate picture of the amount of work that the parser performs to reach this solution. As the main goal

[4]The description and discussion of the corpus can be found in Berthouzoz and Merlo (1996).

of adopting a beam search method is to prune partial analyses that have only a very low probability of being part of a successful final analysis, we have undertaken more accurate experimentation, varying the width of the beam. We aim for a similar performance to what we just reported while decreasing the computation in the course of the analysis.

3.1 Evaluation of the parser without filtering

The first step in the evaluation was to run a modified version of the IPS system which retrieves all the correct analyses the parser generates, instead of retrieving only most highly ranked one.

We also show the number of complete analyses the parser produced for every sentence. The complete analyses differ from the correct analyses in the sense that they can include grammar violations (*cf* Appendix C.2) –whatever the grade of the violation(s) is – while the correct ones are free of any violation.

An example of one of the outputs is given in example (3) . We can see that the parser has produced two correct analyses, but it has generated five more complete analyses which present some grammar violations:

(3)
```
He saw a man with a telescope.
number of constituents :  75
number of complete analyses :  7
number of correct analyses :  2
[CP[TP[DP he ][T' saw [DP a [NP man [PP with [DP a
[NP telescope ]]]]]]]]
score :  40
[CP[TP[DP he ][T' saw [DP a [NP man ]][AdvP[PP with [DP a
[NP telescope ]]]]]]]
score :  40
total CPU time :  190 milliseconds
```

As we are only interested in correct analyses, we do not show all the complete ones. The difference between the complete and the correct hypotheses is sometimes very high, as seen in the example (4) where there are only three correct analyses out of 50 complete ones.

(4)
```
While John was mending the sock fell off his lap.
number of constituents :  362
number of complete analyses :  50
number of correct analyses :  3
[CP[AdvP[CP while [TP[DP John ][T' was [VP mending ]]]]][C'[TP[DP the
[NP sock ]][T' fell [AdvP[PP off [DP[DP his ][D'[NP lap ]]]]]]]]
score :  30
```

```
[CP[AdvP[CP while [TP[DP John ][T' was ]]]][C'[TP[DP[VP mending
[DP the [NP sock ]]]][T' fell [AdvP[PP off [DP[DP his ][D'
[NP lap ]]]]]]]]]
score :  50
[CP while [TP[DP John ][T' was [VP mending [DP the [NP sock [FP[DP ei]
[F'[AP[DP ei][A' fell ]]]]]][AdvP[PP off [DP[DP his ]
[D'[NP lap ]]]]]]]]]
score :  65
total CPU time :  1000 milliseconds
```

The output includes also the number of the generated hypotheses (`number of constituents`), the time needed to complete the parse (`total CPU time`) and the cost of each analysis (`score`). The number of constituents is a good indicator of the computation required and it will be used to adjust the beam width.

The output was then checked to see if the correct analyses produced by the parser were really correct. For instance, for the sentence given in (3) both analyses are correct, but for the one given in (4), only the first one is correct. The number of checked analyses is 2 for example (3) – it corresponds to the number of correct analyses – while the number of checked analyses is 1 for example (4) – it is less than the number of correct analyses.

For each sentence we also computed the number of expected analyses, *i.e.* the number of analyses the parser should produce. An unambiguous sentence has only one expected analysis, like the sentence in (4), while an ambiguous one has more than one analysis, for instance the sentence in (3) has two expected analyses. Table 8 in the appendix B summarizes these outputs for all the test sentences.

Precision and recall for the modified version of the system are:

$$precision = \frac{113}{238} = 0.475$$

$$recall = \frac{113}{126} = 0.897$$

These numbers show that the system without any filtering is not sufficiently precise – a low precision – but it is quite complete – a high recall. This means that in order to analyze correctly almost 90% of the sentences, the parser produces twice as many incorrect analyses.

3.2 Beam applied at end of the parse only

The first hypothesis was that applying a beam at the end of the parse on the complete sentences would provide a first indication to determine the width of the beam filter applied during parsing. For this first experiment, we chose different widths in a range of 0 to 60 on empirical grounds. A width of 0 means that we want only the analyses with the lowest cost. Only one analysis per sentence will pass this filter and it should be the most preferred one. When there is no preference of attachment, all the alternative analyses should have the

same cost, thus increasing the number of correct analyses. This width should increase the precision of the system. A width of 60 is the maximum distance between the analysis with the lowest cost and the one with the highest cost[5]. It should allow the recovery of all the checked analyses, thus having the same recall than the system run without filtering. This width should also increase the precision of the system.

Table 9 of the appendix B shows the results for widths of 0, 5, 10, 30 and 60. Table 4 summarizes the precision and recall for these different values.

Filter	precision	recall
BE 0	0.871	0.587
BE 5	0.779	0.643
BE 10	0.744	0.714
BE 30	0.607	0.857
BE 60	0.536	0.897

Table 4: Precision and recall for different widths of a beam applied at the end of parse

As expected, the precision of the system was increased for every beam width while the recall was lowered, except for the last one. Compared to the values of the system run without filtering, the gain in precision is low for width of 60 (13%). For width of 30, we gain 28% precision for a loss of 4% in completeness only. For width of 10, gain in precision is 57% for a loss of 20% in completeness. Width of 5 increases precision by 64% while it decreases completeness by 28%. And width of 0 allows a gain of precision of 83% for a loss in completeness of 35%. The low recall of the 0-width beam means that there are cases where there is a preference of attachment: the cost of the dispreferred analyses being higher than the preferred one cost, they do not pass the filter, hence lowering the recall.

The left panel of figure 1 shows the different results visually.

3.3 Beam applied during the parse

The next experiment consisted in applying a beam filter during the parse only, to prune the dispreferred constituents hypothesized by the parser. The beam widths were set empirically, varying from 5 to 140. A value of 140 allows the parser to behave exactly as if it were run without filtering. A value of 5 corresponds to the minimal structural cost assigned by the parser. A low value can be so constraining that it prunes constituents with a high cost while they could lead finally to a complete analysis with a lower cost. On the contrary, a high value might not be constraining enough, so that constituents which lead to high-weighted complete analyses are kept uselessly. Table 10 in appendix B shows the results for beam widths of 10, 30, 60 and 100. Table 5 summarizes the precision and recall for these different beam values. As expected, very small recall values for widths of 10 or 30 show that these widths are too constraining to be useful. Compared to the values of the system run without filtering, a width of 100 gains in precision only 16%, while recall degrades slightly. This result shows that a beam of this width would not be efficient. For width of 60, we gain 33% in precision for a loss of 29% in recall. This width seems to be a good compromise. The right panel of figure 1 which shows the different results visually, shows us that we can expect the best value to lie close to 60. This value will thus be kept for further experimentation.

[5]It was determined by manually checking the output of the parser.

Filter	precision	recall
BF 10	0.800	0.032
BF 30	0.543	0.152
BF 60	0.630	0.640
BF 100	0.550	0.888

Table 5: Precision and recall for different widths of a beam applied as a filter during the parse

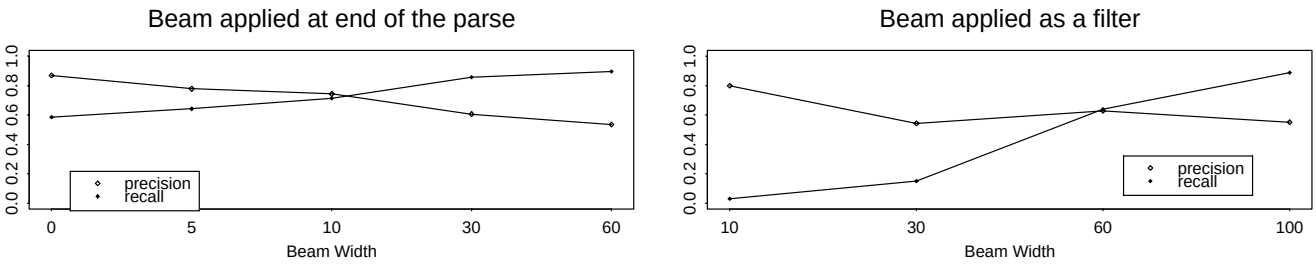

Figure 1: Precision and recall for both beam strategies. BE stands for beam applied at end of parse, BF for beam as a filter.

3.4 Beam applied during and after the parse

With the third experiment, we wanted to determine if a beam applied at the end of the parse would improve significantly the results of a beam filter applied during the parse. We expect that this additional filter will improve the precision, but lower the recall of the system. Based on the results from the second experiment, which give the value of 60 for the beam width. we found the results reported in Table 6.

Compared to the precision and recall values of the beam filter, the results show a general improvement in precision without a too high degradation of recall. For width 10, the gain in precision is 18% for a loss in completeness of 3%. Width 5 allows a gain in precision of 24% for a loss in recall of 11%. Width 0 shows better results: the gain in precision is 34% and the loss of recall is 17%.

Filter	precision	recall
BE 0	0.846	0.528
BE 5	0.780	0.568
BE 10	0.743	0.624

Table 6: Precision and recall of different width of a beam applied at the end of parse after that a beam of width 60 was applied during the parse

These results demonstrate that applying a beam twice, at two different points in the parse can improve significantly the performance of the system. The filtering is done once during the parse, in order to prune the search space from high-cost constituents, and it is applied once again after the parse is completed, in order to rank the analyses and select the preferred ones,

3.5 Discussion

All these experiments reported above show a degradation in the system's performance, thus they show that applying a beam is costly. To see if it is worth degrading the performance of the system by applying a beam filter, we look at the mean number of constituents hypothesized during the parse and at the mean number of complete analyses produced by the parser. We expect that the loss in performance is compensated by a significant reduction of these values. Table 7 shows these numbers for selected widths of beam filter and for the standard system run without filtering.

Beam	constituents	complete
BF 10	77	0
BF 30	97	2
BF 60	122	5
BF 100	140	9
BE or standard	149	12

Table 7: Mean number of constituents and complete analyses for beam applied during the parse (BF) with different widths, for beam applied after the parse (BE) and without any filtering (standard).

Compared to the performance of the standard system, 149 and 12 for constituents and analyses respectively, we gain 6% and 25% for the 100-width filter, 18% and 58% for the 60-width filter. The inaccurateness of the 10 and 30-width filter is confirmed by the mean number of complete analyses, 0 and 2 respectively.

As we have chosen the 60-width filter as the most promising one, we would have expected it to generate fewer constituents. This shows that in fact there is much room from improvement, which we hope to obtain by using different methods.

Finally, we have also made sure that the performance of the beam was not an artifact of other simpler factors, such as length of the sentence, where length is the number of words of the input sentence. We found a significant, but very low correlation (R= 0.2889, p= 0.0239) between the length of the sentence and the number of hypothesized constituents. Thus we conclude that the mean number of constituents is a meaningful way of evaluating the filter, as it cannot be completely explained by some other variable. On the other hand, we found a significant correlation (R=0.4323, p= 0.0005) between the length of the sentence and the number of complete analyses produced by the parser. Since sentence length explains approximately 18% of the variance in the number of complete analyses, we tentatively conclude that the mean number of complete analyses is not a significative test for evaluating the filter. In general, the fact that the correlations are not very high is not surprising: the number of generated constituents and the number of complete analyses depend on the lexical and syntactic ambiguity mainly: a short sentence containing highly ambiguous words will generate many more spurious analyses than a long sentence with less ambiguous words.

4 Conclusion

In this paper, we have presented a beam search algorithm for the IPS system. The metric of the beam is based on the structural cost of the constituents hypothesized by the parser.

Its width is the distance from the most preferred hypothesis. It is used for pruning dispreferred hypothesis and ranking complete analyses. We have also presented some experiments undertaken to select a good value for the beam width.

The conclusion we can draw from these results is that a beam search strategy requires setting the beam width carefully. The chosen value depends mainly on the task we would like to achieve, *i.e.* if we are interested in the completeness of the system or in its exactitude.

The data we have gathered will serve to test a stochastic beam search, the last phase of the project.

References

Berthouzoz, C. and P. Merlo (1996). Beam search strategy for ips: Evaluation of the structure-based heuristics. Notes techniques 96/9, Projet SAMSARA, LATL, University of Geneva.

Briscoe, T. and J. Carroll (1993). Generalized probabilistic lr parsing of natural language (corpora) with unification based grammars. *Computational Linguistics 19*(1), 25–60.

Chow, Y.-L. and R. Swartz (1989). The n-best algorithm: An efficient search procedure for finding top n sentence hypotheses. In *Proceedings of DARPA Speech and Natural Language Workshop*, pp. 199–202.

Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the ACL*.

Dowty, D., L. Karttunen, and A. Zwicky (Eds.) (1985). *Natural language parsing. Psychological, computational, and theoretical perspectives.* Cambridge University Press.

Gibson, E. (1991). *A computational Theory of Human Linguistic Processing: Memory Limitations and Processing Breakdown.* Ph. D. thesis, Carnegie Mellon University.

Kaplan, R. (1973). A general syntactic processor. In R. Rustin (Ed.), *Natural Language Processing*, pp. 193–241. New York: Algorithmics Press.

Kay, M. (1967). Experiments with a powerful parser. In *COLING-67.*

Reyle, U. and C. Rohrer (Eds.) (1988). *Natural Language Parsing and Linguistic Theories.* Reidel.

Walther, C. (1993). Psycholinguistically-based selection of analyses in ips and fips. Notes techniques 93/3, Projet FIPS, LATL, University of Geneva.

Wehrli, E. (1992). The ips system. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, Nantes, pp. 870–874.

Wehrli, E. (1993). Vers un système de traduction interactif. In P. Bouillon and A. Clas (Eds.), *La Traductique*, pp. 423–432. Les Presses de l'Université de Montréal.

Winston, P. (1981). *Artificial Intelligence.* Addison-Wesley.

A Test sentences

1. He saw a man with a telescope.
2. I saw the man without clothes.
3. Paul saw Mary under the bridge.
4. She hits the man with an umbrella.
5. I followed the man with a bicycle.
6. John met his girlfriend under pressure.
7. Mary saw John and the dog over the hill.
8. He closed the windows with wooden panels.
9. John sent the books and the letter to Mary.
10. She talked to the president of the meeting.
11. John gave the candies to the boy in the box.
12. I put the chair near the table with a scratch.
13. John sent the books and the letter to Mary to Paul.
14. She warned the students of the English examination.
15. She warned the specialists of the war of the danger.
16. She placed the sofa near the armchair with a red cover.
17. She talked to the president of the company of the meeting.
18. The lightning struck the tree near the house over the hill.
19. The rain damaged the computers near the boxes with plastic tops.
20. The rain damaged the computers near the empty boxes with colour screens.
21. The company acknowledges some problems.
22. When I realized the story was over I cried.
23. I noticed the car was bouncing all the time.
24. Her sister wishes Paul had a different job.
25. While John was mending the sock fell off his lap.
26. Paul admitted his company had overstated profits.
27. We could not find the charity does anything good.
28. The biologist noticed some strange bird behaviour.
29. He asserts John can not react quickly to competition.
30. They appreciate the economic stability we have achieved.
31. I know my father would have wished a girl instead of a boy.
32. The league's promoters hope tourists will join fans like Paul.
33. While they acknowledge the president will attend several meals.
34. Paul was acknowledging a month's figures do not prove a trend.
35. They could also find a customer that accept a lower grade of oil.
36. Traders and market officials hope the damage will not be permanent.
37. There is no advantage if the president asserts a right of excision.
38. Without admitting any guilt the bureau agreed to several conditions.
39. The company said it expects to realize those gains before the end of the year.
40. The two judges conceded that they assume the quotations were deliberately altered.
41. The president said he would assume Paul's responsibilities if a successor is not found.
42. John had to go.
43. John wants to go.
44. John is happy to go.
45. Give it to this child!
46. Has he never gone there?
47. She is more patient than John.
48. Where will you go tomorrow?
49. This book is easy to talk of.
50. The apple is very easy to eat.

51. John thinks that Mary will come.
52. John wanted to introduce him to us.
53. Is Mary coming because she wants to?
54. John seems to have liked Mary's eyes.
55. At what was the cat looking yesterday?
56. Have they promised each other to look at themselves?
57. Mary said that John made her read it to the children.
58. Paul wanted to have his car fixed by his brother's friend.
59. The small white mouse has been looked at by my sister's cat.
60. I was reading the Sunday afternoon newspapers that my brother bought.
61. The last suicide attempt of the king has been reported in the Sunday newspapers.

B Results

Sentence	Complete	Correct	Checked	Expected
1	7	2	2	2
2	12	4	4	4
3	2	1	1	1
4	2	2	2	2
5	2	2	2	2
6	2	2	2	2
7	17	2	2	2
8	2	2	2	2
9	2	2	2	2
10	2	2	2	2
11	7	5	2	2
12	6	3	2	3
13	2	2	2	1
14	4	4	4	6
15	5	3	3	3
16	9	9	6	8
17	4	4	3	3
18	21	12	3	4
19	12	6	6	8
20	12	6	6	8
21	4	4	1	1
22	20	6	0	1
23	4	1	1	1
24	1	1	1	1
25	50	3	1	1
26	2	2	1	2
27	15	15	1	1
28	3	3	1	2
29	3	3	2	2
30	5	2	1	1
31	8	3	1	1
32	95	29	8	8
33	26	2	0	0
34	4	4	1	1
35	2	2	1	1
36	39	3	3	3
37	46	0	0	1
38	98	6	1	1
39	14	7	4	4
40	3	2	1	1
41	36	24	2	2
42	1	1	1	1
43	1	1	1	1
44	1	1	1	1
45	3	1	1	1
46	4	2	2	1
47	1	1	1	1
48	6	1	1	1
49	49	1	1	1
50	9	2	1	1
51	1	1	1	1
52	2	2	1	1
53	2	0	0	1
54	1	1	1	1
55	2	1	1	1
56	2	2	1	1
57	3	3	2	2
58	11	11	1	1
59	14	4	3	3
60	3	3	1	1
61	8	2	1	1
Total	734	238	113	126

Table 8: Initial values for the number of complete, correct, checked and expected analyses for the test sentences.

Sentence	BE 0		BE 5		BE 10		BE 30		BE60	
	correct	checked	correct	checked	correct	checked	correct	checked	correct	checked
1	2	2	2	2	2	2	2	2	2	2
2	4	4	4	4	4	4	4	4	4	4
3	1	1	1	1	1	1	1	1	1	1
4	2	2	2	2	2	2	2	2	2	2
5	2	2	2	2	2	2	2	2	2	2
6	2	2	2	2	2	2	2	2	2	2
7	2	2	2	2	2	2	2	2	2	2
8	2	2	2	2	2	2	2	2	2	2
9	1	1	1	1	2	2	2	2	2	2
10	1	1	1	1	2	2	2	2	2	2
11	1	1	1	1	2	1	5	2	5	2
12	2	1	2	1	2	1	3	2	3	2
13	2	2	2	2	2	2	2	2	2	2
14	2	2	4	4	4	4	4	4	4	4
15	2	2	2	2	2	2	3	3	3	3
16	2	2	4	4	4	4	8	6	9	6
17	1	1	1	1	3	3	4	4	4	4
18	2	2	3	2	3	2	11	3	12	3
19	2	2	2	2	4	4	6	6	6	6
20	2	2	4	4	4	4	6	6	6	6
21	1	1	2	1	2	1	4	1	4	1
22	3	0	3	0	3	0	6	0	6	0
23	1	1	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1	1	1
25	1	1	1	1	1	1	3	1	3	1
26	1	1	1	1	1	1	1	1	2	1
27	2	0	6	1	10	1	15	1	15	1
28	1	1	2	1	2	1	3	1	3	1
29	1	1	1	1	1	1	2	1	3	2
30	1	1	1	1	1	1	1	1	2	1
31	1	1	1	1	1	1	2	1	3	1
32	2	1	4	1	6	3	11	7	17	8
33	1	0	1	0	2	0	2	0	2	0
34	1	1	2	1	2	1	2	1	2	1
35	1	0	2	0	2	0	2	0	2	0
36	2	2	2	2	2	2	3	3	3	3
37	0	0	0	0	0	0	0	0	0	0
38	2	1	2	1	2	1	2	1	3	1
39	2	2	2	2	2	2	3	2	5	4
40	1	1	1	1	1	1	1	1	2	1
41	1	1	2	1	2	1	7	2	18	2
42	1	1	1	1	1	1	1	1	1	1
43	1	1	1	1	1	1	1	1	1	1
44	1	1	1	1	1	1	1	1	1	1
45	1	1	1	1	1	1	1	1	1	1
46	2	1	2	1	2	1	2	1	2	1
47	1	1	1	1	1	1	1	1	1	1
48	1	1	1	1	1	1	1	1	1	1
49	1	1	1	1	1	1	1	1	1	1
50	1	1	1	1	1	1	2	1	2	1
51	1	1	1	1	1	1	1	1	1	1
52	1	1	1	1	1	1	1	1	2	1
53	0	0	0	0	0	0	0	0	0	0
54	1	1	1	1	1	1	1	1	1	1
55	1	1	1	1	1	1	1	1	1	1
56	1	1	1	1	1	1	1	1	2	1
57	1	1	1	1	1	1	1	1	3	2
58	1	1	1	1	1	1	7	1	9	1
59	2	1	2	1	3	2	4	3	4	3
60	1	1	2	1	3	1	3	1	3	1
61	1	1	1	1	2	1	2	1	2	1
Total	85	73	104	80	121	89	178	107	211	112

Table 9: Number of correct and checked analyses for different width of a beam applied at the end of the parse.

Sent	BF 10				BF 30				BF 60				BF 100			
	comp	corr	OK	cons	comp	corr	OK	cons	comp	corr	OK	cons	comp	corr	OK	cons
1	0	0	0	57	0	0	0	63	5	2	2	73	7	2	2	75
2	0	0	0	60	1	0	0	67	11	4	4	78	12	4	4	79
3	0	0	0	48	0	0	0	48	2	1	1	52	2	1	1	52
4	0	0	0	35	0	0	0	38	2	2	2	44	2	2	2	47
5	0	0	0	45	0	0	0	50	2	2	2	54	2	2	2	54
6	0	0	0	26	0	0	0	28	2	2	2	30	2	2	2	30
7	0	0	0	101	0	0	0	114	16	2	2	140	17	2	2	142
8	0	0	0	35	0	0	0	41	2	2	2	43	2	2	2	43
9	0	0	0	86	1	1	1	95	2	2	2	96	2	2	2	96
10	0	0	0	31	0	0	0	39	2	2	2	43	2	2	2	43
11	0	0	0	71	0	0	0	94	4	4	2	133	7	5	2	135
12	0	0	0	98	0	0	0	112	4	2	1	139	6	3	2	143
13	0	0	0	96	0	0	0	122	2	2	2	126	2	2	2	126
14	0	0	0	49	0	0	0	69	4	4	4	77	4	4	4	77
15	0	0	0	56	0	0	0	85	2	2	2	104	4	3	3	110
16	0	0	0	101	0	0	0	124	4	4	4	166	9	9	6	172
17	0	0	0	43	0	0	0	62	3	3	3	81	4	4	3	83
18	0	0	0	94	0	0	0	123	3	3	2	188	12	12	3	297
19	0	0	0	118	0	0	0	146	4	2	2	180	12	6	6	184
20	0	0	0	193	0	0	0	307	0	0	0	471	12	6	6	518
21	1	1	1	22	3	3	1	25	4	4	1	25	4	4	1	25
22	0	0	0	50	2	1	0	82	13	6	0	96	18	6	0	101
23	0	0	0	65	0	0	0	77	4	1	1	79	4	1	1	79
24	0	0	0	60	1	1	1	64	1	1	1	76	1	1	1	77
25	0	0	0	210	30	1	1	297	39	2	1	332	46	3	1	357
26	0	0	0	49	1	1	1	60	1	1	1	62	2	2	1	62
27	0	0	0	91	6	6	1	102	15	15	1	102	15	15	1	102
28	0	0	0	41	2	2	1	46	3	3	1	50	3	3	1	50
29	0	0	0	43	0	0	0	45	1	1	1	47	2	2	1	48
30	0	0	0	62	0	0	0	68	3	1	1	88	5	2	1	88
31	0	0	0	75	0	0	0	84	0	0	0	90	6	2	1	104
32	0	0	0	396	1	0	0	475	14	7	4	672	34	15	8	933
33	6	1	0	89	25	2	0	147	26	2	0	147	26	2	0	147
34	0	0	0	174	2	2	1	206	2	2	1	241	4	4	1	282
35	0	0	0	95	0	0	0	109	0	0	0	135	4	2	0	141
36	0	0	0	248	4	2	2	307	10	3	3	452	16	3	3	480
37	0	0	0	87	0	0	0	102	0	0	0	169	32	0	0	254
38	0	0	0	86	4	2	1	183	8	2	1	354	42	4	1	595
39	0	0	0	90	0	0	0	128	0	0	0	145	4	3	4	161
40	0	0	0	84	2	1	1	113	3	2	1	113	3	2	1	113
41	0	0	0	106	0	0	0	119	10	5	1	157	30	18	2	192
42	1	1	1	23	1	1	1	23	1	1	1	23	1	1	1	23
43	1	1	1	25	1	1	1	29	1	1	1	29	1	1	1	29
44	0	0	0	23	1	1	0	41	1	1	1	41	1	1	1	41
45	0	0	0	30	3	1	1	39	3	1	1	39	3	1	1	39
46	0	0	0	24	2	2	1	26	4	2	1	26	4	2	2	26
47	0	0	0	45	0	0	0	51	1	1	1	54	1	1	1	54
48	0	0	0	38	3	1	0	47	4	1	1	47	6	1	1	47
49	0	0	0	74	0	0	0	105	9	0	0	205	46	1	1	239
50	0	0	0	55	2	0	0	83	8	2	1	116	8	2	1	120
51	1	1	1	46	1	1	1	50	1	1	1	55	1	1	1	55
52	0	0	0	34	0	0	0	36	1	1	1	37	2	2	1	38
53	0	0	0	36	1	0	0	46	2	0	0	46	2	0	0	46
54	0	0	0	44	1	1	1	45	1	1	1	45	1	1	1	45
55	0	0	0	40	2	1	1	44	2	1	1	44	2	1	1	44
56	0	0	0	49	0	0	0	64	1	1	1	65	2	2	1	66
57	0	0	0	70	0	0	0	75	1	1	1	77	1	1	2	77
58	0	0	0	55	0	0	0	85	1	1	1	113	7	7	1	128
59	0	0	0	144	2	0	0	165	8	4	1	197	12	4	3	217
60	0	0	0	84	0	0	0	108	1	1	1	126	3	3	1	126
61	0	0	0	97	0	0	0	108	0	0	0	132	2	0	1	154
Total	10	5	4	77	105	35	19	97	284	127	80	122	529	202	111	140

Table 10: Number of complete, correct, checked and expected analyses and number of generated constituents for different widths of a beam applied as filter during the parse.

C Some modifications of the parser

C.1 Logical delete

One modification of the chart data structure we would like to adopt is setting up a **logical delete** instead of an actual one. This can be done simply by adding the field `delete : BOOLEAN` to the data structure `Projection`[6]. The constituents which are to be deleted will have this field set to `TRUE`.

A drawback of this modification will be that a test must be added when looking for previous constituents during the parse:

```
IF (const^.delete = FALSE) THEN
  (* do what you want with this constituent *)
END;
```

On the other hand, this modification will make the computational cost lighter because there will be no need to go through the structure and actually delete the constituents.

C.2 C.2 Grammar violations

There are three kinds of violations of the grammar: 1) *recoverable* violations, *i.e.* the constituent can still be completed later on, but if it is not, the sentence is ungrammatical, 2) *non-recoverable* violations, *i.e.* the constituent cannot be completed, but the degree of ungrammaticality is still acceptable, and 3) *fatal* violations, the constituent cannot be completed and the analysis is discarded.

It must be said that the violations do not discard the constituents until the end of the parse even in case of non recoverable violations. The cost of the violations is added to the structural cost of the constituents only at the end of the parse. The resulting value is used to rank the complete analyses.

To reflect better the degree of severity of the violations we propose another set of values given in Table 11 together with the actual values. These new values are not yet tested in the system.

[6] `Projection` is the very large data structure containing all the fields needing for chart parsing with a GB grammar

Recoverable violations	Proposed cost	Actual cost
incomplete DP	50	20
thetaless subject	60	30
subjectless TP	60	30
caseless subject	70	40
complementless to	80	50
incomplete PP	90	60
incomplete passive	100	100
Non recoverable violations		
no determiner	10	10
sentential subject	10	10
no lexical match	15	15
heavy NP-shift	20	20
wrong punctuation	30	30
WH in situ	40	30
Fatal violation		
open abar chain	1000	100

Table 11: Proposed and actual weight of the grammar violations