Neural Computation, 11(8):1995-2016, 1999.

How to Design a Connectionist Holistic Parser?

Ho K. S. Edward
Department of Systems Engineering and Engineering Management,
The Chinese University of Hong Kong,
Shatin, New Territories, Hong Kong

Chan L. W.
Department of Computer Science and Engineering
The Chinese University of Hong Kong,
Shatin, New Territories, Hong Kong

Connectionist holistic parsing offers a viable and attractive alternative to traditional algorithmic parsers. With exposure to a limited subset of grammatical sentences and their corresponding parse trees only, a holistic parser is capable of learning inductively the grammatical regularity underlying the training examples which effects the parsing process. In the past, various connectionist parsers had been proposed. Each approach has its own unique characteristics and yet some techniques are shared in common. In this paper, various dimensions underlying the design of a holistic parser will be explored, including the methods to encode the sentences and the parse trees, whether the sentences and the parse trees share the same representations, the use of confluent inference and the inclusion of phrases in the training set. Different combinations of these design factors will give rise to different holistic parsers. In succeeding discussions, we scrutinize these design techniques and compare the performances of a few

parsers on language parsing, including the confluent preorder parser (CPP), the backpropagation parsing network, the XERIC parser of Berg, the connectionist modular parser of Sharkey & Sharkey, Reilly's model and their derivatives. Experiments are performed to evaluate their generalization capability and robustness. The results unveil a number of issues which are essential for building an effective holistic parser.

1 Language Parsing

In language processing, parsing addresses the problem of finding the hierarchical relationship between the terminals or words in a sequential sentence. Traditionally, this relationship is manifested in the form of a tree: the parse tree. For example, by classifying the words in the sentence "the boy takes the apple on the table", we form a sequence $\langle$ D N V D N P D N $\rangle$, where the terminal D stands for determiner, N for noun, V for verb and P for preposition. Upon successful parsing, the parse tree in Figure 1 is produced, where the non-terminal *np* stands for noun phrase, *vp* for verb phrase, *pp* for prepositional phrase and *s* for sentence. Alternatively, it can also be represented by the Lisp-like structure ((D N) (V ((D N) (P (D N))))).

2 Algorithmic Parsing

2.1 Symbolic Algorithmic Parsers. For a very long time, the computational linguistics approaches have dominated the study of language parsing. Numerous

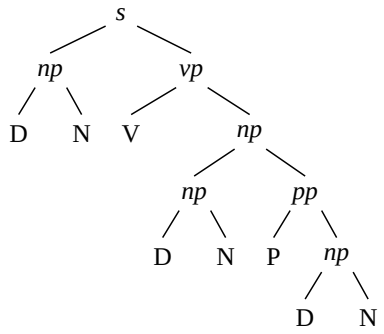

Figure 1: The parse tree of the sentence $\langle$ D N V D N P D N $\rangle$.

models have been proposed, such as the chart parser, the shift-reduce parser and the augmented transition network (Aho et al. 1986; Allen 1995; Gazdar & Mellish 1989; Krulee 1991). Generally speaking, these models share several common characteristics. First, they are algorithmic and rule-based. A well-defined parsing algorithm is implemented which governs the parsing process. During parsing, the input sentence will be analyzed step-by-step, with the target parse tree being built up incrementally at the same time. Second, symbolic representations, such as character strings or pointer structures, will be adopted for the sentences and the parse trees. Third, the construction of the parser usually requires a detailed knowledge of the underlying grammar.

Despite their popularity, the computational linguistics approaches have been criticized of their inadequacy in handling natural languages. In particular, their rule-based nature fails to exhibit the kind of robustness as found in actual language usages. Poor error recovery capability is thus resulted. Second, sym-

bolic representations are simply too crisp to reflect the richness and fuzziness as manifested by natural language structures and meanings. Third, the extreme flexibility of natural language grammars has effectively prohibited the complete enumeration of the grammar rules. Even worse, some of the grammar rules may have to be modified in order to "adapt" to the parser (e.g. in a top-down parser, all left-recursions in the grammar rules have to be removed first).

2.2 Connectionist Algorithmic Parsers. In view of the drawbacks of the traditional approaches, researchers have begun to switch their focus to more data-oriented paradigms. For example, in the statistical or corpus-based approaches (Charniak 1993), statistics or co-occurrence information (such as n-gram) is calculated from a large amount of sample text which can then be used for the parsing task. For example, Franz (1996) has made used of corpus statistics to resolve pp-attachment ambiguities.

Recently, as motivated by the study of cognitive science, connectionist techniques are applied in natural language processing also. Compared with other approaches, neural network parsers have the appeals that they are inherently robust. They are capable of learning inductively and incrementally from examples, and they can generalize naturally to unseen sentence structures.

To exploit these advantages, effort has been paid to integrate the connectionist techniques into the symbolic processing framework, giving rise to various "hybrid" parsers. For example, in the massively parallel parsing system (Pollack & Waltz 1985), a chart parser is used to generate all possible parses of an input

sentence, which are then translated into a connectionist network. Alternative syntactic classes and senses of the words as well as the pragmatic constraints are represented by nodes which are connected via excitatory or inhibitory links. The overall interpretation of the sentence is obtained via parallel constraint satisfaction.

The connectionist deterministic parser (CDP) (Kwasny & Faisal 1992) consists of a symbolic component which is a deterministic parser (Marcus 1980), and a subsymbolic component which is a feedforward network. During parsing, based on the contents of the parser's symbolic data structures (which include a stack and a lookahead buffer), the feedforward network outputs the action to perform and the parser manipulates its data structures accordingly. Effectively, the rigid rules are replaced by the feedforward network which has a more flexible decision boundary, thus achieving robustness.

Other examples include the PARSEC model (Jain 1991), the subsymbolic parser for embedded clauses (SPEC) (Miikkulainen 1996), and the neural network pushdown automaton (NNPDA) (Sun et al. 1993).

However, these early attempts still require the detailed specification of the parsing algorithm and the grammar. In some sense, they are partial or complete neural network re-implementation of some symbolic algorithm only. In other words, they are still algorithmic and rule-based in nature, although the rules may now have more flexible decision boundaries (if they are implemented using neural networks). In many cases, symbolic representations are even retained. The

inductive learning power of neural networks simply has not been fully utilized, and error recovery performance is not satisfactory.

3 Holistic Parsing

Connectionist representation of a symbolic data structure provides the benefit that it allows the manipulation of the vector representation as a whole. Effectively, the structure encoded is operated as a whole also, without first breaking it down into its constituent components. This type of operation, so called holistic transformation (Blank et al. 1992), is generally not supported by common symbolic encoding schemes. But it is useful for implementing structure-sensitive operations such as unification (Stolcke & Wu 1992) and transformation (Chalmers 1992).

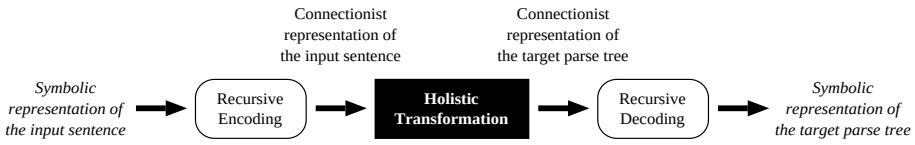

Figure 2: Holistic parsing.

Holistic transformation is equally applicable in language parsing. By developing connectionist representations for the sentence and the parse tree, parsing can be achieved by mapping holistically the sentence representation to the corresponding parse tree representation. We call this *holistic parsing* (see Figure 2).

4 Design Issues of Holistic Parsers

Based on the holistic parsing paradigm as shown in Figure 2, different design dimensions can be explored :

1. *Encoding the sentences.* First, different encoding mechanisms can be adopted for developing the sentence representations. Two common choices are the simple recurrent network (SRN) (Elman 1990) and the sequential recursive auto-associative memory (SRAAM) (Pollack 1990). And when an SRN is used, the sentence representation can be developed in two ways. On the one hand, the SRN can be trained to produce at its output layer the representation of the target parse tree (as obtained by certain connectionist encoding mechanism), upon reading each terminal in the input sentence sequence. Effectively, this implies that the sentence and the parse tree share the same representation.

   Alternatively, sentence coding can also be developed by training the SRN on a sequence prediction task: given the $1st$ to the $(i-1)th$ terminals of the sentence, the SRN predicts what the ith terminal is. Upon successful training, the hidden layer activation of the SRN after reading the whole sentence will be used as its coding.

2. *Encoding the parse trees.* In much the same way, different encoding schemes can be implemented for the parse trees also. Traditionally, the parse tree is represented as a hierarchical data structure and the recursive

auto-associative memory (RAAM) (Pollack 1990) is commonly applied for this task.

Previously, Kwasny & Kalman (1995) have proposed to linearize the parse tree by preorder traversal. For example, the parse tree in Figure 1 corresponds to the sequence $\langle$ s np D $*$ $*$ N $*$ $*$ vp V $*$ $*$ np np D $*$ $*$ N $*$ $*$ pp P $*$ $*$ np D $*$ $*$ N $*$ $*$ $\rangle$. Note that null pointers are explicitly added to the leaves of the tree, as represented by the special symbol $*$. The preorder traversal sequence, instead of the hierarchical parse tree, is then encoded by training an SRAAM.

3. *Implementing the Holistic transformation.* The holistic transformation can be realized in three different ways :

   (a) First, one may choose to develop different representations for the sentence and the parse tree. Then, a feedforward network is trained to effect an explicit transformation from the sentence representation to the parse tree representation.

   (b) Alternatively, the sentence or the parse tree may be encoded first. The representation thus obtained will then be used as the training target for the other encoding process. In this way, both the sentence and the parse tree will share the same representation, although their respective encoding mechanisms are carried out separately in fact.

   (c) Third, the representation of the sentence and that of the parse tree

may "co-evolve" at the same time, instead of being developed separately and independently, and an identical representation will be developed for the sentence and its parse tree. This technique is known as confluent inference (Chrisman 1991).

Note that for (b) and (c), the holistic transformation is effectively implemented as an identity mapping. In other words, the representation obtained via encoding the input sentence can be decoded directly to give the symbolic form of the target parse tree.

4. *Learning to parse phrases.* Originally, a holistic parser only learns to parse a complete sentence to give the total parse tree. But a complete sentence such as $\langle$ D N V D N P D N $\rangle$ in fact consists of a number of phrases: the noun phrase $\langle$ D N $\rangle$, the prepositional phrase $\langle$ P D N $\rangle$, the noun phrase $\langle$ D N P D N $\rangle$ and the verb phrase $\langle$ V D N P D N $\rangle$. They are parsed to give the subtrees (D N), (P (D N)), ((D N) (P (D N))) and (V ((D N) (P (D N)))) respectively. So in addition to complete sentences, a holistic parser can be taught to map the connectionist representation of a phrase to the corresponding subtree representation. A parser being trained this way will have a better knowledge of the internal structures of the sentences and their parse trees. Better generalization performance in parsing can therefore be expected.

5 Previous Holistic Parsers

Different combinations of the above-mentioned techniques will give rise to different holistic parsers. Some of these combinations have in fact been previously studied.

5.1 Reilly's parser. In Reilly's parser (Reilly 1990), the parse trees as hierarchical data structures are first encoded using RAAM. An SRN is then trained to develop at its output layer the RAAM representation of the target parse tree upon reading the last terminal of a complete sentence.

5.2 XERIC parser. Similar to Reilly's model, in the XERIC parser (Berg 1992), parse trees are encoded as hierarchical data structures using RAAM and an SRN is trained to output the RAAM representation of the target parse tree. Unlike the previous approach, the RAAM network and the SRN are now trained together coherently. In other words, confluent inference has been applied. Moreover, the parser learns to parse phrases in addition to complete sentences.

5.3 Connectionist modular parser. In the connectionist modular parser proposed by Sharkey & Sharkey (1992), parse trees are first encoded as hierarchical data structures using RAAM. An SRN is then trained on a sequence prediction task using the sentences. Upon successful training, the hidden layer activation after reading the whole sentence will be used as its coding. A 3-layered feedforward network then maps explicitly the sentence representation produced by the SRN to the parse tree representation developed by the RAAM to effect parsing. As in Reilly's parser, only complete sentences are considered.

5.4 Backpropagation parsing network (BPN). In the BPN (Ho & Chan 1994), the sentences and their respective parse trees are first encoded by training an SRAAM and an RAAM respectively and independently. Then, a 3-layered feedforward network is trained to map the SRAAM coding representing a sentence to the RAAM coding representing its corresponding parse tree. So both linearization and confluent inference are missing and the parser learns to parse complete sentences only.

5.5 Confluent preorder parser (CPP). The architecture of the confluent preorder parser (CPP) (Ho & Chan 1997) is shown in Figure 3. In the CPP, two techniques have been integrated. First, the hierarchical parse tree is linearized by preorder traversal into a sequence, as proposed by Kwasny & Kalman (1995). But unlike their approach, null pointers are not added explicitly to the leaves of the tree. Thus the parse tree in Figure 1 gives rise to the sequence $\langle$ s np D N vp V np np D N pp P np D N $\rangle$. It is this preorder traversal sequence of the parse tree which is encoded, instead of the parse tree itself. Parsing is thus achieved via a sequence-to-sequence transformation: from the sentence sequence to the preorder traversal sequence, and both types of sequences are encoded using SRAAM.

To further promote the generalization performance, confluent inference (Chrisman 1991) is applied. The SRAAM for sentence encoding and the SRAAM for preorder traversal encoding are trained together in a coherent manner, such that identical representation is developed for the sentence and the preorder traversal

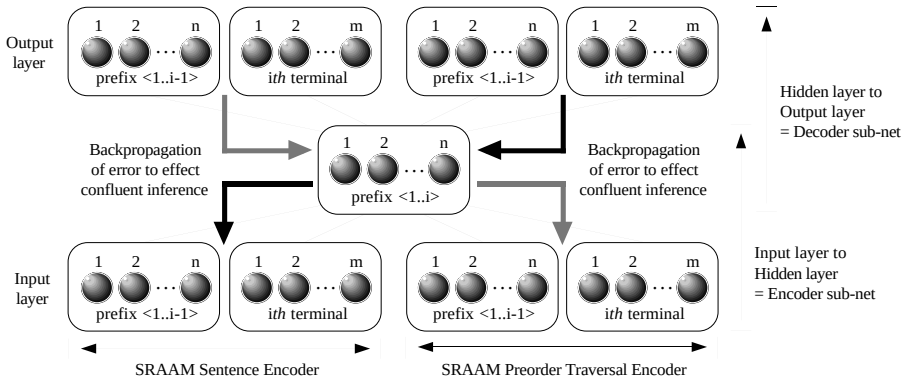

Figure 3: Confluent preorder parser (CPP).

of its corresponding parse tree. In this way, the sentence representation obtained by the sentence encoder can be decoded directly by the preorder traversal encoder to give the target preorder traversal sequence, thus saving an explicit transformation between the two types of representations.

Two versions of the CPP have been implemented. In *CPP1*, the parser is trained using complete sentences only, whereas in *CPP2*, in addition to complete sentences, the parser learns to parse phrases to produce the corresponding subtrees. Simulation results show that they have excellent generalization performances. Besides, they are capable of recovering erroneous sentences and resolving lexical category ambiguities (Ho & Chan 1997).

In succeeding discussions, these parsers will be abbreviated as *Reilly*, *Berg*, *Sharkey* and *BPN* respectively. Summarized in Table 1 are the specific design decisions that have been adopted by each parser.

Table 1: The design configurations of different holistic parsers (*SRN** denotes that the SRN network for sentence encoding is being trained on a sequence prediction task. The suffixes "1" and "2" to the names of the parsers indicate modification of the original design to including or not including the learning of phrases in addition to complete sentences)

	Encode sentences by			*Linearize parse trees*	*Same coding for sentence & parse tree*	*Confluent inference*	*Parse phrases*
	*SRN**	*SRN*	*SRAAM*				
Berg		√			√	√	√
Reilly		√			√		
Sharkey	√						
BPN			√				
CPP1			√	√	√	√	
CPP2			√	√	√	√	√
Berg1		√			√	√	
Reilly2		√			√		√
Sharkey2	√						√
BPN2			√				√

6 Comparing the Performances of Different Holistic Parsers

In succeeding discussions, an experimental comparison will be carried out to evaluate the different holistic parsers. Both their generalization performances and error recovery capabilities are concerned.

6.1 The Grammar Used. We use the context-free grammar as shown in Table 2 which is adopted from Pollack (1990).

A total of 112 sentences and their corresponding parse trees are generated. The length of the longest sentence is 17, while the highest parse tree consists of 5 levels. Among them, 80 sentences are randomly selected for training while

Table 2: The context-free grammar used for evaluation (Pollack, 1990)

sentence	*noun phrase*	*verb phrase*	*prepositional phrase*	*adjectival phrase*
s → np vp s → np V	np → D ap np → D N np → np pp	vp → V np vp → V pp	pp → P np	ap → A ap ap → A N

the remaining 32 sentences are reserved for the testing phrase.

6.2 Results.

Generalization. Each parser is first trained using the same set of training sentences (totally, there are 80 training sentences) and generalization capability is then measured by evaluating its performance on parsing the same set of 32 testing sentences. For each parser, a total of 4 runs have been performed, and in each run, random values are assumed for the weights initially. The results have been summarized in Figure 4.

A point to note is that in some of these approaches, including *CPP2* and *Berg*, the parsers have been taught to parse phrases in addition to complete sentences. As we have claimed in Section 4, a parser being trained in this way would have a better knowledge of the internal structures of the sentences and their parse trees. Better generalization performance in parsing can thus be expected. As a result, in order to have a "fair" comparison, apart from those previously proposed parsers, four other models have been implemented also, namely, *Reilly2*, *Berg1*, *Sharkey2* and *BPN2*. They are derivatives of *Reilly*, *Berg*, *Sharkey* and *BPN* respectively. But unlike their corresponding original

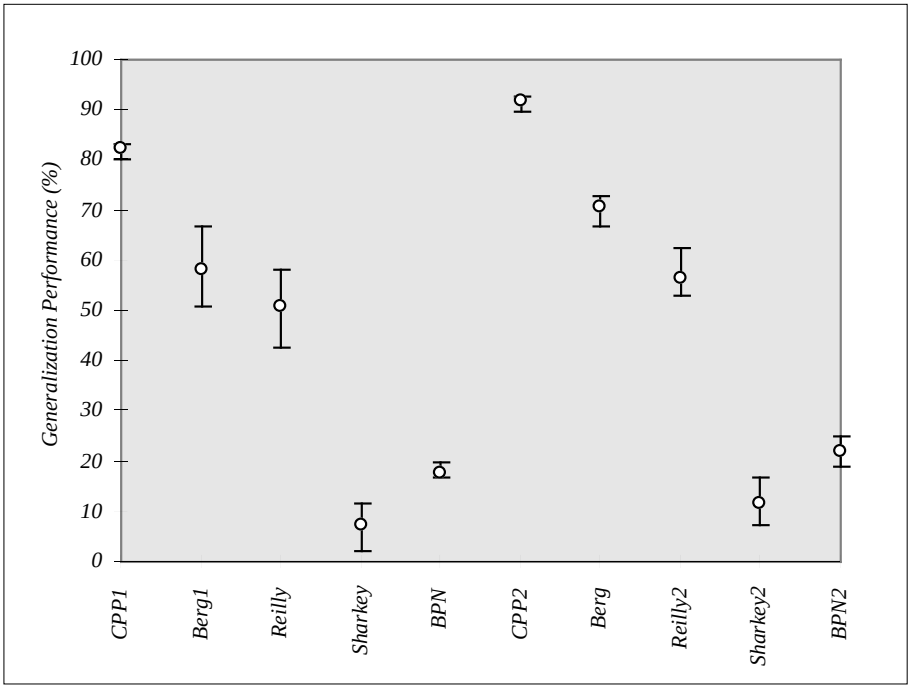

Figure 4: Generalization performances of different holistic parsers.

design, *Reilly2*, *Sharkey2* and *BPN2* have been trained to parse both complete sentences and phrases,while for *Berg1*, the parser learns to parse complete sentences only. The design decisions adopted by these parsers are shown in Table 1 also.

Error Recovery. In addition to generalization performances, we compare the robustness of the holistic parsers by evaluating their capabilities in parsing erroneous sentences.

Four types of errors are considered, namely, erroneous sentences with one ter-

minal substituted by a wrong terminal (SUB), with an extra terminal inserted (INS), with one terminal omitted (OMI), and with two neighboring terminals exchanged (EX). All erroneous sentences are obtained by modifying the 80 training sentences. Multiple erroneous sentences are generated by "injecting" error into every possible position of each training sentence. Totally, there are 853 SUB sentences, 853 OMI sentences. 933 INS sentences and 773 EX sentences.

These erroneous sentences are then parsed by each of the trained parser. A recovery is said to be successful if (a) the length of the sentence corresponding to the parse tree generated and that of the erroneous sentence differ by no more than 1, and (b) the number of mismatched (or different) terminals between these two sentences is at most 2. All the results are summarized in Table 3.

Table 3: Robustness of different holistic parsers (percentage of erroneous sentences that can be successfully recovered)

	SUB	OMI	INS	EX
CPP1	91.21%	69.64%	46.62%	63.65%
Berg1	62.02%	54.16%	41.26%	45.28%
Reilly	65.42%	59.79%	41.91%	46.05%
Sharkey	19.81%	31.42%	17.36%	20.96%
BPN	86.87%	63.19%	51.86%	67.14%
CPP2	94.72%	70.93%	50.80%	66.88%
Berg	69.17%	64.83%	50.80%	53.30%
Reilly2	55.92%	45.37%	31.73%	35.45%
Sharkey2	16.41%	29.78%	15.01%	16.82%
BPN2	84.17%	63.66%	46.30%	62.14%

7 Analysis and Evaluation

7.1 The Design Decisions of the CPP and their Justifications. Recall that in terms of the different design dimensions identified in Section 4, the design of the CPP can be described as follows (see also Table 1) :

1. *Linearization is adopted*
2. *Equal representation for the sentence and the parse tree*
3. *Confluent inference is applied*
4. *Learning to parse phrases (for CPP2)*
5. *Sentences are encoded using SRAAM instead of SRN*

Each of these design decisions can be justified by examining the experimental results as summarized in Figure 4 and Table 3 :

1. **Learning to parse phrases in addition to complete sentences improves generalization performance.**

   As revealed by Figure 4, in all cases, teaching a parser to parse phrases in addition to complete sentences can improve its generalization performance (e.g. compare *CPP1* with *CPP2*, *Reilly* with *Reilly2*, *Berg1* with *Berg*, *Sharkey* with *Sharkey2*, and *BPN* with *BPN2*). As we have claimed earlier, by teaching the parser to parse phrases also, it can have a better knowledge of the internal syntax of the sentence as well as the parse tree. Improvement in performance can therefore be expected.

2. **Sentence representation and parse tree representation should preferably be the same.**

   It can be observed that the generalization performances of *Sharkey* and *BPN* are significantly worse than the generalization performances of the other models.

   Recall that in both of these parsers, confluent inference is not applied. Moreover, the sentence representation (in *Sharkey*, it is obtained by training an SRN while in *BPN*, an SRAAM is used to encode the sentence) is different from the corresponding parse tree representation. Consequently, an explicit transformation has to be adopted which maps the sentence representation to the parse tree representation. In both *Sharkey* and *BPN*, this transformation is implemented by a feedforward network.

   Two disadvantages are evident for this approach. First, since two different representations will be developed independently for the sentences and the parse trees, it is reasonable to expect that the representation for a sentence and that of the parse tree will not be correlated in a regular manner that reflects the structural characteristics of the sentence and the parse tree as well as their relationships as defined by the underlying grammar rules. Consequently, the mapping as implemented by the feedforward network will tend to be arbitrary. Even worse, error will unavoidably be incurred in this explicit transformation. As a result, generalization performance is sacrificed.

3. **Applying confluent inference can improve generalization.**

   The generalization performance of *Berg1* is better than the generalization performance of *Reilly* (note that they both learn to parse complete sentences only). And when both models are trained using phrases in addition to complete sentences, the generalization performance of *Berg* is again better than that of *Reilly2*.

   By comparing their design configurations, the only difference between *Berg1* and *Reilly* is that confluent inference is adopted by the former, but not by the latter. In all other respects, they are the same. In much the same way, confluent inference is only adopted in *Berg*, but not in *Reilly2*. This suggests that the use of confluent inference can improve the generalization performance of a holistic parser.

4. **Encoding sentences using SRAAM can improve robustness.**

   Recall that in both *Sharkey* and *Sharkey2*, sentences are encoded by training an SRN on a sequence prediction task (as denoted by SRN^* in Table 1). In other words, the hidden layer activation of the SRN after reading the last terminal of the sentence will be used as its coding.

   This encoding method has the drawback that the sentence representation produced will depend very much on the last terminal(s) of the sentence. As a result, two sentences which end with the same terminal(s) may give rise to very similar representations, even if they are quite

different in the other parts and should be parsed to give different parse trees. In *Sharkey*, these two pieces of similar representations will have to be mapped (via a feedforward network) to two quite different parse tree representations. Training thus becomes difficult to converge.

More importantly, both generalization performance and robustness are sacrificed. In our comparative study, the design configurations of *Sharkey* and *BPN* are the same, except that sentences are encoded using SRAAM in *BPN*. As shown in Figure 4 and Table 3, *BPN* outperforms *Sharkey* in both generalization and robustness. Consistently, when both parsers are trained using phrases also, *BPN2* is again superior than *Sharkey2*. Therefore, we prefer SRAAM to SRN* for encoding sentence sequences.

On the other hand, sentence representations are produced differently in *Berg* and *Reilly*. An SRN is trained to produce the RAAM representation of the target parse tree, upon reading the last terminal of the sentence (as denoted by *SRN* in Table 1). With reference to Figure 4, the generalization performances of *Berg1* and *Reilly* are better than that of *BPN*. Similarly, when phrases in addition to complete sentences are used to train the parsers, *Berg* and *Reilly2* again outperform *BPN2* in generalization.

We believe that the discrepancy is mainly due to the fact that in *BPN*, the sentence and its parse tree have different representations. But despite the fact that its generalization performance may be unsatisfactory, *BPN* is significantly more robust than *Berg1* and *Reilly*, as unveiled by Table 3

(similarly, *BPN2* is more robust than *Berg* and *Reilly2*). An explanation can be given. In both *Berg* and *Reilly*, the parse tree coding as developed by the SRN will depend very much on the last few terminals of the sentence (much like the case of *SRN**), since the training target (i.e. the RAAM representation of the parse tree desired) will only be applied at the end of the sentence sequence. The "contribution" from other terminals is less. As a result, the parser will become sensitive to errors occurring in the trailing part of the sentence.

When an error does occur which involves some of these last terminals, a totally wrong parse tree coding will be produced (i.e. the coding will deviate significantly from the correct parse tree representation). The parser will thus fail to recover the error.

On the other hand, an SRAAM is basically the same as an SRN, except that auto-association is incorporated. This forces the SRAAM network to reproduce in each step (at its output layer) the current terminal of the sentence which is appearing in the input layer. As a result, the coding produced will not be biased for certain terminal(s) of the sentence and each terminal has a "substantial" influence on the final coding. So when certain part of the sentence is corrupted, there is still a good chance that the correct coding and parse tree can be produced, provided that the majority of the sentence remains "intact". Better robustness is thus resulted[1].

[1]The advantage of incorporating auto-association in an SRN has also been studied by

As mentioned at the beginning of this paper, error recovery capability is a primary concern in natural language parsing since ungrammatical or erroneous sentences occur very frequently in actual language usages. In fact, robustness is a major advantage of connectionist parsers over the traditional approaches. As a result, we prefer SRAAM to SRN for sentence encoding.

5. **Linearization can improve generalization performance and robustness.**

   Among the different holistic parsers studied, *CPP1* and *CPP2* are the only ones in which the parse trees are linearized by preorder traversal. As revealed by Figure 4 and Table 3, they have the best generalization performance and robustness when compared to the other models.

As a final remark, an interesting observation is worth of further consideration. Although the use of phrases in training can improve the generalization performance of each of the parsers studied, the robustness does not always benefit from that. As revealed by Table 3, only in *CPP* and *Berg* will the error recovery capability be actually improved by learning to parse both phrases and complete sentences. By examining the configurations of the parsers, the only models which exhibit improvement in robustness when taught to parse phrases also are those in which confluent inference has been adopted. In each of the

Maskara & Noetzel (1993) previously.

other cases, robustness drops slightly in fact.

A probable explanation can be given. As we have claimed earlier, with confluent inference, the sentence coding and the parse tree (or preorder traversal) coding will be evolving at the same time, thus affecting one another as a result. A regular correlation can therefore be established between them.

On the other hand, by using phrases in addition to complete sentences in training, an extra correlation is established between the representation of a phrase (e.g. $\langle$ D N $\rangle$) and that of the respective subtree (correspondingly (D N)). As there exists a part-whole relationship between a complete sentence (e.g. $\langle$ D N V $\rangle$) and its constituent phrases (correspondingly $\langle$ D N $\rangle$), as well as between a total parse tree (such as ((D N) V)) and its subtrees (correspondingly (D N)), it is hoped that this correlation between the representations of the phrase and the subtree can bring closer together the representation of the complete sentence and that of the total parse tree. In this way, if only minor error occurs in the input sentence, there is still a good chance that it can be encoded and then mapped to the representation of the correct parse tree. Robustness can therefore be promoted.

However, this advantage can only be exploited if confluent inference is applied in training also. With confluent inference, the two types of mappings: the extra correspondence between the representations of a phrase and its respective subtree, as well as the mapping between the representations of the complete sentence and the total parse tree, will be trained together at the same time.

In this way, each of them can influence the development of the other. Intuitively, the correspondence between phrases and subtrees will act as an extra "constraint" on the evolution of the representations of the complete sentence and the total parse tree (and vice versa). The final coding obtained will thus have taken into account the mapping between its constituent phrases and their respective subtrees also. But if confluent inference is not applied (or even that the sentence representation is different from the parse tree representation), the correspondence between the representation of the phrase and that of the subtree will simply exist as an extra arbitrary mapping only. In the worst case, it may become an interference to the encoding process and deteriorating robustness as a result.

7.2 The Overall Performance of the CPP. To summarize, the CPP with its specific combination of design decisions has achieved very good performance in parsing. With reference to Figure 4, when generalization is concerned, both *CPP1* and *CPP2* outperform all other implementations of holistic parsers. More importantly, given different initial conditions of training, a more stable performance level is achieved by the *CPP* (again, both *CPP1* and *CPP2*). The generalization performance in the best case and that in the worst case differ by 3% only. In addition, as revealed by Table 3, the *CPP* is also significantly more robust than other holistic parsers, especially when SUB and OMI errors are concerned.

8 Conclusions

Compared to algorithmic parsers, connectionist holistic parsers have the appeals that they are capable of learning inductively from examples. Little knowledge of the detailed parsing mechanism and the target grammar will thus be assumed. This knowledge is often unknown or debatable when natural language is concerned. Besides, connectionist holistic parsers are inherently robust. Having learned to parse grammatical sentences only, the parser automatically acquires the ability to recover erroneous sentences.

In this paper, we have presented a general framework for holistic parser design. Several design dimensions have been identified and discussed. We find that their exact combination will have a significant impact on both the generalization performance and the robustness of the resulting parser model. The experimental results have justified on an empirical basis that the CPP is superior than other holistic parsers previously proposed.

Acknowledgements

The authors gratefully acknowledge support from the Research Grants Council (RGC) of Hong Kong (Earmarked Research Grant CUHK 4133/97E).

References

Aho, A.V., Sethi, R., & Ullman, J.D. (1986). *Compilers: Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley.

Allen, J. (1995). *Natural Language Understanding*. Redwood City, CA: Benjamin/Cummings.

Berg, G. (1992). A connectionist parser with recursive sentence structure and lexical disambiguation. *Proc. of the Tenth National Conference on Artificial Intelligence (AAAI-92), San Jose* (pp. 32–37).

Blank, D.S., Meeden, L.A., & Marshall, J.B. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In J. Dinsmore (Ed.), *The Symbolic and Connectionist Paradigms: Closing the Gap* (pp. 113–148). Hillsdate, NJ: Lawrence Erlbaum Associates.

Chalmers, D.J. (1992). Syntactic transformation of distributed representations. In N. Sharkey (Ed.), *Connectionist Natural Language Processing* (pp. 46–55). Boston: Kluwer Academic Publishers.

Charniak, E. (1993). *Statistical Language Learning*. Cambridge: MIT Press.

Chrisman, L. (1991). Learning recursive distributed representations for holistic computation. *Connection Science*, *3*, 345–366.

Elman, J.L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Franz, A. (1996). Learning PP attachment from corpus statistics. In S. Wermter, E. Riloff, & G. Scheler (Eds.), *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing* (pp. 188–202). Berlin: Springer-Verlag.

Gazdar, G., & Mellish, C. (1989). *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*. Reading, MA: Addison-Wesley.

Ho, K.S.E., & Chan, L.W. (1994). Representing sentence structures in neural networks. *Proc. of the International Conference in Neural Information Processing Systems 3, Seoul* (pp. 1462–1467).

Ho, K.S.E., & Chan, L.W. (1997). Confluent preorder parsing of deterministic grammars. To appear in *Connection Science, 9*, 269–293.

Jain, A.N. (1991). Parsing complex sentences with structured connectionist networks. *Neural Computation, 3*, 110–120.

Krulee, G.K. (1991). *Computer Processing of Natural Language*. Englewood Cliffs, NJ: Prentice-Hall.

Kwasny, S.C., & Faisal, K.A. (1992). Symbolic parsing via subsymbolic rules. In J. Dinsmore (Ed.), *The Symbolic and Connectionist Paradigm: Closing the Gap* (pp. 209–236). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kwasny, S.C., & Kalman, B.L. (1995). Tail-recursive distributed representations and simple recurrent networks. *Connection Science, 7*, 61–80.

Marcus, M.P. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge: MIT Press.

Maskara, A., & Noetzel, A. (1993). Forced simple recurrent neural network and grammatical inference. *Proc. of the Fifteenth Annual Conference of the Cognitive Science Society*, (pp. 420–425).

Miikkulainen, R. (1996). Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science, 20*, 47–73.

Pollack, J.B. (1990). Recursive distributed representations. *Artificial Intelli-*

gence, *46*, 77–105.

Pollack, J.B., & Waltz, D. (1985). Massively parallel parsing: A strongly interative model of natural language interpretation. *Cognitive Science*, *9*, 51–74.

Reilly, R. (1990). Connectionist techniques for on-line parsing. *Network*, *3*, 37–45.

Sharkey, N.E., & Sharkey, A.J.C. (1992). A modular design for connectionist parsing. *Proc. of the Twente Workshop on Language Technology 3: Connectionism and Natural Language Processing*, (pp. 87–96).

Stolcke, A., & Wu, D. (1992). *Tree Matching with Recursive Distributed Representations*. Technical Report TR-92-025, International Computer Science Institute, Berkeley.

Sun, G.Z., Giles, C.L., Chen, H.H., & Lee, Y.C. (1993). *The Neural Network Pushdown Automata: Model, Stack and Learning Simulations*. Technical Report UMIACS-TR-93-77 & CS-TR-3118, University of Maryland.