Stratified Recursive Backup for Best First Search

P.A. Jones*, P. van Bommel, C.H.A. Koster, Th.P. van der Weide
Computing Science Institute
University of Nijmegen
The Netherlands

Technical Report CSI-R9720

November 10, 1997

Abstract

In this paper a new abstract machine model, the Stratified Recursive Backup machine model, is described. This machine model can be used to implement *best first search* algorithms efficiently. Two applications of best first search, a text layouting system and a natural language parser, are analyzed to provide an in-depth understanding of the Stratified Recursive Backup machine model.

*Dept. of Information Systems, Faculty of Mathematics and Computing Science, University of Nijmegen, Toernooiveld, NL-6525 ED Nijmegen, The Netherlands, E-mail: paulj@cs.kun.nl

Contents

1 Introduction

While working on several unrelated projects, the authors noticed an important similarity between them: each project could be viewed as a search problem where a 'best' solution is to be found in a very large search space, enriched with a quality metrics. Realizing that the projects could make use of a generic framework, a new project was started to define and implement such a framework. Resources from several major projects have been pooled to start developing the generic framework. These major projects are in the fields of natural language parsing, information retrieval and document generation.

The AGFL (Affix Grammars over Finite Lattices) project is an ongoing project in natural language parsing (see e.g. [Kos91]). Over the years a parser generator has been developed which is in use at several language departments spread out over the world. Several linguists are currently using this system to develop grammars for english, dutch, french, arabic, spanish, greek and russian. Furthermore, a number of industrial applications has been found. Currently, research in the AGFL project is directed toward probabilistic parsing which is seen as the best solution to ambiguity problems.

The PROFILE (Proactive Filtering) project concerns the development of active filtering technology for the internet (see e.g. [WBHW97]). The project covers the whole cycle of information gathering. In this project, grammars written in AGFL are used to locate nouns and noun phrases used to translate the information need of the user into a formal query. AGFL is also used to generate keywords/descriptions characterizing a particular information object (document). After matching the derived formal query against the (pre)generated characterisations of documents the relevant documents are presented to the user. The PROFILE project is a collaboration between the research institutes CSI (Computing Science Institute) and NICI (Nijmegen Institute for Cognition and Information).

The DORO (DOcument ROuting) project aims at producing a platform-independent system performing automatic classification and routing of human-readable documents in electronic form. The system will perform a shallow linguistic analysis of the contents of the documents using knowledge of the characteristics of the possible addressees. This analysis phase is performed using a parser generated with the AGFL system. The DORO project is an EC project with partners in the commerical and research communities.

In the joint research project DocuMate between the CSI and Edmond Research & Development B.V. the generation of personalized documents is researched. The problems being addressed in the DocuMate project lie in the specification of constraints governing the layout of printed documents and the presentation of that specification. Personalized documents are used frequently for marketing purposes in so called *mailings*. In a mailing, potential customers receive a letter or an advertisement which has been personalized using available information on the customer such as name, age, gender, interests, etc. Designing personalized documents is no trivial task, because the designer is simultanuously specifying thousands of (slightly) different documents instead of just one document.

The generic framework for Best First Search (BFS) introduced in this paper will be directly used by the AGFL and DocuMate project. The DORO and PROFILE project will also benefit from improvements in the AGFL project as they are both highly dependend on the speed and features of the AGFL system. It is for instance expected that the ability to generate

efficient probabilistic parsers will aid significantly in the later development of the DORO and PROFILE project.

The generic framework for BFS we aim at will be defined in terms of the Stratified Recursive Backup model, since this model allows for a quick implementation of efficiently running BFS solutions. In this paper we show how this can be accomplished. The approach is illustrated with examples from the areas of Natural Language Parsing and Text Layouting.

2 Generic framework for Best First Search

The problem to solve is one of finding which transformation rules must be applied and in what order to transform an initial state into some goal state, based on quality metrics of the states. There may be more than one solution in which case the top n *best* solutions are to be found. One solution is considered better than another based on a *valuation* (or fitness) function. In principle *all* possible solutions should be examined unless it can be proven that applying a certain rule can never lead to one of the solutions being searched for. For a general introduction to search algorithms the reader is referred to [Knu75].

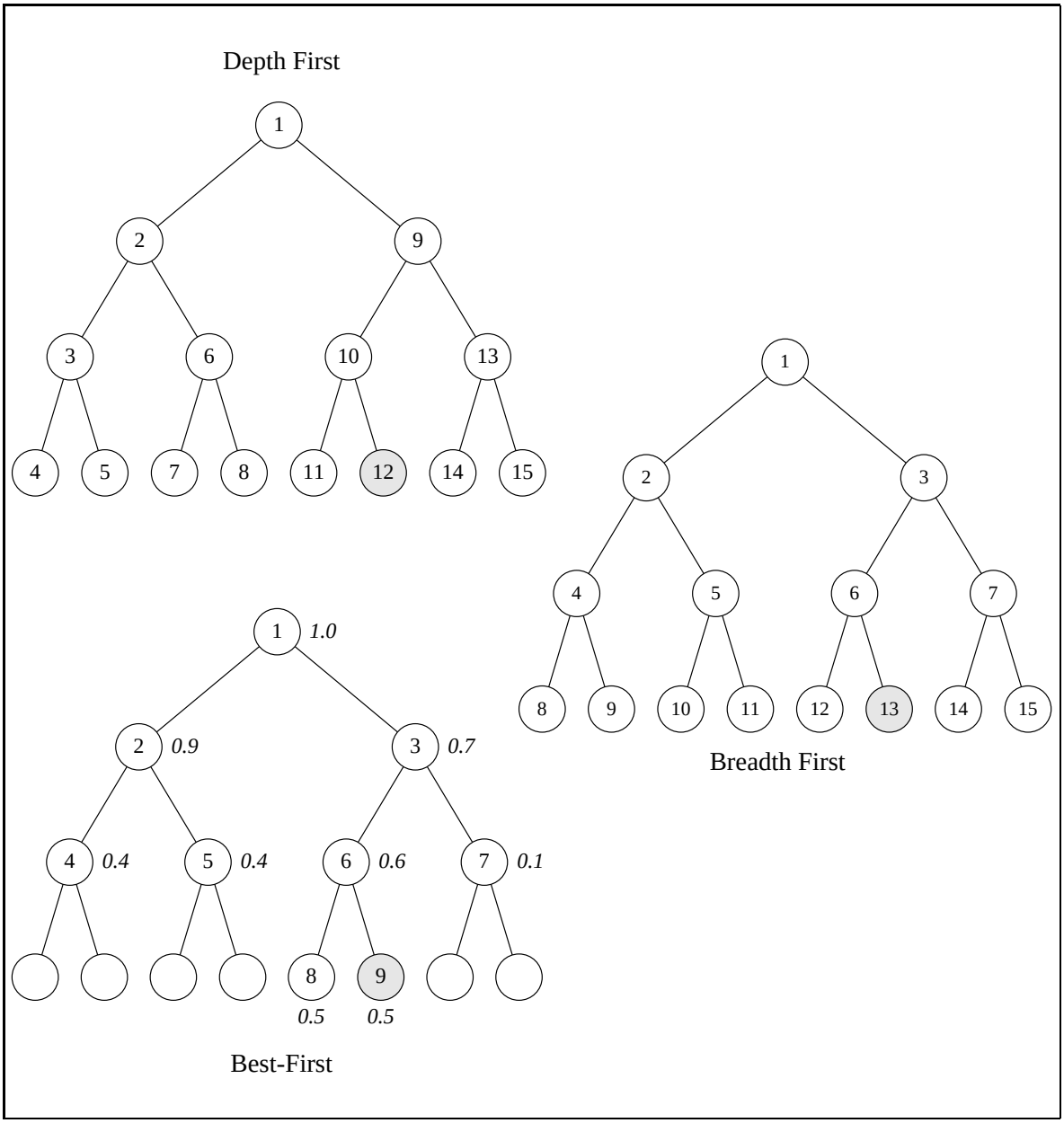

Figure 1: Traversing the search space

Figure 1 shows the differences between best first, depth first and breadth first search. The circles in this figure denote the different states. The number inside each circle shows the order in which the states were visited. The value next to a circle is the result of the evaluation function. A grey circle shows which state represents the best solution. The figure shows how best first search can reduce the time needed to find the best solution.

A generic framework is to be developed with which Best First Search problems can be implemented quickly and efficiently. The framework should be defined in terms of an abstract machine, such that machine specific issues can be abstracted. The abstract machine model should however be easily implemented in a language such as C, so that a wide range of hardware platforms can be supported. The framework should be able to handle very large state spaces. States should be stored economically because many partial solutions may be under consideration at any one time.

3 Stratified Recursive Backup

The Stratified Recursive Backup model is based on the Recursive Backup machine model described in [Kos74] and more formally presented in [Mey92]. Continuing in the style of Meyer, denotational semantics will be used to describe the construction and the ideas behind the framework. A more detailed description will be given using Update Schemes (see [Osb95, Osb92, Mey92] providing a solid basis for direct implementation.

3.1 Recursive Backup

The Recursive Backup machine model can be seen as an elegant and efficient implemention of *Top Down Backtracking*. In stead of using data structures with pointers, Recursive Backup uses continuations to keep track of the state space. Another property of the model is that for every action (operation) in the machine model there exists an *inverse* of that action known as the *undo-action*. Rather than storing the state in full, it recomputes parts of it on backtrack (using the inverse actions) trading space for time. In [Mey92] this model is derived starting from a straightforward specification of backtracking. Here we will build on the more advanced model.

The first step in developing a general framework is to define the semantics of the primitives. The second step consists of defining how the primitives can be combined. As a last step an example is used to show how a problem can be defined in terms of the framework. The framework is described using denotational semantics, from which an efficient and correct machine model can be obtained by transformation.

3.1.1 Domains

Various domains are used within the framework. The domains *state* and *result* are not defined here as they are part of the application domain. The other domains used within the framework do not assume anything more than their existence. In this model a tuple of *state*

and *results* called *environment* is used to represent the complete state. The domain *results* is used to maintain a list of results.

First a definition of the domains needed to model the *success* and *failure* continuations:

** $results ::= [] | result \succ results$
** $environment = state \times results$
** $failure = environment \rightarrow results$
** $success = failure \rightarrow failure$

A transformation rule is described in terms of a predicate (*pred*) checking for context conditions, a state transformer and a recording action. The following domains reflect this:

** $pred = state \rightarrow bool$
** $transform = state \rightarrow state$
** $record = result \rightarrow result$
** $rule = transform \times record$

The *action* domain is used to model the combination of rules and control flow.

** $action = success \rightarrow success$

3.1.2 Semantics

Using these domains the control flow of the recursive backup model can be described. Control flow is made explicit by using two continuations: the *success* continuation is used to represent future work and the $failure$ continuation is used to represent what must be done to undo previous work.

In the semantic description the variables are defined as follows:

** $env \in State, res \in results, cr \in result$
** $f \in failure, s \in success, r \in record, t \in transform, p \in pred$
** $e1, e2 \in action$

First the semantics of applying a single rule is given. Applying a rule consists of checking context conditions, computing a new state and updating the result. If the context conditions do not hold, backtracking is used to try another rule. The application of a single rule is therefore defined as:

** $DO \in pred \rightarrow rule \rightarrow action$
$DO\ p\ (t,r)\ s\ f\ (env, cr \succ res) = (p\ env)\ ?\ s\ (UNDO\ (t^{-1}, r^{-1})\ f)\ (t\ env, r\ cr \succ$

$res) : \ f \ (env, cr \succ res)$

** $UNDO \in rule \rightarrow failure \rightarrow failure$
$UNDO \ (t, r) \ f \ (env, cr \succ res) = f \ (t \ env, r \ cr \succ res)$

The auxiliary function $UNDO$ recomputes the environment that was passed to DO. Note that $UNDO$ uses the inverse of the transform and record functions which therefore *must* exist. If such an inverse does not exist, a stack can be used to store the information needed to undo the work of the transform or record function. The original function is then modified to save the required information on top of the stack, the inverse function can use the contents on the top of the stack to undo the effects of the original function.

The next step is to model the choice of the rule which is to be applied next. The action ALT is used to model choosing one rule from a set of *alternatives*. The action SEQ is used to enforce a *sequence* in which rules must be applied. Their definitions are:

** $ALT \in action \rightarrow action \rightarrow action$
$ALT \ e1 \ e2 \ s \ f \ (env, res) = e1 \ s \ (e2 \ s \ f) \ (env, res)$

** $SEQ \in action \rightarrow action \rightarrow action$
$SEQ \ e1 \ e2 \ s \ f \ (env, res) = e1 \ (e2 \ s \ f) \ f \ (env, res)$

Some states are considered as *end* states, i.e. states in which no rules can be applied. Within this model those states are classified as either a *success* or a *failure* state. A success state represents a solution, all other states represent dead ends. For this purpose special actions are introduced with which end states may be identified:

** $SUCC \in pred \rightarrow success$
$SUCC \ p \ f \ (env, cr \succ res) = (p \ env) \ ? \ f \ (env, cr \succ cr \succ res) \ : \ f \ (env, cr \succ res)$

** $FAIL \in failure$
$FAIL \ (env, res) = res$

Note that $SUCC$ saves a copy of the current result. It is the only place in the machine model where a datastructure is copied.

3.1.3 Example: Top-Down recognition

Using the above model the problem of recognizing whether a sequence of symbols (the input) is part of the language described by a particular context free grammar without left recursion can be defined. To keep the semantics as simple as possible, the construction of parse trees is

omitted, instead the number of parse trees is counted. Extending the description to actually construct parse trees is straightforward.

First the application specific domains modeling the input and the output are defined. The input is modelled as a list of symbols. The state simply represents the list of symbols that needs to be recognized. Because the framework already keeps a list of results, the *result* domain can be a constant.

** $state ::= [] | symbol \succ state$
** $result ::= \mathbf{ok}$

The grammar of the language is described by the domains *grammar* and *expr*:

** $grammar = \mathcal{P}(\mathbf{define}\ nonterminal\ expr)$
** $expr ::= \mathbf{empty} | \mathbf{fail} | expr\mathbf{;}expr | expr\mathbf{,}expr | nonterminal | \text{"}symbol\text{"}$

The choice of the symbols **,** for sequential composition and **;** for alternative composition follow usual Prolog conventions. The constant **empty** is used to denote the empty alternative and the constant **fail** denotes explicit failure.

The semantics given below describe how the problem of Top-Down recognition may be specified in terms of our framework. In this description the variables are defined as follows:

** $G \in grammar$
** $C \in symbol$
** $S \in nonterminal$
** $A, B \in expr$

The semantics are described in terms of a mapping from the grammar onto the recursive backup framework:

** $M[\_,\_] \in grammar \rightarrow nonterminal \rightarrow state \rightarrow results$
$M[G,S]\ i = E[G,S]\ (SUCC\ (EMPTY))\ (FAIL)\ (i, \mathbf{ok} \succ [])$

** $E[\_] \in script \rightarrow expr \rightarrow action$
$E[G,S] = E[A]\ ,\ (\mathbf{define}\ S\ A) \in G$
$E[G,(A\mathbf{;}B)] = ALT\ (E[G,A])\ (E[G,B])$
$E[G,(A\mathbf{,}B)] = SEQ\ (E[G,A])\ (E[G,B])$
$E[G,\text{"}C\text{"}] = DO\ (INPUT\ C)\ (POP\ C, ID)$
$E[G,\mathbf{empty}] = DO\ (TRUE)\ (SKIP, ID)$
$E[G,\mathbf{fail}] = DO\ (FALSE)\ (SKIP, ID)$

The description above uses a set of *predicates* to check for various context conditions which are defined below:

```
** INPUT ∈ symbol → pred
INPUT C (C' ≻ env) = (C = C')
INPUT C [] = false

** EMPTY ∈ pred
EMPTY env = (env = [])

** TRUE ∈ pred
TRUE env = true

** FALSE ∈ pred
FALSE env = false
```

The mapping also uses some transformers to implement state changes. The following *transformers* are used:

```
** POP ∈ symbol → transform
POP C (C ≻ env) = env

** SKIP ∈ transform
SKIP env = env
```

Finally the output is built incrementally using recorders. In this example only an identity function is used as a recorder because the underlying framework already keeps track of multiple results.

```
** ID ∈ record
ID cr = cr
```

In the above specification, given a grammar G, a root nonterminal S and input i, $|M[G,S]\ i|$ should count the number of ways in which the nonterminal S can be rewritten into i according to the grammar G.

3.2 Directing the computation

The stratified recursive backup model is an extension of the recursive backup model presented above. The models differ in the order in which solutions are generated. As discussed earlier, the stratified recursive backup model uses an explicit evaluation function with which computation can be directed towards generating the best solutions first (hence the name best first search). The evaluation function must be monotonic if branch and bound techniques are to be used. More formally, the following must hold for the evaluation function:

$$\forall_{a,b \in state} : b \subseteq a \rightarrow f(b) \geq f(a)$$

where $b \subseteq a$ means that partial solution b has been constructed from a partial solution a.

In the stratified recursive backup model a rule therefore consists of four parts: a predicate, a recorder, a transformer and an evaluation function. The evaluation function should be of type:

$$** \ eval = state \rightarrow real \rightarrow real$$

The idea is to maintain a value representing an evaluation of the current state. The evaluation function uses this value and the current state to compute a new value representing the evaluation of the new state. This leads to the following definitions for *rule* and *environment*:

$$** \ environment = state \times real \times results$$
$$** \ rule = transform \times eval \times record$$

In the stratified recursive backup model several partial solutions are considered in parallel. However at any one time only one partial solution (the best one) is being extended. After an extension has taken place another partial solution will be the best and attention will be switched to that partial solution. Therefore a sorted list of partial solution is maintained, which is sorted on decreasing values of the evaluation function. This datastructure is defined as:

$$** \ solution = success \times failure \times environment$$
$$** \ solutions ::= [] | solution \succ solutions$$

Note that the success and failure continuations are stored as part of the solution.

$$** \ best \in solutions \rightarrow environment \rightarrow success \rightarrow failure \rightarrow$$
$$ID \ \ cr \ = \ cr$$

Extra partial solutions can only be introduced by the *ALT* action as it allows several rules to be applied on one and the same state. This would require the environment to be copied whenever more than one rule is applicable, which might be very time and space consuming, especially if the environment is large. There are two obvious ways around this: recomputing environments (as in Recursive Backup) 'and sharing (parts of) environments. Recomputing environments could become very time consuming especially when switching between two solutions that have allmost completly different environments. Because switching between working on one partial solution to working on another partial solution is expected to happen ofteni, recomputing environments is probably not a good idea.

If (parts of) environments are to be shared, a sophisticated memory manager is needed to know which parts of the environment can be updated and which parts are shared. Within the framework an optimized reference counting memory manager will be used. The main drawback of using a reference counting memory manager is that it cannot manage circular datastructures. However datastructures with back-pointers such as (double) linked lists *can* be managed by the memory manager as a special case. If arbitrary circular datastructures are needed an extra memory manager can be used to reclaim memory used by circular datastructures at regular intervals.

3.3 Natural Language Parsing

There is a growing demand, coming from natural language based applications like information filtering, full-text information retrieval and translation, for fast parsers for natural languages. Even though Context-Free grammars may not be powerful enough for the adequate syntactic description of natural languages and are usually extended with features to obtain more expressive power, practical parsing methods are generally based on, or start from, Context Free parsing.

The AGFL (Affix Grammars over Finite Lattices) system, developed at the university of Nijmegen, uses a context free grammar extended with a second level of affixes as a language specification. From this specification the AGFL system constructs a top down recursive backup parser which can interface with a lexicon system. The lexicon system is used to transform the text input into a more abstract form called the lexical network. During the lexicalization process, words are looked up in a database and replaced by the wordclasses (such as: preposition, verb, etc.) they belong to. These wordclasses may have affixes describing properties of the word (such as: singular, plural, etc.).

Detailed information on the AGFL project can be found in [Kos91] or at the WWW site `http://www.cs.kun.nl/agfl`. A good overview of similar systems can be found in the *Natural Language Software Registry* (see the WWW site `http://www.dfki.de/lt/registry`).

In natural language parsing, ambiguity is one of the bigger problems. An ambiguous sentence can be read (parsed) in several ways, an example being: "time flies" which can mean that time seems to pass faster during activities or it can can mean that some aspect of the animal must be measured in time. One way to address this problem is to fix the meaning of words beforehand, based on the surrounding words. Usually this is done by some probabilistic tagging technique.

In the AGFL project the decision has been made to work with multiple parses (meanings) of a sentence and to present the parses in decreasing order of probability. In the AGFL project, probability factors will be used to take into account that words are usually used one way and to take into account that certain ways of constructing sentences are less common than others. This is done in AGFL by allowing probability annotations in the language specification.

Example 3.1.3 explains how parsing can be implemented with the recursive backup machinemodel. In this simplified example ambiguity is already handled by allowing multiple solutions to be found. For natural language parsing affixes need to be added. This poses no problem: affixes can be stored in the state and the constraints imposed by affixes can be

checked for in the predicates that guard the application of a rule. In fact this model is what is being used internally in the current AGFL system.

In order to be able to use best first search an evaluation function is needed. In AGFL probability factors, which are part of the language specification, can be used to construct an evaluation function which evaluates a particular state by simply multiplying the probability factors of the rules that have been applied sofar. This evaluation function is monotonic and can be computed incrementally. As part of this project the AGFL system will be re-implemented based on the stratified recursive backup model.

3.4 Text Layouting

Text layouting poses the problem of determining the placement of text and illustrations on pages, given a set of constraints. These constraints are introduced by the structure of the document (paragraphs, sections, footnotes, etc.) and by style considerations (margins, font size, figures, etc.).

In the field of typography much research has gone into determining how text should be formatted for optimum readability. After studing the human reading process, rules of thumb have been defined on subjects such as the placement of individual letters within a word, the spacing between words, the spacing between lines, etc. (see [Rub88]). It has also been discovered that sometimes irritating optical effects can occur due to akward formatting of text. A well known example of such an optical effect are the 'rivers of whitespace' that occur when whitespace in two adjacent lines occur at roughly the same (horizontal) position.

The problem of formatting text to fit in a specified amount of space has been studied in the literature and is known as *line breaking*. The algorithms used range from simple algorithms to highly sophisticated algorithms such as the algorithm used by the TEX typesetting system (see [Knu84a, Knu84b, KP81]). Most algorithms use some heuristic method to find a 'good' way to format the text quickly. Heuristics are used because the number of ways in which a piece of text can be formatted is exponentional in the number of words to be formatted. Trying all these posibilities would take too long for any piece of text longer than a dozen lines.

Documents are usually formatted more or less interactively, i.e. the document designer lets the layouting tool format the document and manually adjusts the the formatting of the document by changing the wording or using explicit line breaks. This method of composing a document is acceptable as long as the number of documents that needs to be produced is small. A problem arises when personalized documents are needed.

Personalized documents are often used for marketing purposes: rather than sending the same document to all (potential) customers, each customer is sent a document with a 'personal' touch. Information on the customer is used to generate tailor-made documents. This can be as simple as using the customers name and address in the document. More complex situations arise when parts of the document are included or left out on a per-customer basis. The interactive method of designing documents can not be directly applied to the design of personalized documents, as it would require manual inspection and modification of each of the generated documents. Inspecting and manually adjusting more than a few hundred

documents is impractical. Trying to inspect and adjust a hundred thousand personalized documents (an average size mailing) is not an option. Therefore a new approach to designing personalized documents is needed.

Designing personalized documents involves the simultanuous specification of thousands of slightly different documents. This poses two problems: a presentation and a specification problem. Instead of specifying the layout of one document the document designer must specify the layout of all the personalized documents at once. A design tool should be able to work with an example (to show to the designer) and an underlying specification of how the documents should be formatted. Modifications to the example should be automatically translated into modifications to the underlying specification. The design tool should also provide a way to directly modify the specification and to see the effects on the example being shown. The designer needs to be able to look at several examples to verify that the layout specification gives the desired results.

This problem of layouting text can be formulated as a best-first search problem. It is not particulary difficult to map the problem onto our framework. The input consists of the objects to be placed within the document. The state records the position of each object within the document. The layout specification is used to generate rules that calculate a position for an object based on absolute requirements (such as physical page dimensions, unprintable areas, etc.). The other constraints in the specification can be used to evaluate how well the document has been formatted. Criteria for the evaluation can be: even distribution of white space; the avoidance of unwanted optical effects (such as the rivers of white space); closeness of adherence to style considerations imposed by the layout specification.

Due to the nature of the framework the evaluation function is expected to evaluate roughly the same number of possible layouts that would be tried using heuristics. The advantage of our approach is that the optimal solution is never overlooked. However, finding the optimal solution may take an exponentional amount of time. In practice this problem can be solved by limiting the amount of time spend on a document. If the optimal solution could not be found within this time the document is either put aside or the best solution found sofar is used. After the layouting proces those documents that were put aside can be processed by a human operator.

The DocuMate project, a research project in collaboration between the CSI (Computing Science Institute) and Edmond Research & Development B.V., tries to tackle the text layout problem for personalized documents. The project has two goals:

- the first goal is to design a specification language for personalized documents. The specification language should provide primitives to deal with unprintable areas, floating objects, fixed objects, text in different fonts and sizes, chapters, sections, headers and footers, paragraphs and grouping constructs (to keep objects together on the same page).
- The second goal is to implement a prototype text formatter based on the specification language. Using the prototype a comparison with Microsoft Word, Wordperfect and LaTeXwill be conducted in order to evaluate the performance of the prototype.

The input for each of the different textformatters must be generated from a master template

to eliminate the possibility of handtuning the documents. Using a medium sized database (at least 20000 entries) several mailings will be run through each text formatter. The output of the different text formatters will be compared based on the number of hyphenations inserted, the variance of the interword and interparagraph spacing and the number of lines of text used.

4 Conclusions

Many seemingly unrelated problems can be described as *best first* search problems. In the first place, natural language parsing can be viewed as a best first search problem. Furthermore, in the DocuMate project the problem of generating and formatting personalized documents will be treated as a Best First Search problem.

In this paper a generic framework has been discussed which should minimize the amount of work involved in the implementation of best first search problems. Experience with the AGFL system has shown that the approach can be very successfull and we hope to see positive results for the DocuMate project within the next six months.

5 Planning

```
**************************************************************************
*                                  PLANNING                              *
**************************************************************************

- Senter subside tot 01/03/99
- Overeenkomst CSI-Edmond tot 01/10/99

02/07/97 - 06/10/97
Ontwikkelen SRB framework
Opzetten prototype DocuMate
        -> Eerste prototype DocuMate

06/10/97 - 05/01/98
Implementatie prototype AGFL/SRB
        Schrijven: specificatie framework
        -> Technisch rapport/artikel specificatie framework

05/01/98 - 02/03/98
Ontwikkeling evaluatie functie DocuMate
Implementatie prototype AGFL/SRB
        -> Prototype probabilistisch AGFL

02/03/98 - 01/06/98
Testen evaluatie functie DocuMate
```

```
        Vergelijkend waren onderzoek DocuMate, Word, WordPerfect en LaTex
        Schrijven: ervaringen met best first textlayouting
        -> Technisch rapport/artikel best first textlayouting
        -> Prototype DocuMate

01/06/98 - 02/09/98
Analyzeren optimalisatie mogelijkheden binnen framework
Schrijven: ervaringen met probabilistisch AGFL
        -> Technisch rapport/artikel probabilistisch AGFL

02/09/98 - 04/01/99
Analyzeren optimalisatie mogelijkheden binnen framework
Implementeren optimalisaties van het framework
        Testen optimalisaties aan prototype DocuMate
        -> Final Prototype DocuMate

04/01/99 - 05/04/99
Schrijven: optimalisaties aan het framework
        Testen optimalisaties aan prototype probabilistisch AGFL
        -> Technisch rapport/artikel optimalisaties framework

05/04/99 - 01/10/99
Combineren technisch rapporten/artikelen tot proefschrift
-> Proefschrift
```

References

[Knu75] D.E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, Massachusetts, 1975.

[Knu84a] D.E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984.

[Knu84b] Donald E. Knuth. *Computers & Typesetting*. Addison-Wesley, 1984.

[Kos74] C.H.A. Koster. A technique for parsing ambiguous grammars. *LNCS*, 26, 1974.

[Kos91] C.H.A. Koster. Affix Grammars for natural languages. In *Attribute Grammars, Applications and Systems, International Summer School SAGA*, volume 545 of *Lecture Notes in Computer Science*, pages 469–484. Springer-Verlag, Berlin, Germany, June 1991.

[KP81] Donald E. Knuth and Michael F. Plass. Breaking Paragraphs into Lines . *Software Practice & Experience*, 11:1119–1184, 1981.

[Mey92] E. Meyer. *Calculating Compilers*. PhD thesis, University of Nijmegen, The Netherlands, 1992.

[Osb92] H.R. Osborne. Update Plans. *Proceedings of 25th Hawaii International Conference on System Sciences*, pages 488–496, 1992.

[Osb95] H.R. Osborne. *Update Plans.* PhD thesis, University of Nijmegen, The Netherlands, 1995.

[Rub88] Richard Rubinstein. *Digital Typography.* Addison-Wesley, 1988.

[WBHW97] B.C.M. Wondergem, P. van Bommel, T.W.C. Huibers, and Th. van der Weide. Towards an Agent-Based Retrieval Engine. In J. Furner and D.J. Harper, editors, *Proceedings of the 19th BCS-IRSG Colloquium on IR research*, Aberdeen, Scotland, April 1997. Electronically available from: http://www.cs.kun.nl/∼bernd/.