Applying CLP to machine translation: A Greek case study

O. N. Nikolaenkova

1 St Petersburg State University, 7–9 Universitetskaya Emb., Saint Petersburg 199034, Russia

Abstract. In this research we concentrate on how CSP techniques can contribute to computational linguistics. For this purpose, we tested CP techniques on a linguistic problem of polysemy in machine translation. Ambiguity arises almost every time computers try to deal with human language. In information retrieval, the computer often provides information about alternative meanings of the search terms, meanings that we have no interest in. In machine translation, different meanings of an English word may be expressed by very different words in the target language, so it is important to determine which meaning of the English word the author had in mind and this is where a computer fails. Over and over, attempts to use computers to process human language have been frustrated by the computer’s limited ability to deal with polysemy (Miller 2001).

A toy system has been designed for translating between English and Greek. It is based on 6 sentences that, using the same lexicon, can be expanded to slightly more phrases. The translation is based on the results of two parsers — the English and the Greek one. Parsing uses a set of constraints (both semantic and grammatical). The main objective of this research is to evaluate Backtracking as one of CSP algorithms for machine translation. We used PROLOG to write an application because it is the only programming language that supports backtracking and where the choice made by the program can be traced back at every stage.

Keywords: CLP, Greek, ambiguity treatment, machine translation, constraint programming, ProLog.

Introduction

Constraint Satisfaction Problems (CSPs) have been a subject of research in Artificial Intelligence for many years. The very first works on networks of constraints dealt mainly with picture processing (Waltz 1975; Montanari 1974). During the 1960s–70s considerable progress in constraint-based reasoning was made within the field of AI research. Many powerful algorithms were designed to become the basis of current constraint satisfaction algorithms.

A Constraint Satisfaction Problem (CSP) consists of:
• a set of variables \( X = \{x_1, \ldots, x_n\} \),
• for each variable \( x_i \), a finite set \( D_i \) of possible values (its domain),
• and a set of constraints restricting the values that the variables can simultaneously take.

A solution to a CSP is an assignment of a value from its domain to every variable, in such a way that every constraint is satisfied (Russell, Norvig 2002, 137).

We may want to find:
• just one solution, with no preference as to which one,
• all solutions,
• an optimal, or at least a good solution, given some objective function defined in terms of some or all of the variables; in this case we speak about Constraint Optimisation Problem (COP).

Solutions to a CSP can be found by searching systematically through possible assignments of values to a variable. Search methods divide into two broad classes, those that traverse the space of partial solutions (or partial value assignments), and those that explore the space of complete value assignments (to all variables) stochastically.

Most problems that constraint programming (CP) deals with are those conventional programming finds hardest. With unconstrained search, such problems are time-consuming, which
increases exponentially with the problem size. Constraint programming has a big potential in solving real-life problems in a natural and effective way. Talking about CP as a natural way of solving CSP we should say that the search strategies that are used in CP are based on the search strategies the humans use (Bartak 2001). In speech processing we assign some semantic values to what we just heard, however, in some circumstances these semantic values can become illegal due to what comes next. Each element in the speech chain is connected to the previous one but also to the next one, the more elements we have, the more information about the context we get, the better semantic assignment we make. Context in this situation can be represented by a constraint network that defines which values are legal and which are not. In other words, representing a problem as CSP makes the formulation simpler, the solution easier to understand, and the choice of good heuristics to guide the solution strategy more straightforward. From the programming point of view, although CSP algorithms are essentially very simple, they can sometimes find a solution more quickly than integer programming methods, the program is not so heavy and needs less coding.

Thus, CP techniques represent the most natural way of solving such linguistic tasks as:

• speech processing of all kinds,
• speech synthesis,
• language processing,
• machine translation.

But CP techniques are also widely used in other fields of human activity, including:

• timetabling,
• workforce management,
• planning and scheduling,
• sequencing,
• resource allocation,
• analysis and synthesis of analogue and digital circuits,
• option trading analysis,
• cutting stock,
• network configuration.

Many algorithms for solving CSP search systematically through possible assignments of values to variables. These algorithms are very powerful for they always find a solution if it exists or prove that the task is impossible to solve. The main disadvantage of these algorithms is that they are time-consuming.

The most common algorithm for performing systematic search is backtracking. According to the backtracking algorithm, variables are instantiated sequentially and as soon as all the variables relevant to a constraint are instantiated, the validity of the constraint is checked. If a partial assignment violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available (Bartak 2001). Even if a solution is found and none of the assigned values violates the given constraints, the program backtracks in order to check if other possibilities of consistent assignments are possible — it exhausts all the possibilities of solving the given problem. Due to this, the backtracking algorithm considerably slows down the program:

• backtracking is performed chronologically, the reassignment is done step by step, at each value change a check of its consistency is performed,
• even if a right combination of values is found the program goes on with value reassignments and stops only when all combinations are checked.

The backtracking algorithm has three major disadvantages:

• thrashing, repeated failure due to the same reason. This occurs because what the backtracking algorithm does is checking different combinations of values according to a set of constraints, it does not detect the reason of the conflict caused by a certain assignment. Thrashing occurs because the standard backtracking algorithm does not identify the real
reason of the conflict, i. e., the conflicting variables. Therefore, a search in different parts of the space keeps failing for the same reason. Thrashing can be avoided by backjumping, sometimes called intelligent backtracking, i. e., when backtracking is performed on the variable that caused the failure.

- The other drawback of backtracking is redundant work. Even if the conflicting values of variables are identified during intelligent backtracking, they are not remembered for immediate detection of the same conflict in subsequent computation. The methods to resolve this problem are called backchecking or backmarking. Both algorithms are useful methods for reducing the number of compatibility checks. There is also a backtracking-based method that eliminates both of the above drawbacks of backtracking. This method is traditionally called dependency-directed backtracking and is used in truth maintenance systems. It should be noted that using advanced techniques adds other expenses to the algorithm that has to be balanced with the overall advantage of using them.

- Finally, the basic backtracking algorithm still detects the conflict too late as it is not able to detect the conflict before the conflict really occurs, i. e., after assigning the values to all the variables of the conflicting constraint. This drawback can be avoided by applying consistency techniques to forward check possible conflicts.

Applying CLP to Machine translation

Prolog Implementation

Our research team designed an application with an aim to translate sentences from English to Greek and vice versa. As we already mentioned in the introduction, our purpose was to test if backtracking algorithm could effectively solve the most frequent problem of machine translation — ambiguity. As our program is a test application, we decided to use a restricted lexicon to have full control over the application. The set of sentences used for these purposes consists of 6 basic sentences which can be expanded to a slightly larger number of phrases due to different combinations of words from the lexicon. Every English sentence contains a word that corresponds to two different Greek words depending on the context. Constraints have been applied to exclude cases where an ambiguous English word is translated incorrectly into Greek.

Table 1. The Greek and English sentences

| English Sentence                                      | Greek Sentence                                      |
|-------------------------------------------------------|-----------------------------------------------------|
| Zak saw the application running.                     | Ο Zak είδε την εφαρμογή να τρέχει.                   |
|                                                       | /o zak ide tin efarmogi na trekhi/                   |
| Zak saw the application for the accommodation.       | Ο Zak είδε την αίτηση για στέγαση.                   |
|                                                       | /o zak ide tin etisi gia stegasi/                    |
| Zak checked the case of the violin.                  | Ο Zak ελέγξε την θήκη του βιολιού.                   |
|                                                       | /o zak elenkse tin thiki tu violiu/                  |
| Zak checked the case of the noun.                    | Ο Zak ελέγξε την πτώση του ουσιαστικού.             |
|                                                       | /o zak elenkse tin ptosi tu usiastiku/               |
| Zak was at the bank to check the account.            | Ο Zak ήταν στην τράπεζα για να ελέγξει τον λογαριασμό. |
|                                                       | /o zak itan stin trapeza gia na elenksi ton logariasmo/ |
| Zak was at the bank to check the water.              | Ο Zak ήταν στην όχθη για να ελέγξει το νερό.           |
|                                                       | /o zak itan stin okthi gia na elenksi to nero/       |
The program

The parsers. The program consists of three parts: the English parser, the Greek parser and the translation block. Both parsers contain a set of syntactic rules that are common for both languages. Each rule corresponds to a sentence containing a noun phrase (np), a verb phrase (vp), a complement phrase (cp) or an adjective phrase (ac). These rules are further expanded to their components, a combination of determiners (det), nouns (n), verbs (v), prepositions (p), adjectives (a), or a combination of other phrases.

The Lexicon contains the words to be used for each language. Each word is denoted by its grammatical category: determiner (det), noun (n), preposition (p), verb (v), and gerund (g). Here we set the syntactic and grammatical constraints. Each Greek word, next to the grammatical category, has a number of variables: the interlingua representation, the syntactic and the semantic constraints. The syntactic constraints are used only in the Greek parser in order to avoid cases of nouns (male, female, or neutral at any case) having wrong determiners. The semantic representation is common for both parsers. The sentences have been categorized into six semantic fields: economics (constraint 1), nature (constraint 2), linguistics (constraint 3), container-content (constraint 4), documents (constraint 5), and computers (constraint 6). It should be mentioned that each clause begins with a ‘g’ if it refers to a clause for the Greek language and an ‘e’ to an English one, e.g. ‘gnp’ for a Greek noun phrase, ‘es’ for an English sentence.

The translation. Here two parsers work together. Each sentence is parsed and represented in an interlingua derived from the common rules for the sentence and the interlingua representation for each word. An interlingua is the output of the language to be translated and the input for the target language. The translation can be either from Greek to English ‘getr’ or from English to Greek ‘egtr’.

Building the program: Stages and Results of each stage

Stage 1. Parsers are a top priority for this project. The first parsers did not have any constraints. Their advantage is the precise syntactic representation of sentences. The backtracking algorithm checks every single rule and assigns all the valid values. However, there is a major drawback. As there are no constraints, the program generates a very big number of naive sentences. Thus, we lose computing performance and get a big number of invalid sentences. Table 2 shows a part of the program output.

Table 2. The first stage of the English parser

| Input | Syntactic rules | Lexicon |
|-------|-----------------|---------|
| es(s(Np,Vp))---> enp(Np),evp(Vp). | edet(the)-->(the). |
| es(s(Np,Vp,Pp))---> enp(Np),evp(Vp),epp(Pp). | edet(he)-->(he). |
| enp(np(det(Det),n(N))---> edet(Det),en(N). | edet()-->(). |
| epp(pp(p(P),Np))---> ep(P),enp(Np). | en(zak)-->(zak). |
| evp(vp(v(V),Np))---> ev(V),enp(Np). | en(bank)-->(bank). |
| en(water)-->(water). | en()-->(). |
| ev(checked) -->(checked). | ev(was)-->(was). |
| ev(at)-->(at). | ep(at)-->(at). |
A. Generating Sentences
| ?- es(X,X,()).

X =s(np(det(()),n(zak)),vp(v(checked),np(det(he),n(water)))) ,
Y = (zak,checked,he,water) ;

X =s(np(det(()),n(he)),vp(v(was),np(det(()),n(bank)))) ,
Y = (he,was,bank) ;

| ?- es(X,(zak,was,bank)),()).

X =s(np(det(()),n(zak)),vp(v(was),np(det(he),n(bank)))) ;

The syntactic trees of all the above sentences are satisfying but sentences like ‘Zak checked the water.’ and ‘He was bank.’ should be eliminated. For that reason, we designed a parser that takes care of the above mentioned constraints for both languages.

**Stage 2.** The benefits of representing the problem as Constraint Satisfaction Problem, are multiple. The program still generates many invalid sentences but they carry information that have to do with the constraints. So, when the translation starts, and this is a generation process, the constraints will be applied to the target sentence. During the translation the generator considers constraint information as well as a syntactic pattern from the input language that is processed to produce the output sentence. Furthermore, input sentences where the constraints conflict are not parsed (see Table 3, Output).

Table 3. Query concerning Parsing. One query fails and one succeeds

| Syntactic rules | Lexicon |
|-----------------|---------|
| es(s(Np,Vp,Cp))---> | edet(the,_)-->(the). |
| enp(Np,X),evp(Vp,Y),ecp(Cp,Y). | en(zak,z)-->(zak). |
| ecp(cn(p(P),Vp,Np),X)---> | en(bank,c1)-->(bank). |
| ep(P,X),evp(Vp,X),enp(Np,X). | en(bank,c2)-->(bank). |
| enp(np(det(Det),n(N)),X)---> | en(account,c1)-->(account). |
| edet(Det,X),en(N,X). | en(violin,c4)-->(violin). |
| evp(vp(v(V),Np,Pp),X)---> | ev(check,_)-->(check). |
| ev(V,X),enp(Np,X),epp(Pp,X). | ev(was,_)-->(was). |
| ep(at,_)-->(at). | ep(to,_)-->(to). |
| edet((),z)-->(.). | edet((),_)-->(.). |
| edet((),_)-->(.). | en((),_)-->(). |
Syntactic rules

A. Generating Sentences
Still problematic

B. Parsing sentences
| ?- es(X,(zak,was,at,the,bank,to,check,the,violin),()).
no

| ?- es(X,(zak,was,at,the,bank,to,check,the,account),()).
X=s(np(det()),n(zak)),vp(v(was),np(det()),n()),pp(p(at),np(det(the),n(bank))));

Output

At the second stage of building the program a number of constraints were applied to both English and Greek parsers. The syntactic constraints that were used for the Greek parser are as follows: ‘m’ for masculine in accusative case and singular number, ‘f’ for female in accusative and singular number, ‘n’ for neutral either in accusative or nominative (both cases have the same form) and singular number, ‘g’ for genitive of any gender (it is similar for all the genders) and singular number, ‘nom’ for neutral in nominative and singular, ‘pn’ to denote constraints for proper names.

The semantic constraints are common for both the languages. They are categorized into fields and then defined before parsing starts: economics ‘c1’, nature ‘c2’, linguistics ‘c3’, container-content ‘c4’, documents ‘c5’, and computers ‘c6’.

Backtracking tries to find all the possible syntactic trees when a sentence is parsed or generated. When the syntactic rules are many and include common parts (for example, all the rules have an ‘np’ or a ‘vp’) the program does not provide a possible solution in the output and errors like “Heap Space Full” occur. This problem arises because the program uses a large amount of space and before it crashes, it ends the procedure. Even a toy program has problems of this kind. In most cases, to save computational effort, it is better to put sentences in an order that does not create any problems with heap space.

Stage 3. At the final stage we created some rules that contribute to the translation of the aforementioned sentences from one language to the other. Now, the program has to combine the rules that are used for parsing in both Greek and English and go on with the translation. The below algorithm describes the procedure:

a. The program takes a sentence from the source language (the language that we want to translate from) as the input and tries to parse it according to the corresponding rule and constraints of the target language.

b. After the sentence is parsed, the program gets the structure tree of the source language and checks if there is a correspondent tree in the target language (the language that we want to translate into). This tree also has to consider the constraints.

c. If this occurs, then the program matches the words of the source language to the words of the target language according to the tree, the lexicon and the constraints.

d. The program gives translation in the target language in the output.
The Problems of the program

After the code was ready, we checked the efficacy of the program, which means we tried to test if it can translate ambiguous sentences from one language to the other, while considering the constraints. The program proved to be effective. However, since it is a toy program, it is obvious that in real world applications it has many flaws, e. g., a small lexicon or the lack of complex syntactic rules.

Table 4. Problems occurring when translating sentences

| Query | Translated Sentence |
|-------|---------------------|
| ![Image](image.png) | ![Diagram](diagram.png) |

Furthermore, due to the fact that backtracking tries to get as many solutions as possible, the program sometimes gives many more trees apart from the correct one. From these trees the first one is always correct and the rest emerge due to the order of syntactic rules in the code. See Table 3 for an example. This multiple tree generation problem also affects translation. We always get the right solution first but then other solutions are generated. These solutions are either the same sentence generated more than once or ungrammatical sentences.

Finally, heap space full, as it is described at the session “stage 2”, was another problem we faced when implementing this project.

Conclusions

As a result of our research work, we can conclude that Constraint Logic Programming (CLP) techniques are very helpful in solving linguistic tasks. They represent the most natural way of approaching and solving linguistic problems as these techniques resemble the way humans deal
with similar problems. Representing a linguistic problem using CLP makes the formulation simpler, the solution easier to understand, and the program less heavy. Moreover, it needs less coding.

The translating system that we created can parse and translate a big number of simple sentences and some complex ones. The output sentences are all syntactically correct due to the constraints used for showing the agreement between the article and the noun common for the Greek language. A system of constraints was also used to avoid semantically incorrect sentences, which also allows the ambiguity to be solved in an appropriate way.

However, due to the fact that in our implementation we chose a backtracking algorithm as the main search strategy we had the drawbacks typical for this algorithm:

After giving the right answer the program was going on with backtracking until it had tried all possible combinations of value reassignment. As a result, it worked quite slowly even with such a small dictionary. Besides, it was necessary to use a specific order of statements in the parsers blocks to avoid the program running out of memory because of enormous backracking.

Appendix

% English part

```
es(s(Np,Vp,Pp))--> enp(Np,X),evp(Vp,Y),epp(Pp,Y).
es(s(Np,Vp,Ap))--> enp(Np,X),evp(Vp,Y),eap(Ap,Y).
es(s(Np,Vp,Cp))--> enp(Np,X),evp(Vp,Y),ecp(Cp,Y).
es(s(Np,Vp))--> enp(Np,X),evp(Vp,Y).
ecp(cnp(p(P),Np),X)--> ep(P,X),enp(Np,X).
ecp(cnp(p(P),Vp,Np),X)--> ep(P,X),evp(Vp,X),enp(Np,X).
enp(np(det(Det),n(N)),X)--> edet(Det,X),en(N,X).
epp(pp(p(P),Np),X)--> ep(P,X),enp(Np,X).
evvp(vp(v(V),Np),X)--> ev(V,X),enp(Np,X).
evvp(vp(v(V),Np,Pp),X)--> ev(V,X),enp(Np,X),epp(Pp,X).
eap(ap(a(A)),X)-->(A).
edet(the,_)-->(the).
en(zak,pn)-->(zak).
en(bank,c1)-->(bank).
en(bank,c2)-->(bank).
en(account,c1)-->(account).
en(water,c2)-->(water).
en(case,c3)-->(case).
en(case,c4)-->(case).
en(noun,c3)-->(noun).
en(violin,c4)-->(violin).
en(accomodation,c5)-->(accomodation).
en(application,c5)-->(application).
en(application,c6)-->(application).
ev(check,_)-->(check).
ev(checked,_)-->(checked).
ev(was,_)-->(was).
ev(saw,_)-->(saw).
egf(running,c6)-->(running).
ep(at,_)-->(at).
ep(to,_)-->(to).
ep(of,_)-->(of).
ep(for,_)-->(for).
edet((,pn)-->(,).
edet((,c5)-->(,).
```
Applying CLP to machine translation: A Greek case study

gs(s(Np,Vp,Pp)) -> gnp(Np,_,X), gvp(Vp,_,Y), gpp(Pp,_,Y).
gs(s(Np,Vp,Ap)) -> gnp(Np,_,X), gvp(Vp,_,Y), gap(Ap,_,Y).
gs(s(Np,Vp,Cp)) -> gnp(Np,_,X), gvp(Vp,_,Y), gcp(Cp,_,Y).
gs(s(Np, Vp)) -> gnp(Np,_,X), gvp(Vp,_,Y).

gcp(cnp(p(P),Np),A,X) -> gp(P,A,X), gnp(Np,A,X).
gcp(cnp(p(P),Vp,Np),A,X) -> gp(P,A,X), gvp(Vp,A,X), gnp(Np,A,X).
gnp(np(det(Det),n(N)),A,X) -> gdet(Det,A,X), gnp(N,A,X).
gcp(pp(p(P),Np),A,X) -> gp(P,A,X), gnp(Np,A,X).
gvp(vp(v(V),Np),A,X) -> gvp(V,A,X), gnp(Np,A,X).
gvp(vp(v(V),Np,Pp),A,X) -> gvp(V,A,X), gnp(Np,A,X), gpp(Pp,A,X).
gap(ap(a(A)),B,X) -> gg(A,B,X).

gp(to,_) -> ('gia na').
gdet(the,m,_) -> (ton).
gdet(the,n,_) -> (ti).
gdet(the,g,_) -> (to).
gdet((),nom,pn) -> (o).
gdet((),_,_) -> ()

% Translating block

ep(A,X) :- es(A), es(X,A,()).
gp(B,X) :- gs(B), gs(X,B,()).

cetr(A,B,_) :- es(X,A,()),
gs(X,B,()).

getr(A,B,_) :- gs(X,B,()),
es(X,A,()).
Sources

Association for Logic Programming (community web). Available at: http://www.cwi.nl/projects/alp/ (accessed 15.06.2019). (In English)

Bartáček, R. (1998) On-line Guide to Constraint Programming. [Online]. Available at: http://kti.mff.cuni.cz/~bartak/constraints (accessed 15.06.2019). (In English)

Bartáček, R. On-line Guide to Prolog Programming (tutorial). [Online]. Available at: http://kti.mff.cuni.cz/~bartak/prolog/ (accessed 15.06.2019). (In English)

Blackburn P., Bos J., Striegnitz K. Learn Prolog Now! (tutorial). Available at: http://lpn.swi-prolog.org/lpnpage.php?pageid=online (accessed 19.07.2019). (In English)

Constraint Programming online (community web). Available at: http://www.cp-online.org/ (accessed 15.06.2019). (In English)

References

Bartáček, R. (2001) Constraint propagation and backtracking-based search. A brief introduction to mainstream techniques of constraint satisfaction. Prague: Charles University, 43 p. (In English)

Bratko, I. (2001) PROLOG programming for artificial intelligence. 3rd ed. Harlow: Addison-Wesley, 696 p. (In English)

Dechter, R., Frost, D. (1999) Backtracking algorithms for constraint satisfaction problems. Constraints, 48. [Online]. Available at: https://pdfs.semanticscholar.org/ad74/df16952abab23ac0f6e9f0af0e2133d39188.pdf (accessed 19.07.2019). (In English)

Freuder, E. C., Carchae, T., Beck, J. Ch. (2003) Satisfaction guaranteed. Proceedings of the IJCAI-2003 Workshop Configuration, 135 (1–2), pp. 1–6. (In English)

Miller, G. A. (2001) Ambiguous words. iMP Magazine. [Online]. Available at: https://www.kurzweilai.net/ambiguous-words (accessed 19.07.2019) (In English)

Montanari, U. (1974) Networks of constraints fundamental properties and applications to picture processing. Information Sciences, 7: 95–132. (In English)

Russell, S., Norvig, P. (2002) Artificial intelligence: A modern approach. 2nd ed. New Jersey: Prentice Hall, 1109 p. (In English)

Waltz, D. L. (1975) Understanding line drawings of scenes with shadows. In: Psychology of Computer Vision. New York: McGraw-Hill Publ., pp. 19–91. (In English)

White, Ch. M. (1995) Converting context-free grammars to constraint dependency grammars. A Master of Science thesis (Philology). West Lafayette, Purdue University, 88 p. (In English)