Logic as Vehicle for Real Life Applications

Panagiotis Stamatopoulos Isambo Karali

University of Athens, Department of Informatics
Panepistimiopolis, 157 71 Athens, Greece
E-mail: {takis,izambo}@di.uoa.gr

Abstract

Real life applications involve entities and algorithms that are complex to model and implement. Logic programming, the most practical approach of logic, facilitates the development procedure for these applications. However, its declarativeness does not seem sufficient to provide the means for the implementation of fully usable systems. In this paper, we discuss the applicability of logic programming in combinatorial search real life applications. We consider also some extensions of logic programming, such as constraints and parallelism. The addressed problems come from a variety of domains such as personalized tour construction, university course timetabling and airline crew scheduling.

1 Introduction

A large variety of real life applications involve the need of carrying out some kind of "intelligent action" [5]. The term "intelligent action" refers to a behavior that simulates the way humans act when they face difficult problems. A problem may be both difficult to model and difficult to solve. In most cases, an artificial system may act intelligently if it is able to *i)* access the required knowledge and *ii)* draw inferences on it. For the purposes of knowledge representation and deduction of new knowledge from existing pieces of knowledge, a tool that has been extensively used by the Artificial Intelligence (AI) community is logic.

In a logic-based framework, the world is modelled as a set of holding formulas which play the role of the axioms that support the proof of theorems. Actually, a theorem corresponds to a problem that has to be solved and the way the existing knowledge has to be combined in order to produce the desired result is nothing else but a search problem, at most cases a very difficult one.

In this paper, we discuss the use of logic for tackling real life applications that cope with a specific class of problems, the combinatorial search problems, with or without the requirement for optimization. More precisely, we are going to talk about first order logic and its specialization to a "computing machine", namely logic programming. The focus will be on the way this computational framework and some extensions of it are applied to the problems under consideration. Various real life applications are used to demonstrate the presented ideas, ranging from planning tourists' visits to interesting places and automatic timetable construction to airline crew scheduling. We comment on the introduction of constraints in logic programming and the exploitation of parallelism for speeding up search. Next, we give our assessment for the technology under discussion and, finally, we present our opinion about the usefulness of

developing intelligent heuristics to guide the search to more promising alternatives, or going to a direction of applying specialized algorithms hidden behind a declarative interface. Most of the results derive from our working experience in Esprit projects as well as internal research in the Department of Informatics of the University of Athens.

2 From Logic to Logic Programming

Logic finds its roots back to Aristotle's era and a lot of mathematicians, such as Frege, Skolem, Gödel, Herbrand, Tarski, Church and Turing, to mention some of them, starting from the last quarter of the 19th century, formalized it on a concrete basis. However, it was only during the 1970's that Alain Colmerauer and Robert Kowalski realized that a subset of logic might be used as a programming language appropriate for tackling problems in an entirely new way.

The story begins with the resolution principle, an elegant proof mechanism augmented with a unification capability, introduced by J.A. Robinson. Resolution was applicable to the so called first order logic for the purpose of deducing theorems from axioms. Unfortunately, this methodology was too general to be efficient. At early 70's though, Colmerauer and Kowalski independently concluded that by specializing first order logic to deal just with Horn clauses, i.e. implications that do not deduce disjunctive results, the emerging tool might be of significant importance.

The above gave rise to logic programming [8, 11], a problem solving approach that is still active, more than 20 years after its birth. Logic programming provides a declarative way of stating what the properties of a problem are but not how this should be solved. The latter is hidden into the procedural meaning of a program which is proved to be equivalent to its declarative meaning [14]. The most famous instantiation of the ideas of logic programming is the Prolog language. Despite the fact that in Prolog the equivalence between the declarative and the procedural semantics of a program is lost, many Prolog systems have been implemented and a lot of them have been extended in various promising ways. Nowadays, it is not uncommon for an application developer to commit to a Prolog-based approach for solving a real life problem.

3 Combinatorial Search Problems

There is a very broad class of problems which fall under the general areas of planning, scheduling and resource allocation and which are difficult to model but even more difficult to solve. The solution of such a problem consists of an appropriate assignment of values to the variables that model the problem's domain in such a way that various constraints are respected. These problems are often referred to as combinatorial search problems, in the sense that what we have to search for is a feasible combination of values for the incorporated variables.

In a combinatorial search problem, someone might look for one, some or all feasible solutions. Depending on the solution density of the search space, finding one or a few solutions might equally be a quite easy or an extremely difficult task. On the other hand, finding all feasible solutions might be out of the question, or even out of usefulness, in case there is a huge number of them. However, what is actually required in most cases is to find the optimal solution according to a given objective function. Then, we are talking about optimization problems, which is the kind of problems that the Operations Research (OR) people are attacking for many years now.

In the following, the presented ideas have emerged from the authors' involvement to the development of a variety of real life applications that faced combinatorial search problems. Among them, we mention:

Personalized tour construction: Given a set of available places with all possible activities that a tourist may visit as well as all relevant data (distances, costs, active periods, etc.), plan tours that fit better to specific tourist's interests and constraints.

University course timetabling: Given a set of university courses with their respective properties, schedule them in time and place in a way that does not violate any required constraint and satisfies all involved parties (teachers, students, etc.)

Airline crew scheduling: Given a set of flight legs that an airline company has to carry out, organize them into trips from home base to home base, called pairings, and allocate crew members to pairings in a way that respects all legal, contractual, union, etc. rules and optimizes some measure of schedule quality, such as fair assignment, satisfaction of preferences, seniority consideration, etc.

4 The Methodologies

In this section, we discuss how combinatorial search problems can be tackled in a logic-based environment and comment on the possibilities to achieve the desired results, especially when the overall application comes from and is going to work in the real world. Firstly, we consider a pure logic programming approach. Next, we deal with the introduction of constraints and, then, we augment it with the exploitation of parallelism as well. Finally, we comment on the employed tools and we conclude that significant effort has to be put for inventing intelligent heuristics that might guide the search to the most promising parts of the search space or, alternatively, enrich the high level logical approach with computation engines, that work behind the scenes, based on existing specialized algorithms.

4.1 Logic Programming

Logic programming provides a declarative way of modelling combinatorial search problems as well as a proof mechanism that might reach, at least theoretically, the desired solutions. The straightforward approach is to follow a *generate-and-test* methodology which consists of the exhaustive enumeration of all possible solutions and checking about their feasibility. Unfortunately, this procedure cannot behave efficiently, unless we are facing a "toy" problem or we are looking for a single solution in a search space very dense in solutions.

At least for prototyping purposes, somebody might start from a *generate-and-test* method or, perhaps, an improvement of it called *test-and-generate*. The latter approach is not that inefficient as the former is, since the feasibility tests are stated at the beginning and get suspended before the generation phase starts. In the generation phase, the appropriate tests are waken as soon as they can be checked during the construction of a candidate solution, which results in avoiding the complete generation of solutions whose part has been proved to be infeasible. However, neither of the two is sufficient to support the implementation of a real life working system.

4.2 Logic Programming + Constraints

The *test-and-generate* method described briefly in the previous paragraph led to the idea of a more active exploitation of the involved tests. A test, called constraint hereafter, may be used to prune inconsistent values of the involved variables, before getting to the point of choosing values for these variables. The effect of this pruning may be propagated then, through another constraint, to the possible values of other variables, leading in this way to a data-driven form of ensuring consistency. The overall result may be a significant reduction of the search space, depending, of course, on the nature of the involved constraints.

The above method, called *constrain-and-generate*, is based on the pioneering work of Pascal van Hentenryck [15], who designed, formalized and implemented the exploitation of constraints into logic programming, introducing, in this way, the Constraint Logic Programming (CLP) technology. Since then, a lot of CLP systems have been developed, most of which have been used for the implementation of applications dealing with combinatorial search problems. Unfortunately, this methodology has its limits as well and cannot give results when the problems tend to grow up in size [2].

4.3 Logic Programming + Constraints + Parallelism

An improvement of CLP that has been proposed and used in practice is to speed up the search by exploring alternative choices in parallel. There exist CLP systems that, instead of following a sequential depth-first traversal of the search tree with chronological backtracking, exploit parallel computing to examine simultaneously different values of a single variable. Depending on the nature of the problem, this approach may result up to linear speedup when looking for all solutions, or even to superlinear speedups, which, however, is a matter of good luck, when looking for one or the optimal solution.

Performing a parallel search is, certainly, better than a sequential approach, but even this does not solve the problem. When we face combinatorial search problems coming from the real world, the usage of constraints may prune significantly the search space and the more processing elements we have the more efficient is our search. However, even if we have at our disposal extremely powerful massively parallel machines, examining all the alternatives exhaustively is still out of the question, as the number of alternatives in a real life problem is incomparably greater than the number of processing elements used. So, are we hopeless? The answer is "No".

4.4 Intelligent Heuristics and Specialized Algorithms

When searching for the optimal solution of a combinatorial search problem, even a near optimal one might be satisfactory. In any case, the definition of the optimum is based on an objective function which may quantify some subjective criteria to some extent. Thus, it is not very easy to discriminate the quality of approximately equivalent solutions. So, we might be happy if we could guide the search by using problem dependent heuristics to satisfactory solutions, avoiding, in this way, computationally intensive enumerations. Certainly, finding out which heuristics to employ is a difficult task that might require extensive experimentation. In any case, this is something that people do in their every day life. We face combinatorial search problems and we manage to solve them without spending that much computing power, simply by taking the appropriate decisions at the right points. Actually, inventing and applying intelligent heuristic rules is an AI issue.

An alternative to the application of problem specific heuristics is to "steal" specialized methods from the OR area that have been applied successfully in the past for solving problems of the considered kind. The drawback of these methods might be that they are not very flexible and they do not contribute to a declarative formulation of the problem at hand. However, they can be coupled in an elegant way with a logic-based environment, by letting them work as constraint manipulating entities, while keeping the top-level framework highly declarative [9]. Then, the results might be very good both from the modelling and the efficiency points of view.

5 The Results

The methodologies of the previous section served as the implementation platforms for the development of the already mentioned applications.

As far as the personalized tour construction problem is concerned, it seemed that the parallel constraint logic programming technology was sufficient to cope with the encountered complexity [13], although, in some cases, the computation was driven to hardly acceptable efficiency [12]. This was due to the depth of the search space, the number of alternatives and the distribution of solutions.

The other two problems, i.e. the university course timetabling and the airline crew scheduling, were more complicated. The pure parallel CLP approach provided by the ECL^{i}PS^e language [3], which is also enriched with a built-in branch-and-bound method for tackling optimization problems, was followed for the timetable construction [4]. However, it was proved that the efficiency was very sensitive to the input data, thus, intelligent heuristic rules were employed as well for guiding the search, in order to build a usable system. The obtained results were absolutely satisfactory.

The crew scheduling problem was tackled both in a logic-based platform [10] and in an object-oriented one [6], i.e. a C++ library for parallel constraint programming, named Ilog Solver [7]. This library merges ideas coming from the logic programming area within an imperative programming approach, resulting to a possibility of doing more efficient searches. This was proved in practice, since for the crew scheduling problem, the logic approach might be used just for prototyping purposes, while the one that was based on Ilog Solver tended to be quite acceptable for a real world system. In addition, for the more difficult subproblems of crew scheduling, intelligent heuristics were applied as well, but, despite the very good results obtained, the conclusion was that something more is needed to attack the complexity. What is going to be applied for the rostering (assignment) subproblem of crew scheduling is to combine parallelism, constraint programming and OR methods with the aim to profit from all these technologies. This will be done in the context of the European Union Parrot Esprit project, which will start soon and where the University of Athens participates.

6 Conclusions

In this paper, we presented logic-based approaches for dealing with a special class of difficult problems, the combinatorial search problems. We explained why the declarativeness offered by logic is desirable for modelling such problems, but for solving them efficiently something more powerful than the blind application of resolution is needed. We shouldn't be afraid of coupling the advantages offered by logic with other technologies, provided we know what we are doing. As a hint of the direction that the researchers tend to follow in this area for the

future, someone might look to a very recent work of a key person of the logic programming community, Krzysztof Apt [1].

Acknowledgments

The authors would like to thank their colleagues in the EDS, APPLAUSE and PARACHUTE projects for all fruitful discussions they had for eight years on the topics covered in this paper and, especially, Prof. Constantin Halatsis who led the involvement of the group in these projects.

References

[1] K. R. Apt and A. Schaerf. Search and imperative programming. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 67–79, 1997.

[2] Y. Cras. Using constraint logic programming in services: A few short tales. In M. Bruynooghe, editor, *Proceedings of the 1994 International Symposium in Logic Programming*, pages 3–16, 1994.

[3] *ECL^{i}PS^e 3.5: User Manual*, February 1995.

[4] H. Frangouli, V. Harmandas, and P. Stamatopoulos. UTSE: Construction of optimum timetables for university courses — A CLP based approach. In *Proceedings of the Third International Conference on the Practical Application of Prolog*, pages 225–243, 1995.

[5] M. Ginsberg. *Essentials of Artificial Intelligence*. Morgan Kaufmann Publishers, 1993.

[6] C. Halatsis, P. Stamatopoulos, I. Karali, T. Bitsikas, G. Fessakis, C. Fouskakis, T. Schizas, and S. Sfakianakis. CREM integration and testing. PARACHUTE Project Deliverable DD39, University of Athens, 1996.

[7] *ILOG Solver: User's Manual — Version 3.2*, 1996.

[8] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[9] K. McAloon and C. Tretkoff. 2LP: Linear programming and logic programming. In V. Saraswat and P. van Hentenryck, editors, *Principles and Practice of Constraint Programming*, pages 101–116, 1993.

[10] C. Pavlopoulou, A. Gionis, P. Stamatopoulos, and C. Halatsis. Crew pairinig optimization based on CLP. In *Proceedings of the Practical Application of Constraint Technology Conference*, pages 191–210, 1996.

[11] J. A. Robinson. Logic and logic programming. *Communications of the ACM*, 35(3):40–65, 1992.

[12] P. Stamatopoulos and I. Karali. A tourist advisory system for Greece. In A. Herold, editor, *The Handbook of Parallel Constraint Logic Programming Applications*, chapter 7, pages 186–203. APPLAUSE Project Deliverable D.WP4.ECRC.4B, 1995.

[13] P. Stamatopoulos, I. Karali, and C. Halatsis. A tour advisory system using a logic programming approach. *Applied Computing Review*, 1(1):18–25, 1993.

[14] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[15] P. van Hentenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.