The 16^{th} International Conference on Principles and Practice of Constraint Programming (CP'10)

Doctoral Programme Proceedings

6 - 10 September 2010
St Andrews, UK

Welcome to the Proceedings of the 2010 Constraint Programming Doctoral Programme, held in conjunction with the 16th International Conference on Principles and Practice of Constraint Programming (CP'10) in St Andrews, UK. The doctoral programme is open to (mostly doctoral) students in all areas related to constraint programming. All participants present work either within the doctoral programme or in the main technical programme of the conference.

The papers in this proceedings are those which have been submitted directly to the doctoral programme. They contain a wide variety of work, either completed or in progress, being undertaken by the current generation of PhD students. We also list all students who have papers accepted into the main conference.

The Doctoral Program would not be possible without the support of many people and organisations. In particular, we would like to thank the program committee members and the sponsors of the CP conference. Moreover, we thank Google for funding the DP dinner.

We hope you have a great time in St Andrews!

Peter Nightingale and Standa Živný
Doctoral Programme Chairs, 2010

Programme Committee:

Sebastian Brand, NICTA and University of Melbourne
Hubie Chen, Universitat Pompeu Fabra
Chris Jefferson, University of St Andrews
Zeynep Kiziltan, University of Bologna
Michela Milano, University of Bologna
Justin Pearson, Uppsala University
Karen Petrie, University of Dundee
Claude-Guy Quimper, Google Inc.
Roland Yap, National University of Singapore
Neil Yorke-Smith, American University of Beirut and SRI International

Students with papers in the main CP programme:

Karl Sundequist Blomdahl, Uppsala University
Siddhartha Jain, Brown University
Mehdi Khiari, Universite de Caen Basse-Normandie
Lars Kotthoff, University of St Andrews
Anastasia Paparrizou, University of Western Macedonia
Justyna Petke, University of Oxford
Elaine Sonderegger, University of Connecticut
Justin Yip, Brown University

Table of Contents

Refining Portfolios of Constraint Models with Conjure

Ozgur Akgun, Ian Miguel, and Chris Jefferson
{ozgur,ianm,caj}@cs.st-andrews.ac.uk

School of Computer Science, University of St Andrews, St Andrews, Scotland, UK.

Abstract. Modelling is one of the key challenges in Constraint Programming (CP). There are many ways in which to model a given problem. The model chosen has a substantial effect on the solving efficiency. It is difficult to know what the best model is. To overcome this problem we take a portfolio approach: Given a high level specification of a combinatorial problem, we employ non-deterministic rewrite techniques to obtain a portfolio of constraint models. The specification language (Essence) does not require humans to make modelling decisions; therefore it helps us remove the modelling bottleneck.

1 Introduction

Many interesting real life problems can be formalised as constraint satisfaction problems (CSPs). A CSP consists of decision variables with associated domains, constraints on the assignments of values to a subset of decision variables and optionally an objective function. Solving a CSP is a well studied practice. There are many existing solvers, which employ advanced algorithms to reason about given constraints and run efficient search algorithms.

In order to solve a problem using a CSP solver, one needs to *model* the problem at hand. CSP solvers have different input languages - a common input language provides boolean and integer variables, arithmetic, logical and global constraints on these variables. CSP solvers provide a relatively high level of abstraction and expressivity when modelling a problem compared to MIP and SAT, and they still provide fast and scalable black-box solvers.

Most real world problems contain complex combinatorial structures such as sets, multi-sets, functions, relations, tuples, etc. Modelling a problem that can naturally be specified using these high level constructs is not a straightforward task - there are many ways to model a certain combinatorial object and a relation between a number of combinatorial objects. Moreover these alternative ways of modelling the same abstract expression do not dominate each other in terms of efficiency. Thus, given an abstract problem specification, building an efficient CSP model requires a great deal of expertise in CSP technologies and many experiments.

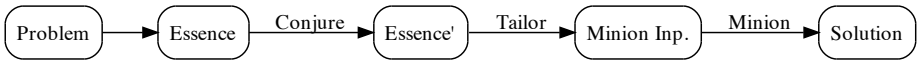
This work is an ongoing attempt to automate the CSP modelling process. In order to do so, we first design and implement a system to generate a selection

of valid CSP models given an abstract problem specification. The natural next step is to study relative strengths of generated models and design a system to select a good (if not the best) model. We will also explore the selection of a *portfolio* of models (as opposed to just selecting one) and running multi-model search techniques on this portfolio.

2 Tool chain

Our automated tool chain takes the approach of specifying the problem in an abstract constraint specification language, then compiling it to a low level model which current constraint solvers will accept. There are many decisions to be made at every step. These decisions have crucial impact on the actual time we spend on solving a given problem.

We employ different tools at different levels to best handle these tasks. Conjure takes Essence[1] specifications and generates a portfolio of Essence′ models. Tailor[2] takes Essence′ models as input and generates efficient Minion[3] input files. Conjure and Tailor have the capability to work at the problem class level, whereas Minion, the actual solver, works at the instance level.



3 Current status

The implementation of a working version of the refinement system Conjure is mostly complete. It is implemented as a non-deterministic term rewriting system. The current design and implementation of Conjure uses Haskell, provides an embedded implementation (EDSL) of the Essence language to be used by the rule authors, and an actual implementation of the language to be used by the problem owners. It successfully decouples the task of rewrite rule authoring from the implementation of the language and the actual process of applying the rewrite rules. The current rewrite rules database is a proof of concept demonstrating the fact that we can handle almost all of the language structures.

There exists a prototype implementation of Conjure, presented in [4], which refines a fragment of Essence limited to nested set-based decision variables into models in the Essence′ solver independent modelling language[1]

Current implementation successfully supports most of the Essence types, with minor limitations. For instance we currently do not handle function variables which map items from a nested combinatorial type to any other type. Function variables mapping integers to any type are fully supported though.

[1] Essence′ is, in turn, the input for the Tailor system [2], which transforms Essence′ models into input suitable for a particular constraint solver.

3.1 The Architecture of Conjure

This section gives an overview of the architecture of Conjure, which is a compiler-like system. Like most of the compilers, it has a pipeline which starts with parsing, validating the input, and type-checking. After these foundation phases, it has several phases for preparing the input specification for the rewriting phase, the actual rewriting phase, and some housekeeping phases. The pipeline is summarised below:

1. Parsing
2. Validating the input
3. Type checking the input
4. Representations phase
5. Auto-Channelling phase
6. Adding structural constraints
7. Expression rewriting
8. Fixing auxiliary and quantified variable names

Phases 1–3 are the foundation phases. The representations, auto-channelling, and adding structural constraint phases (4–6) prepare the input specification for the actual task of rewriting (Phase 7). Phase 8 can be viewed as housekeeping, it makes the output models easier to read and understand. Phase 7 (expression rewriting) is described in detail in the following sections. We will now give brief descriptions for the three preparatory phases preceding it.

Representation phase There are typically many ways to represent a combinatorial object. In this phase we make the representation decisions on the input specification in every possible way, and create multiple copies of it.

Auto-Channelling phase If we choose more than one way of representing a combinatorial object within a specification, we automatically add channelling constraints between different representations of the same variable at this phase. This way we link the different representations and make sure they represent the same combinatorial object.

Adding structural constraints At this phase we add all necessary structural constraints on every decision variable in the specification. The structural constraint for a representation of a decision variable makes sure the selected representation actually represents a valid combinatorial object with the intended properties. We add these constraints before rewriting take place, because they will be added regardless of the rest of the specification and they only depend on the representation of a combinatorial object.

3.2 Non-deterministic Rewriting

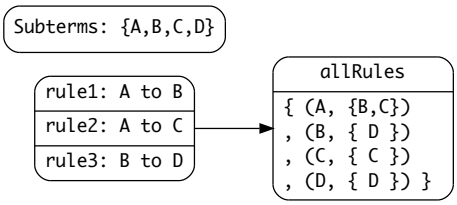
Our automated modelling system employs a term rewriting system to refine Essence specifications into the target language Essence′. Generally, rewrite rules can be thought of as partial functions, which map from a subterm to an *equivalent* subterm [5]. Given a set of rewrite rules and a term, a rewrite system

repeatedly applies the rules until no further rules can be applied. The term is then said to be in **normal form**.

As noted earlier, in order to produce alternative models we wish to generate not a single normal-form term, but all the normal-form terms that are attainable by applying the given rules to the input term. For this purpose we slightly adjust the definition of a rewrite rule: instead of a function that maps from a subterm to an equivalent subterm, we define a rewrite rule to be a function that maps from a subterm to a *set* of subterms.

Hence, a single rule in this definition is sufficient to represent the whole rule database. This representation is natural while applying the rules, but it is not a natural way to write them. It is, however, trivial to automate the combination of a set of partial functions into the single function used by the implementation.

For example we can combine `rule1`, `rule2` and `rule3` in `allRules` as follows:



Here `rule1`, `rule2` and `rule3` are partial functions. However the combined `allRules` is a total function, which maps from a subterm to a set of equivalent subterms.

`rule1` rewrites `A` into `B`, `rule2` rewrites `A` into `C` and `rule3` rewrites `B` into `D`. Since there is no rule matching `C` or `D` they are mapped to a singleton set of themselves.

In what follows, we will present our rules as partial mappings from single subterms to single subterms.

Figure 1 presents the elements of a rule: the mapping denoted by the $\rightsquigarrow$ operator; the guards that the left hand side of the mapping must satisfy; and the declarations to be used while constructing the right hand side of the mapping. Any expression that matches with the left hand side of the $\rightsquigarrow$ symbol is replaced by the right hand side, if all guards are satisfied.

Figure 2 shows an example rule that matches with a`subseteq` constraint between two sets of same types. It rewrites the constraint into a universal quantification over the first set. Can be read as *every element in set a, must also be an element of set b*. Notice also that it creates a quantified variable of type τ, which is the type of the elements of the two sets a and b. The actual name of the quantified variable is to be decided by the system.

```
essence_expression ⇝ equivalent_expression
    guards: properties that essence_expression must satisfy
    declarations: newly created variables and local aliases for expressions
```

Fig. 1. Anatomy of a refinement rule

```
a subseteq b ⇝ forall i : a . i elem b
    guards: a ∼ set of τ
            b ∼ set of τ
    declarations: i = quantifiedVar(τ)
```

Fig. 2. An example rewrite rule, `ruleSetSubsetEq`

It is useful to view our rules as operating upon an *Abstract Syntax Tree* (AST) representation of an ESSENCE specification. In the AST, every node represents a term in the specification and is also labelled with that term's type. To illustrate, Figure 3 presents a simple specification and its associated AST.

The root of the AST is the outer equality constraint, its immediate children are the decision variable x and the intersection operator, and so on. An identifier node, such as that associated with x, serves as a reference to the declaration of that identifier in the specification (the `find` statement in the case of x).

The rewriting system works by traversing the AST and attempting to apply the rules in the database at every node. A rule is allowed to modify the subtree rooted at the current node, and, for contextual information, is allowed to access (but not to modify) the remainder of the AST via the parent of the current node. If a rule matches the current node, the whole subtree is replaced with the equivalent subtree the rule suggests.

4 What's next?

Having a robust implementation of the automated refinement system, we are now one step closer to our ultimate goal, exploiting the opportunity of having multiple equivalent models for a given problem and eventually removing the *modelling bottleneck* from CSP to make it more accessible to wider audiences.

There is still a great necessity of improvements on the rules database. The quality and quantity of generated models directly depend on the quality and diversity of rules at hand.

```
given a,b : set of τ
find x    : set of τ

such that
    x = a intersect (b union c)
```

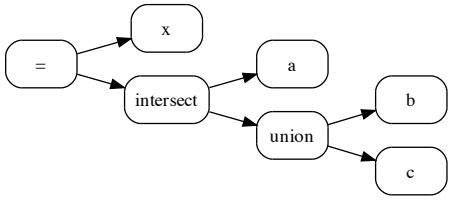


Fig. 3. A simple Essence specification and its AST view. Note that τ represents any concrete type.

Once we are confident about the models we generate, we will start studying effective model selection techniques. There are two basic stages where we can benefit from having multiple models, as described below.

Static exploitation of multiple models. Most constraint models describe a parameterised problem "class" (e.g. the class of sudoku puzzles). For input to a constraint solver, an instance of the class is obtained by giving values for the parameters (e.g. the pre-filled cells on the sudoku grid). We can exploit multiple models of a problem class by using small sized training instances to find the best-performing model, then we can use the best-performing model to solve other instances of the problem class. Although the search through the model space is initially uninformed, the system will learn which components of models tend to lead to better models and use this information to inform future model selection decisions.

Dynamic exploitation of multiple models. Constraint solvers typically employ a backtracking-style search combined with inference at each search node (constraint propagation) in order to find solutions. Since each search decision results in a new sub-problem that differs slightly from that associated with its parent node, our initial model selection might in fact be sub-optimal after a few decisions have been made. Hence, we can exploit multiple models dynamically by switching model mid-search. In order to do so, we must be confident that the new model will perform better than our current selection. The changing structure of the problem resulting from the decisions made by the constraint solver will provide the basis for this model selection, again employing a machine-learning methodology.

The result of these two approaches will be significantly enhanced performance of constraint solving, which will benefit a wide variety of industrial and academic users with combinatorial problems to solve. It will also remove the modelling bottleneck, in that it will no longer be necessary to have the expertise to select the "best" model.

References

1. Frisch, A.M., Grum, M., Jefferson, C., Hernández, B.M., Miguel, I.: The design of ESSENCE: A constraint language for specifying combinatorial problems. In Veloso, M.M., ed.: IJCAI. (2007) 80–87
2. Rendl, A.: Thesis: Effective compilation of constraint models. (2010)
3. Gent, I.P., Jefferson, C., Miguel, I. (In: ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings)
4. Frisch, A.M., Jefferson, C., Hernández, B.M., Miguel, I.: The rules of constraint modelling. In Kaelbling, L.P., Saffiotti, A., eds.: IJCAI, Professional Book Center (2005) 109–116
5. N. Dershowitz, J.-P. Jouannaud: Rewrite Systems. In: Handbook of Theoretical Computer Science. North-Holland (1990)

On the Power of Restarts for CSP

Luís Baptista[1,2] (student) and Francisco Azevedo[1] (supervisor)

[1]CENTRIA, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa, Portugal
[2]Instituto Politécnico de Portalegre, Portugal
lmtbaptista@gmail.com, fa@di.fct.unl.pt

Abstract. The use of restart techniques in solving Constraint Satisfaction Problems (CSPs) is considered of small importance for backtrack search algorithms. In this paper we propose to conduct a preliminary study on the impact of restarts in randomized backtrack search algorithms for solving CSPs. We show that the well-know n-queens problem has a heavy-tail distribution. We present empirical evidences that restarts can effectively improve the time to solve the n-queens problem. We implement a conflict-driven variable heuristic, and present empirical evidences that this heuristic effectively improve the time to solve the n-queens problem.

Keywords: search, constraint, restart, randomization, heuristic

1 Introduction

Constraint Satisfaction Problems (CSPs) are a well-known case of NP-complete problems [1]. They have extensive application in areas such as scheduling, configuration, timetabling, resources allocation, combinatorial mathematics, games and puzzles, and many other fields of computer science and engineering.

In this paper we propose to conduct a preliminary study on the impact of restarts in randomized backtrack search algorithms for solving CSPs. We also conduct a preliminary study on the impact of using knowledge from the past runs of the algorithm. We show that the runtime for computing the solution to the n-queens problem, using a randomized backtrack search algorithm, has a heavy-tail distribution. And we show that the use of restarts (restarting the algorithm) on a randomized backtrack search algorithm, with state-of-the-art techniques, improves the overall performance of backtrack search algorithm. And finally, adding a conflict-driven heuristic is shown to be better than the traditional, widely used, fail-first heuristic.

2 Constraint Satisfaction Problem

A Constraint Satisfaction Problem (CSP) consists of a set of variables, each with a domain of values, and a set of constraints on a subset of these variables. In this paper we will use a CSP with finite domains. A propositional satisfiability problem (SAT) is a particular case of a CSP where the variables are Boolean, and the constraints are defined by propositional logic expressed in conjunctive normal form.

Backtrack search algorithms are widely used for solving CSPs. It is commonly accepted that those algorithms should incorporate advanced search pruning techniques for space reduction, e.g., domain consistency techniques. Also, the use of heuristics based on the fail-first principle [2] is of great importance for efficiently solving CSPs. On the contrary, the use of restart techniques is considered of small importance for backtrack search algorithms. As a consequence they are not standard on state-of-the-art solvers.

The area of constraint satisfaction problems and the area of propositional satisfiability (SAT) share many techniques [3]. In SAT the use of restarts is a standard technique in state-of-the-art solvers. Restarts were essential in solving real-world instances of SAT [4, 5].

3 Restarts

A complete backtrack search algorithm is randomized by introducing a fixed amount of randomness in the branching heuristic [6]. The utilization of randomization results in different sub-trees being searched each time the search algorithm is restarted.

For many combinatorial problems different executions of a randomized backtrack search algorithms, on the same instance, can result in extremely different runtimes. This large variability in the runtime of the complete search procedures can be explained by the phenomena of heavy-tail distribution [6-8]. The heavy-tail distribution is characterized by long tails, as we can see in figure 1. The curve gives the cumulative fraction of successful runs as a function of the number of backtracks [8].

A randomized complete search algorithm is repeatedly run, each time limiting the maximum number of backtracks to a cutoff value. In practice a good cutoff values eliminates the heavy-tail phenomena, but unfortunately such a value has to be found empirically [6]. The resulting algorithm is not complete. A solution to this problem is to implement a policy for increasing the cutoff value [9]. A simple policy is to increment by a constant the cutoff value after each restart. The resulting algorithm is complete, and thus able to prove unsatisfiability [4].

As noted in [10] the impressive progress in SAT, unlike CSP, has been achieved using restarts and nogood recording [11, 12] (plus efficient lazy data structures). And this is starting stimulate the interest of the CSP community in restarts and nogood recording.

4 Experimental Results

In our empirical study we use the Comet System (http://www.comet-online.org), using the constraint programming solver over finite domains.

4.1 The N-Queens Problem

For our study of the impact of restarts we focus on instances of the well-known n-queens problem. This is a well studied problem [13], and has been used to illustrate various techniques used in solving CSPs [14]. When solving this problem, the use of domain consistence and fail-first heuristic are crucial.

We use the *alldifferent* global constraint to post all the constraint of the problem, where the variables $x_1, \ldots, x_n$, are the columns of the chessboard, and the domains are the possible rows. All queens are placed in:

- different rows: *alldifferent*($x_1, x_2, \ldots, x_n$)
- different downward diagonals: *alldifferent*($x_1+1, x_2+2, \ldots, x_n+n$)
- different upward diagonals: *alldifferent*($x_1-1, x_2-2, \ldots, x_n-n$)

4.2 Heavy tail distribution

The heavy-tail distribution in figure 1 was created with 872649 runs of a randomized backtrack search algorithm to solve the 8-queens problem. It selects the next variable to label and its value randomly.

Figure 1 shows that 50% of the runs solve the instance in 5400 backtracks (approximately) or less (the left part of the distribution). However, 1.2% of the runs do not result in a solution after 100000 backtracks (the right part of the distribution, the long tail).

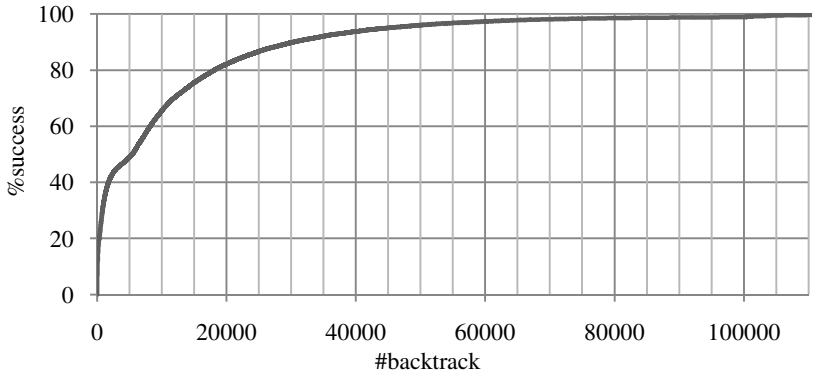


Fig. 1. 8-queens heavy-tail distribution

As we can observe, the 8-queens problem has a heavy-tail distribution. Using a randomized restart strategy, the long tails could be avoided [7]. So we will show that the use of restarts solves the n-queens problem more efficiently.

4.3 Restarts

In this session we use as a base configuration for all algorithms a backtrack search with constraint propagation and the fail-first heuristic. For the experimental results we use the following algorithms:

- **BK**, we are using the base configuration.
- **BK+Rand**, base configuration and randomly choose among the possible values for the variable.
- **BK+Rst**, base configuration, randomly choose among the variables with the three best values (according to the fail-first heuristic), and restart the search (the initial cutoff value is 1; the increment to the cutoff value after each restart is 10).

The results of running the algorithms are the number of backtracks needed to solve the instance. The backtrack limit for each instance was set to 500000. Since randomization was used in the last 2 algorithms, the number of runs was set to 10. Hence, the results shown correspond to the average values for all the runs.

Table 1. Results for different instances (with different number *n* of queens)

n	100	200	500	1000	1500	2000
BK	29	200217	(1)	2	4265	(1)
BK+Rand	58,8	35525,9	5791,8	103460,4 (2)	100310,2 (2)	50003,6 (1)
BK+Rst	84,2	63,1	352,7	846,2	2069,0	857,2

In table 1, the values in parenthesis specify the number of times the backtrack limit was reached (without solving the instance).

The **BK** configuration was used for reference. Occasionally the algorithm is lucky, as in the case of the 100-queens and the 1000-queens. This is because the runs are in the left most part of the heavy-tail distribution. In the other cases the algorithm has difficulties to solve the instances, or could not solve the instances. This is because the runs are in the right most part of the heavy-tail distribution.

The **BK+Rand** configuration can be seen as a non-deterministic version of the **BK** configuration. All the instances could be solved, but for some runs the algorithm could not solve the instance. Again this is because those runs are in the right most part of the heavy-tail distribution.

As we can observe, the results for the **BK+Rst** configuration uncover the power of using restarts:

- Restarts allow the algorithm to solve all the instances in all the runs.
- Restarts allow the algorithm to solve the harder (bigger) instances more efficiently (it needs fewer backtracks).

- Without restarts the algorithms can exhibit long run times, because of the heavy-tail distribution. But, as expected, with restart the algorithm avoid the long tail of the distribution.

4.4 Conflict-driven heuristic

A very important decision heuristic in SAT is based on clause recording (nogood recording) [5]. The general idea is to increment the value of literal involved in conflicts. The heuristic then select the variables involved in more conflicts.

We implement one counter for each variable. When the algorithm does not have more value to assign to a variable (a conflict) we increment that variable counter by 1. The variable heuristic select the variable involved in more conflicts, and break ties with the fail-first heuristic.

Table 2. Result for the conflict-driven heuristic

n	100	200	500	1000	1500	2000
BK+Rst	84,2	63,1	352,7	846,2	2069,0	857,2
BK+Rst+Conf	30,9	142,0	126,2	197,8	268,5	213,8

The label **BK+Rst+Conf** represent the backtrack search algorithm with restarts and with the conflict-driven heuristic. As we can see this heuristic improves, except in one case, the number of backtrack to solve the n-queens instances. So, in those instances this heuristic is better than the fail-first heuristic.

In [5] the counters are periodically divided by a constant, but in our implementation we do not divide the counters. This could be the reason why in some situations this heuristic behaves poorly. Nevertheless, this heuristic show promising results.

5 Conclusions

This paper gave a preliminary study on restarts applied in CSP. Restarts are not standard in CSP backtrack search algorithms, and are not considered useful in improving the algorithm. This paper contributes by contradicting the conventional wisdom. It shows that:

- Restarts can effectively improve the time to solve the n-queens problem.
- Conflict-driven variable heuristic can also effectively improve the time to solve the n-queens problem.

Restart do not substitute other techniques, it complements, empower other techniques, like constraint propagation and heuristic decisions. Restarts are an open area for CSP. Progress in SAT was due to restarts and nogoods [10].

In the near future we expect to:

- Confirm the results with other CSP instances (real-world and randomly generated).
- Include in the study nogood recording from conflict (learning).
- Study the impact of different cutoff and cutoff increment values.
- Enhance the conflict-driven heuristic.
- Test other forms of making the algorithm with restarts complete.

Acknowledgments. This work is supported by *Fundação para a Ciência e a Tecnologia* (SFRH/PROTEC/49859/2009).

References

1. Apt, K.R.: Principles of constraint programming, Cambridge University Press (2003).
2. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Proceedings of the 6th international joint conference on Artificial intelligence. pp. 356-364 Morgan Kaufmann Publishers Inc., Tokyo, Japan (1979).
3. Bordeaux, L., Hamadi, Y., Zhang, L.: Propositional Satisfiability and Constraint Programming: A comparative survey, ACM Comput. Surv., vol. 38, 2006, p. 12.
4. Baptista, L., Silva, J.P.M.: Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming. pp. 489-494 Springer-Verlag (2000).
5. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. Proceedings of the 38th annual Design Automation Conference. pp. 530-535 ACM, Las Vegas, Nevada, United States (2001).
6. Gomes, C.P., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. Proceedings of the fifteenth national conference on Artificial intelligence. pp. 431-437 American Association for Artificial Intelligence, Madison, Wisconsin, United States (1998).
7. Gomes, C., Selman, B., Crato, N.: Heavy-Tailed Distributions in Combinatorial Search, Principles and Practices of Constraint Programming, 1997, pp. 121-135.
8. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems, Journal of Automated Reasoning, vol. 24, 2000, pp. 67-100.
9. Walsh, T.: Search in a Small World. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. pp. 1172-1177 Morgan Kaufmann Publishers Inc. (1999).
10. Lecoutre, C.: Constraint Networks: Techniques and Algorithms, Wiley-ISTE (2009).
11. Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Nogood recording from restarts. Proceedings of the 20th international joint conference on Artifical intelligence. pp. 131-136 Morgan Kaufmann Publishers Inc., Hyderabad, India (2007).
12. Lecoutre, C., Saïs, L., Tabary, S., Vidal, V.: Recording and Minimizing Nogoods from Restarts, Journal on Satisfiability, Boolean Modeling and Computation, vol. 1, 2007, pp. 147–167.
13. Bell, J., Stevens, B.: A survey of known results and research areas for n-queens, Discrete Mathematics, vol. 309, Jan. 2009, pp. 1-31.
14. Rossi, F., Beek, P.V., Walsh, T.: Handbook of constraint programming, Elsevier (2006).

Facets for Alldifferent Systems

David Bergman
Supervisor: John N. Hooker - john@hooker.tepper.cmu.edu
Supervisor: Willem-Jan van Hoeve - vanhoeve@andrew.cmu.edu

Tepper School of Business, Carnegie Mellon University,
Pittsburgh PA 15213, USA
dbergman@andrew.cmu.edu

Abstract. We study the convex hull of integer points satisfying multiple overlapping alldifferent constraints. We describe various general results for the facial structure of the polytope and describe several classes of facet-defining inequalities for particular families of alldifferent systems. As there are exponentially many facets, we discuss how to separate the inequalities in polynomial time and present one such separation algorithm.

1 Introduction

The *alldifferent* constraint was introduced to constraint programming (CP) in 1978. The constraint requires that variables take pairwise different values. Thus if X is a set of variables $\{x_1, \ldots, x_n\}$, where each x_i has *domain* D_i, the constraint *alldifferent*(X) is satisfied by tuples $(d_1, \ldots, d_n)$ such that each $d_i \in D_i$ and $d_i \neq d_j$ for all i, j with $i \neq j$.

Although the single alldifferent constraint can be handled efficiently [13], many practical problems require multiple overlapping alldifferent constraints in their formulations; such systems are called *alldifferent systems*. Applications of alldifferent systems include parallel processing [11], course timetabling [14], register allocation [6], quasigroup completion and Latin square problems [3, 9],graph coloring problems and a number of scheduling problems.

An alldifferent system is defined for any family of variable sets $V_1, \ldots, V_q \subset X$ and variable domains. The system is equivalent to the conjunction of the constraints *alldifferent*(V_f) for $f = 1, \ldots, q$. In general, finding feasible solutions to alldifferent systems is NP-hard [7], and with the recent push toward integrating solution methods from CP and Integer Programming (IP), research has gone into polyhedral characterizations of alldifferent systems in order to create tight relaxations.

There are many instances where the benefit of integrating solution methods from CP and MIP is displayed. For example in [8], a CP algorithm with an assignment problem relaxation is used to solve timetabling problems two to fifty time faster than CP alone. Other uses of polyhedral characterization include using valid inequalities alongside CP filtering to prune search via bounds propagation (see [10] for other methods of integrating CP and MIP).

Our task in this paper is to contribute to the polyhedral analysis of alldifferent systems in which all the variable domains are equal, with the aim of constructing a tight continuous relaxation. One way to obtain a relaxation for an alldifferent system is to write a MIP formulation and investigate its polyhedral structure. However, the MIP formulation introduces 0-1 variables that generally do not occur in the original problem. Recent work has therefore focused on polyhedral analysis in the original variables [1, 2, 12], which opens the door to describe valid inequalities that are perhaps much different than those in the lifted system, and hence tightening the relaxation in both the 0-1 formulation and in the space of the original problem variables.

Relatively little is known about the polytope for a general alldifferent system. It is proved in [2] that if the system has a certain *inclusion* property, its polytope has no new facets beyond those of the individual alldifferent constraints. The same authors identify a small class of new facets for systems of three alldifferent predicates without the inclusion property. This class is extended somewhat to comb-like structures of alldifferent constraints in [12] and is extended here as well. In addition, since two alldifferent constraint always satisfy the aforementioned inclusion property, a careful examination of three alldifferent constraints done in [4] is a crucial step toward understanding more general and complex systems.

The remainder of the paper is organized as follows: first we describe a few general observations about the polytope. We then present several classes of facet defining inequalities for various families of alldifferent systems. Finally, we discuss how to separate these inequalities and present one such separation algorithm.

2 Preliminaries

For a given alldifferent system, let P_I be the convex hull of integral points satisfying the alldifferent system. In other words,

$$P_I = conv\{x \in \mathbb{R}^n : \forall i \in [n], x_i \in d(x_i) \text{ and } x_i, x_j \in V_f, \text{ some } f \in [q] \rightarrow x_i \neq x_j\}.$$

As in [1, 2, 12], we assume that all variables have the common domain set $D = \{0, 1, \ldots, k\}$.

In [10] P_I is described for the single alldifferent constraint and it is shown that the following inequalities provide a convex hull relaxation:

$$\frac{s(s-1)}{2} \leq \sum_{i \in S} x_i \leq ks - \frac{s(s-1)}{2}, \forall S \subseteq X, s = |S|$$

Using the above we can now define a linear relaxation for any alldifferent system. Since for all sets of variables which are common to one constraint, the inequalities above are valid, we have the following valid relaxation for any alldifferent system:

$$P_L = \{x \in \mathbb{R}^n : \forall f \in [q], \forall S \subseteq V_f, s = |S|, \frac{s(s-1)}{2} \leq \sum_{i \in S} x_i \leq ks - \frac{s(s-1)}{2}\}$$

These inequalities simply imply that for every subset of variables of size s in each constraint, their sum cannot exceed the sum of the largest s domain values and cannot be less than the sum of the smallest s domain values.

We will be working with this continuous relaxation and seeking facet defining inequalities of P_I.

3 General polyhedral results

In this section we present two general properties of the polytope. These properties hold for arbitrary alldifferent systems. The first result describes a certain symmetry and the second property provides a necessary condition for the existence of facet defining inequalities with positive and negative coefficients.

Complement inequalities Suppose we have a facet defining inequality $ax \geq \delta_1$ for some alldifferent system. If we let δ_2 be the maximum value of ax over all feasible x, consider the inequality $ax \leq \delta_2$. Clearly this is a valid inequality, but furthermore, if $ax \geq \delta_1$ is facet defining, then $ax \leq \delta_2$ is facet defining [5]. Hence, for the remainder of the paper, we discuss inequalities and separation algorithms only for inequalities of the form $ax \geq \delta_1$.

Large domain set If $|D| \geq 2\Delta + 2$ (e.g., $D = \{0, 1, \ldots, 2\Delta + 1\}$), then any facet defining inequality must have all coefficients of the same sign [5]. We see how this manifests itself in [4], as one of the classes of facets described there are valid if and only if the cardinality of the domain set is a particular value.

4 Inequalities

We now present the main results of the paper. We focus on path systems, providing both an example inequality and a separation algorithm. For the other systems we present only a few of the known inequalities and note that [5] will have a more thorough list of inequalities, along with detailed proofs and separation algorithms.

Paths An alldifferent system on $V_1, \ldots, V_q$ forms a *path* if $V_f \cap V_{f+1} = S_f$ is nonempty for $f = 1, \ldots, q-1$ and $V_f \cap V_g = \emptyset$ for all other pairs f, g. Let $x_a \in V_g - V_{g+1}$, $x_b \in V_h - V_{h-1}$, and let $x_f \in S_f$ for the subpath corresponding to $f = g, \ldots, h-1$. Then the inequality

$$2x_a + 2x_b + x_g + x_{g+1} + \ldots + x_{h-1} \geq \frac{h - g + 4}{2} \tag{1}$$

is facet defining if and only if the length $(h - g + 1)$ of the subpath is odd.

Example 1. Consider the following alldifferent system,

$$V_1 = \{x_1, x_2\}, V_2 = \{x_2, x_3\}, \ldots, V_q = \{x_q, x_{q+1}\}$$

$$D = \{0, 1, 2, 3, 4\},$$

with q odd and the point

$$x^0 = (0, 1, 1, 0, 1, 0, 1, 0, \ldots, 0, 1, 0).$$

x^0 is not feasible to the alldifferent system as $x_2^0 = x_3^0$. In addition, it is readily shown that x^0 does not violate any single alldifferent inequalities.

However, the following path inequality cuts this infeasible point:

$$2x_1 + x_2 + x_3 + \ldots + x_q + 2x_{q+1} \geq \frac{q+3}{2}.$$

If $q \geq 5$, there are other path inequalities that can be generated; for example,

$$2x_1 + x_2 + x_3 + x_4 + x_5 + 2x_6 \geq 4,$$

since the constraints covering these variables form a path of odd length.

Cycles An alldifferent system on $V_1, \ldots, V_q$ forms a *cycle* if $V_f \cap V_{f+1}$ is nonempty for $f = 1, \ldots, q-1$ as is $V_q \cap V_1$, but $V_f \cap V_g = \emptyset$ for all other pairs f, g. Let $S_f \subseteq V_f \cap V_{f+1}$ for $f = 1, \ldots, q-1$ and $S_q \subseteq V_q \cap V_1$, each of cardinality s. The following is facet defining:

$$\sum_{i \in S} x_i \geq \frac{q}{2} \cdot \frac{2s(2s-1)}{2}$$

General A family of sets $\{V_f\}_{f=1}^q$ forms an *intersecting* family if for $f = 1, 2, \ldots, q$ the sets Z, A_f and W_f are nonempty, where

$$Z = \bigcap_{f \in [q]} V_f, \qquad A_f = V_f \setminus \bigcup_{g \in [q] \setminus \{f\}} V_g, \qquad W_f = \bigcap_{g \in [q] \setminus \{f\}} V_g \setminus V_f$$

Z represents the intersection between all of the variables sets, A_f represents variables uniquely in each of the constraint sets V_f, and W_f represents variables in the intersection of all sets $V_g, g \neq f$ but not in V_f.

Let $x_f \in A_f$, $S_f \subseteq W_f$, $|S_f| = s$ for all $f = 1, 2, \ldots, q$ with $A = \bigcup_{i=1}^q \{x_i\}$, $S = \bigcup_{f \in [q]} S_f$, and $U \subseteq Z$ with $|U| = u$. The following are facet defining inequalities:

Inequality 1:

$$(qs + u) \sum_{i \in A} x_i + \frac{q(q-1)}{2} \sum_{i \in S \cup U} x_i \geq \frac{q(q-1)}{2} \frac{(qs+u)(qs+u+1)}{2}$$

Inequality 2:

$$qs\sum_{i\in A} x_i + \frac{q(q-1)}{2}\sum_{i\in S} x_i + q^2 s\sum_{i\in U} x_i \geq \delta$$

$$\delta = qs(qu) + \frac{q(q-1)}{2}\left[usq + \frac{sq(sq+1)}{2}\right] + q^2 s\frac{u(u-1)}{2}$$

5 Separation

All classes of facets described in this paper can be separated in polynomial time. The separation algorithms follow the same general framework as the separation algorithms presented in [2] and [12]. We present here a separation algorithm for the path inequalities. We generate only the lower bounding inequalities; the upper bounding inequalities can be separated similarly.

Let $\tilde{x}$ be the current solution of the relaxation.
For $g = 1, \ldots, q-2$
  For $h = g+2, \ldots, q$
    Let $\tilde{x}_a := \min_{i\in V_g \setminus V_{g+1}} \{\tilde{x}_i\}$, $\tilde{x}_b := \min_{i\in V_h \setminus V_{h-1}} \{\tilde{x}_i\}$
    For $f = g, g+1, \ldots, h-1$ let $\tilde{x}_f := \min_{i\in V_f \cap V_{f+1}} \{\tilde{x}_i\}$
    If $2(\tilde{x}_a + \tilde{x}_b) + \sum_{f=g}^{h-1} \tilde{x}_f < \frac{h-g+4}{2}$ then
      Generate the cut $2(x_a + x_b) + \sum_{f=g}^{h-1} x_f \geq \frac{h-g+4}{2}$

It can be shown that if some path inequality is violated by $\tilde{x}$ then the algorithm generates a separating cut in $O(q^2) + O(n)$ time.

6 Conclusion

In conclusion, we have presented additional facets for alldifferent systems, adding to the facets from [1, 2, 12]. All inequalities presented here can be separated in polynomial time. What remains to be resolved is whether or not these inequalities are *invariant* with respect to domain sets, in that for the same alldifferent system on domain set D_1 and D_2 there is a precise correspondence between the facets. In [5] it is shown that if there is an affine mapping between the two domain sets, then the polytopes are invariant, in that there exists a facet $ax \geq \delta_1$ for the system on domain set D_1 if and only if there is a facet $ax \geq \delta_2$ on domain set D_2. For arbitrary domain sets, the method described in [5] that allows facets to be generalized to arbitrary domain sets seems to generalize all facets described above.

References

1. G. Appa, D. Magos, and I. Mourtos. On the system of two all-different predicates. *Information Processing Letters*, 94:99–105, 2004.
2. G. Appa, D. Magos, and I. Mourtos. A polyhedral approach to the *alldifferent* system. manuscript, 2008.
3. G. Appa, D. Magos, I. Mourtos, and J. Janssen. On the orthogonal Latin square polytope. *Discrete Mathematics*, 306:171–187, 2006.
4. D. Bergman. Facets for three alldifferent constraints. Technical report, Tepper School of Business, Carnegie Mellon Univeristy, 2010.
5. D. Bergman and J.N. Hooker. Polyhedral results for alldifferent systems. in preparation.
6. G. J. Chaitin, M. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. Markstein. Register allocation via coloring. *Computing Languages*, 6:47–57, 1981.
7. K. Elbassioni, I. Katriel, M. Kutz, and M. Mahajan. Simultaneous matchings: Hardness and approximation. *Journal of Computer and System Sciences*, 74:884–897, 2008.
8. F. Focacci, A. Lodi, and M. Milano. Cost-based domain filtering. In J. Jaffar, editor, *Principles and Practice of Constraint Programming (CP 1999)*, volume 1713 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 1999.
9. C. P. Gomes and D. B. Shmoys. Completing quasigroups or Latin squares: A structured graph coloring problem. In *Proceedings of the Computational Symposium on Graph Coloring and Generalizations*, 2002.
10. J. N. Hooker. *Integrated Methods for Optimization*. Springer, 2007.
11. D. Kaznachey, A. Jagota, and S. Das. Neural network-based heuristic algorithms for hypergraph coloring problems with applications. *Journal of Parallel Distributed Computing*, 63:786–800, 2003.
12. S. Kruk, S. Toma, and M. Wallace. Some facets of multiple alldifferent predicate. In P. Belotti, editor, *Workshop on Bound Reduction Techniques for Constraint Programming and Mixed-Integer Nonlinear Programming, at CPAIOR*, 2009.
13. J.-C. Régin. A filtering algorithm for constraints of difference in CSP. In *National Conference on Artificial Intelligence (AAAI 1994)*, pages 362–367. AAAI Press, 1994.
14. A. Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13:87–127, 1999.

Energy-Efficient Task-Mapping for Data-Driven Sensor Network Macroprogramming Using Constraint Programming

Student: Farshid Hassani Bijarbooneh,
Advisors: Pierre Flener, Justin Pearson, and Edith Ngai

Department of Information Technology
Uppsala University, Box 337, SE – 751 05 Uppsala, Sweden
{farshid.hassani, pierre.flener, justin.pearson, edith.ngai}@it.uu.se

Abstract Using constraint programming (CP), we address the task-mapping problem in data-driven macroprogramming for wireless sensor networks (WSNs). A task graph representing the flow of data among tasks assists the application developer to specify the features of a WSN at a higher level of abstraction. A problem that arises in this context is how to map the tasks to nodes in the target network before the deployment of sensors, in order to achieve an energy efficient WSN. We take a published formulation of the task-mapping problem solved by mixed integer programming (MIP) solvers, and rewrite it as a constraint program. We minimise the maximum energy spent by each node for real-life instances of the problem, and show that our CP model results in significantly better runtimes than the MIP model.

1 Introduction

Wireless sensor networks (WSNs) operate as distributed systems where sensors cooperatively monitor or control a condition in an environment, such as temperature, speed, or pressure [1]. Nodes in a WSN consist of processor, radio transmitter, and a battery. It is of great concern to reduce the energy consumption at each node before deploying the network to the environment. The lifetime of a WSN depends critically on the energy consumption of the nodes, especially in cases where the battery cannot be charged once it is drained [4].

In a WSN, a *node* is assigned to perform several tasks. *Tasks* are pieces of code implementing applications of the network. The entire network repeats a same behavior over a time period called a *round*. Nodes have an initial energy level, which drops as the they communicate and process the assigned tasks [9].

Tasks are grouped in three categories: sensing tasks, operative tasks, and actuator tasks. *Sensing tasks* call a sensor to collect data at each round. For example, invoking a sensor to measure temperature in a room is performed by a sensing task. *Operative tasks* perform operations on data that has been gathered by the sensing tasks. Taking the temperature from different positioned sensors in a room and computing the average is an example of such a task. Finally, *actuator tasks* perform an action to affect the environment, which is based on

data processed by operative tasks. For example, consider a task that turns on a heater to warm the room. Sensing tasks and actuator tasks can only run on nodes with the appropriate sensors or actuators, while operative tasks are free to run on any node with sufficient computational resources. Programming a task for each sensor individually is a very time consuming process. We favour a methodology that allows defining and deploying tasks regardless of WSN architecture. In the WSN context, *data-driven macroprogramming* refers to an approach that facilitates sensor network programming by specifying the features of a WSN as a task graph representation [2,6]. Using this method, a task graph is created based on data flow that is independent of the network topology.

Data-driven macroprogramming poses many challenges including how to map the task graph efficiently onto the nodes in a WSN. A task mapping can be made more efficient by reducing the energy consumption. Note that task mapping is not only initiated at the deployment of the macroprogram, but also at any time during the operation of the WSN if the energy level of a node drops under a certain level.

In this paper, we optimise the task-mapping process in a multi-hop [1] heterogeneous [8] WSN to achieve overall minimum energy consumption by the sensors. A *multi-hop* WSN allows several nodes to forward data toward their destination, so nodes have to consider routing information. A *heterogeneous* WSN is a network that is able to provide several wireless services simultaneously.

For the mapping process, we consider task computation costs, data rates, node routing costs, and placement constraints with regard to a directed acyclic graph representing a data-driven task graph. Our task graph and the general modelling framework are the same as those in [2,10]. We take the framework and instance data related to two real-life applications of the problem: the monitoring of heating, ventilation, and air conditioning (HVAC) [3] and traffic management [7]. We implement our model as a constrained optimisation problem. Our approach follows the non-linear mathematical formulation of [10] and rewrites it into a constraint programming (CP) model. We show that our CP model achieves at least an order of magnitude speedup compared to the MIP model.

2 The Problem and Some Applications

We investigate two real-life applications. The first problem is a highway traffic management system where the aim is to reduce the congestion of vehicles on a highway by controlling speed limits and vehicle access to the highway. The speed sensor on each lane senses the speed of passing vehicles, a presence sensor indicates the presence of a vehicle on the ramp, and a red/green signal on the ramp controls vehicle entry to the highway.

The second application is building environment management for monitoring HVAC. This is similar to the traffic problem, but here the sensing tasks are sampling humidity and temperature, and the actuator control these aspects. The operative tasks collect the sampled data from sensors, compute averages, and

respond with a proper action for the actuator nodes. For each of the problems we experimented with two instances taken from [10].

3 Model

We present a constraint model of the task mapping problem by formulating it as a non-linear system of equations similar to the one of [10]. In our CP model, we model the non-linear equations with binary expressions and logical operators, and solve our model using *Gecode* [5].

3.1 The Decision Variables

Let t and n be the number of tasks and nodes respectively, and let $v_1, v_2, \ldots, v_t$ be an array of integer decision variables in the node domain $[1, \ldots, n]$, such that v_i denotes the node that is associated with task i. Also let $x = \{x_{ip} \mid 1 \leq i \leq t, 1 \leq p \leq n\}$ be a two-dimensional array of redundant 0/1 decision variables, where the value of x_{ip} is 1 if and only if node p is associated with task i ($v_i = p$). In the MIP model, the solver directly operates on the x variables, whereas in the CP model we branch on the v variables and maintain the relationship between v and x variables with channeling. Throughout this paper, the variables always have lowercase identifiers, and the constants have uppercase identifiers, the indices i, j always refer to a task, and the indices p, q, r always refer to a node.

As explained in Section 2, a sensing task or an actuator task may be limited to some nodes, and therefore it cannot be mapped to any other nodes. This limit is reflected in the model by dropping value p from the domain of variable v_i for every task i that cannot be mapped to node p (in other words, constraining x_{ip} to be 0 for all tasks i that cannot be mapped to node p). Let e_p be an array of integer decision variables in the domain $[0, \ldots, \infty]$, such that e_p is the total energy spent by node p according to the tasks mapped to it by the v_i variables.

3.2 The Problem Formulation

The total energy e_p spent by node p in one round consists of two terms: communication cost and computation cost. We can ignore the computation cost, since it has a far smaller value in our problems than the communication cost. The communication cost e_p for each node p is the sum of the costs of routing messages between every pair of nodes that are communicating:

$$e_p = \sum_{(i,j)\in A} F_i \cdot S_{ij} \cdot R_{v_i v_j p} \quad \forall p \ (1 \leq p \leq n) \tag{1}$$

where A is the set of arcs in the data-driven task graph, indicating if task i is sending data to task j. Task i is fired F_i times at each round, and S_{ij} is the size of the data between tasks. The energy consumed by node p while routing one unit of data from node q to node r is denoted by R_{qrp}. Intuitively e_p is the cost

for a particular mapping given by v and it can also be calculated by including all possible nodes v_i and v_j in (1) and multiplying by $x_{iq} \cdot x_{jr}$ to indicate whether there is a message routed from node q to node r via node p:

$$e_p = \sum_{(i,j)\in A} \sum_{q=1}^{n} \sum_{r=1}^{n} F_i \cdot S_{ij} \cdot R_{qrp} \cdot x_{iq} \cdot x_{jr} \quad \forall p \ (1 \leq p \leq n) \qquad (2)$$

The optimisation metric is defined by minimising the maximum fraction of energy spent by any node in the system at a round. This metric is defined regarding the fact that, in task-mapping for WSNs, we aim at maximising the time to reconfiguration (TTR), which is the time that the energy level of some node goes below a fraction of its initial energy E_p. By minimising the maximum fraction of energy spent by all nodes, the value of TTR will be maximised. Thus, the objective function to be minimised is the following:

$$\max_{1\leq p\leq n} \ \frac{1}{E_p} \cdot e_p \qquad (3)$$

3.3 The Constraints and Objective Function

Our CP model changes the non-linear part of (2) into a binary expression:

$$e_p = \sum_{(i,j)\in A} \sum_{q=1}^{n} \sum_{r=1}^{n} F_i \cdot S_{ij} \cdot R_{qrp} \cdot (x_{iq} \wedge x_{jr}) \quad \forall p \ (1 \leq p \leq n)$$

The conjunction constraint for binary operator $\wedge$ (logical and) is more efficient than the constraint *mult* in *Gecode*. This model does not require explicitly defining a variable to replace $x_{iq} \wedge x_{jr}$.

We let the cost c $(0 \leq c \leq 1)$ be a floating point decision variable that holds the maximum fraction of the initial energy E_p spent by a node p in the network at each round. To perform the minimisation of (3), we constrain the total energy e_p of each node p to be at most equal to the total cost c, subject to minimising c. This implies posting constraints $\frac{1}{E_p} \cdot e_p \leq c \quad \forall p \ (1 \leq p \leq n)$, but we can modify it to involve only integer variables. We assume that the initial energy E_p for all nodes is the same. Therefore, it does not affect the resulting total cost c in the model, so we can avoid the fraction, and extend the domain of c to $[0, \ldots, E_p]$. So the constraints to be posted are $e_p \leq c \quad \forall p \ (1 \leq p \leq n)$

Finally, we maintain the relationship between the solution decision variables v_i and the binary variables x_{ip} by *channelling* between them as follows, using the *channel* constraint of *Gecode*:

$$v_i = p \iff x_{ip} = 1 \quad \forall i, p \ (1 \leq i \leq t, 1 \leq p \leq n) \qquad (4)$$

We can implement the channelling (4) by either the *element* constraint, *reification* or *channel* of *Gecode*. We need to enforce constraints that each tasks is mapped to exactly one node: $\sum_{p=1}^{n} x_{ip} = 1 \quad \forall i \ (1 \leq i \leq t)$, which is implied in reification implementation. Reification performs faster than the *element* constraint both with and without the implied constraints, but it performs slower than the *channel* constraint. In this paper, we only present the results based on the channelling implementation of (4).

The search procedure branches on the v_i variables only. It uses the first-fail strategy for variable selection, and selects values greater than the mean of smallest and largest values in the domain of the selected variable.

	HVAC				Highway Traffic			
⟨nodes, tasks⟩	⟨7, 6⟩		⟨106, 80⟩		⟨74, 36⟩		⟨124, 60⟩	
time and cost	time	cost	time	cost	time	cost	time	cost
Gecode	0.01	10.00	0.73	12960	1.74	**300**	13.35	**300**
Gurobi	0.29	10.00	0.00	12960	44.00	**300**	212.00	**300**
SCIP	0.14	10.00	3.22	12960	3600.00	**300**	3600.00	**300**
lpsolve	0.04	10.00	3600.00	**12960**	3600.00	**320**	3600.00	**360**

Table 1. Optimisation results of different solvers for four instances of the HVAC and traffic management task mapping problems. The time unit is seconds. The boldface values indicate best cost achieved before one hour timeout.

4 Experiments

We experimented with two representative instances for the problems in Section 2. Our CP model is implemented in *Gecode* (revision 3.2.2) and runs under Mac OS X 10.6.3 64 bit on an Intel Core 2 Duo 2.53 GHz with 3MB L2 cache and 4GB RAM. We set a timeout of one hour for each instance, recording the best minimum cost achieved and the time at which it has been reached.

We also solved our instances using the MIP solvers *Gurobi* (revision 2.0.2),[1] *SCIP* (revision 1.2.0),[2] and *lpsolve* (revision 5.5)[3] under the same system configuration and experimental set-up. We chose *lpsolve* since it is the solver used in [10]. We also chose to experiment with *Gurobi* and *SCIP*, as these two solvers are among the fastest commercial and non-commercial MIP solvers respectively (according to the *SCIP* home page).

In Table 1 we present the results of our CP model compared to the model solved by the three MIP solvers. The boldface values indicate a best cost achieved before timeout, whereas the rest were optimally solved under the timeout limit. In the second HVAC instance, an MIP solver, namely *Gurobi*, is faster than *Gecode*, because *Gurobi* managed to solve the problem in one simplex iteration.

We observe that, in the rest of the instances, *Gecode* can reach the same solution about 90% faster than the fastest MIP solver we used, *Gurobi*. We are also always faster than the MIP approach in [10]. It is also worth mentioning that MIP solvers use a pre-solve phase to eliminate as many variables and constraints as possible before solving the actual problem, and pre-solve in the hardest instance (traffic ⟨124, 60⟩) is taking 22 seconds, which is included in the runtime in Table 1, and it is already longer than the total time spent by *Gecode* to find a first optimal solution.

[1] available from `http://gurobi.com`
[2] available from `http://scip.zib.de`
[3] available from `http://lpsolve.sourceforge.net/5.5`

5 Conclusion

Macroprogramming for WSNs is an evolving area, where efficiency of a WSN can benefit considerably by task mapping in a configuration phase, as well as in reconfiguration when the energy level of a node drops below a certain amount.

We presented a CP model for the task mapping problem in a multi-hop heterogeneous WSN. Our model involves placement constraints and the cost of routing. The optimisation is done by minimising the maximum energy spent by each node. We model the non-linear formulation effectively using binary expressions and show that it is competitive with the current MIP implementations. To the best of our knowledge, there has been no work addressing a CP approach for a general case of task mapping in a multi-hop WSN achieving energy optimisation under routing costs.

Acknowledgements. This research is sponsored by the Swedish Foundation for Strategic Research (SSF) under research grant RIT08-0065 for the project *ProFuN: A Programming Platform for Future Wireless Sensor Networks*. We thank Animesh Pathak of INRIA Paris-Rocquencourt (France) for useful discussions and datasets.

References

1. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.
2. A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner. The abstract task graph: A methodology for architecture-independent programming of networked sensor systems. In *EESR'05: Proceedings of the 2005 workshop on End-to-End, Sense-and-Respond systems, applications and services*, pages 19–24, 2005.
3. M. Demirbaş. Wireless sensor networks for monitoring of large public buildings. *Computer Networks*, 46:605–634, 2005.
4. S. C. Ergen and P. Varaiya. Energy efficient routing with delay guarantee for sensor networks. *Wirel. Netw.*, 13(5):679–690, 2007.
5. Gecode Team. Gecode: Generic constraint development environment, 2006. Available from `http://www.gecode.org`.
6. R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using kairos. In *Distributed Computing in Sensor Systems (DCOSS)*, pages 126–140, 2005.
7. T. T. Hsieh. Using sensor networks for highway and traffic applications. *Potentials, IEEE*, 23(2):13–16, April–May 2004.
8. D. Kumar, T. C. Aseri, and R. Patel. Energy efficient heterogeneous clustered scheme for wireless sensor. *Computer Communications*, 32(4):662–667, 2009.
9. A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA'02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM.
10. A. Pathak and V. K. Prasanna. Energy-efficient task mapping for data-driven sensor network macroprogramming. *IEEE Transactions on Computers*, 59:955–968, July 2010.

Mapping periodic applications on unary and cumulative resources

Alessio Bonfietti and Michela Milano

DEIS, University of Bologna

Abstract. Synchronous Data-Flow Graphs (SDFGs) are a very powerful modeling abstraction for high-end multi-core embedded system applications. They intrinsically represent concurrency and compactly express periodicity. Their optimal platform allocation and scheduling subject to cumulative resource and real-time constraints is strongly NP-hard. State of the art approaches mainly focus on incomplete algorithms. We have developed a complete constraint-based framework based on graph modifications as search decisions and two global constraints. The first is an efficient incremental version of the throughput constraint proposed in [1] the second is a cumulative resource constraint properly extending the Precedence constraint posting approach for periodic applications. An extensive experimental evaluation on realistic SDFGs shows the advantages of the proposed approach.

1 Introduction

Multi-core systems-on-chip (MPSoCs) [2] are becoming truly distributed systems at the micro-scale. They provide high performance and low power and have been applied in many embedded computing domains like wireless communication, imaging, audio and video processing, graphics. Applications in these areas feature significant functional parallelism, which can effectively be expressed though a data-flow model of computation

For this reason increased research effort is being focused on developing methods and tools for efficiently mapping data-flow applications onto many-core MPSoC platforms [3]. Synchronous data-flow (SDF, [4]) is one of the most widely used models, as it is sufficiently semantically rich to express **periodic applications**, while being still analyzable with reasonable efficiency.

As most of the data-flow applications are resource and real-time constrained, a key problem that must be addressed by SDF mapping tools is throughput-constrained, resource-consistent allocation and scheduling. This is unfortunately a NP-hard problem, and it is usually solved by sequential decomposition and incomplete search. Recent results have however demonstrated that this problem can be solved to optimality for non-trivial SDF graphs using advanced constraint-programming approaches [1].

In this work, we devise an allocation and scheduling framework for managing periodic applications, where decisions to be taken correspond to graph modifications, namely arcs and token additions; we propose an incremental filtering

algorithm for the throughput constraint that achieves one order of magnitude speed-up w.r.t. the one described in [1] and we present an extension to the Precedence Constraint Posting approach [5] for modeling cumulative resources in the context of periodic applications.

An extensive experimental evaluation on realistic SDFGs shows the advantages of the proposed approach.

2 SDFG

Synchronous Data-flow Graphs (SDFGs) [4] are used to model streaming applications with hard real-time constraints. They are very powerful modeling tools that enable the representation of both pipelined streaming and cyclic dependencies between actors. In SDFG, nodes represent tasks (*activities*), edges are data dependencies, i.e., (*precedence relations*) and are labeled with numbers (called *rates*) constraining the executions of nodes. Data are represented with tokens upon edges (also called *Delays*).

Actor execution is defined in terms of firings. In detail, whenever an actor fires it consumes a given and fixed amount of tokens from its input edges and produces a known and fixed amount of tokens on its output edges. These amounts are called *rates*. The rates determine how often actors have to fire w.r.t. each other such that the distribution of tokens over all edges is not changed. This periodicity property is captured by the repetition vector.

We define "iteration" of the graph a complete execution of all activities. As the graphs describe periodic applications, the number of execution iterations should be considered infinite. In real-time the performance of a periodic application is measured with the throughput index (further details in sec. 3.2).

The SDFG reported in figure 1 has three actors. Actor A has a dependency edge to itself with one token on it. It means that the two firings of A cannot be executed in parallel. Also, each time A executes it produces one token that can be consumed by B. Each time B executes it produces 3 tokens while C consumes 2 tokens. Also when C executes it produces 2 tokens while B requires 3 tokens to fire. Thus, every 2 executions of B correspond to 3 executions of C. This is captured in the repetition vector reported in figure 1.

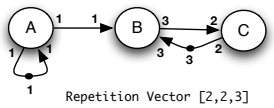


Fig. 1. A Synchronous Data-Flow Graph

SDFGs in which all rates equal 1 are called Homogeneous Synchronous Data Flow Graphs (HSDFGs,[4]). Every SDFG G can be converted to an equivalent HSDFG G', by using the conversion algorithm in [6], sec. 3.8. For each node in the SDFG we have a number of nodes in the HSDFG equal to the corresponding number in the repetition vector. Equivalence means that there exists a one-to-one mapping between the SDFG and HSDFG actor firings. As a consequence, the two graphs have the same throughput.

The research presented in this work adopts the Homogeneous version of the graph.

3 The Framework

The problem considered in this work is the allocation and scheduling of an HSDFG on a target set of unary and cumulative resources subject to throughput (i.e., real time) constraints.

Given a HSDFG labeled with actor durations, given a target platform in terms of cumulative resources, the problem is to assign each actor to a resource element and to define an ordering between actors allocated on the same resource such that the throughput constraint is satisfied and the execution is guaranteed to be conflict-free. Note that our approach can easily handle optimal throughput allocation and scheduling. This problem is strongly NP-HARD.

Allocation and scheduling problems are usually modeled in terms of variables ranging on temporal domains i.e., activity starting point and ending point (in case of variable duration). Synchronous Data-flow Graphs (SDFGs) represent periodic applications. For this reason, the representation using temporal variables and their relations is not suitable. We have therefore adopted an alternative approach where, instead of working on temporal variables, we consider graph modifications: the graph representing the application is modified, by constraint propagation and the search strategy, by adding arcs and tokens so as to tighten the search space. Therefore, the global constraints proposed works on the graph and modifies its structure.

3.1 The Scheduling model

For modeling the periodic scheduling problems, we adopt two sets of variables: (1) $Arc_{i,j}$ representing the existence of a directed arc between activity i and j ($i \rightarrow j$) and (2) $Tok_{i,j}$ representing the number of tokens over the arc (i, j).

A precedence constraint appearing in the initial graph, such as $E_i \leq S_j$, is modeled with $Arc_{i,j} = 1$ and $Tok_{i,j} = 0$. The presence of a token ($Tok_{i,j} \geq 1$) implies that the activity j could fire at the same time of i. If A is the set of activities, we say that a_i precedes a_j iff it exists a path without tokens that connects a_i with a_j ($a_i \prec a_j$).

Using this representation, **the decision of allocating two activities i and j on the same resource can be modelled by imposing two arcs between i and j, namely $Arc_{i,j} = 1$ and $Arc_{j,i} = 1$. A scheduling decision is instead modelled as adding a token on the arc between i and j if j precedes i and on the arc between j and i otherwise.**

If a precedence constraint between i and j exists without tokens in the original graph, this implies that a possible edge from j to i should have at least one token, otherwise the execution of the graph deadlocks.

The scheduling problem we face considers alternative discrete and cumulative resources. We introduce in the model a set of variables ($Res_{i,j}[0, 1]$) defining the allocation of the activity i on resource j.

We have developed a cumulative *resource* constraint whose purpose is to avoid conflicts over the resources by modifying the application graph. In this work we use the *precedence constraint posting* approach, proposed by [5] [8], since it well integrates with graph modifications concept. The PCP framework is indeed extended as graph modifications do not only refer to precedence constraint posting, but also to token addition.

The first extension to the PCP is needed as we handle applications with periodic behaviors and it is necessary to consider resource conflicts over multiple application executions. In the following, we detail how the extended PCP works and how we solve conflicts over multiple executions. In PCP, possible resource conflicts are resolved off-line by adding a fixed set of precedence constraints and tokens between the involved activities. The resulting augmented graph defines a conflict-free graph and if all activities are allocated, the augmented graph becomes the solution.

Let $A = \{a_0, a_1, a_2\}$ be the set of three activities not in relation with each other. The solver allocate them on the same unary resource and their execution create a conflict. A solution could be $a_0 \prec a_1 \prec a_2$. An intuitive way to map on a graph a precedence constraints is adding an edge between the actors. Simulating the execution we can notice that when the actor a_0 terminates its execution, it produces a token over the edge (a_0, a_1). At this point the activity a_1 can execute but, since the graph has a periodic behavior, a_0 can re-execute too. As the resource is unary, we have a *multi-iteration* conflict.

To avoid *multi-iteration* conflict we need to lock the re-execution of the first one only after the last one finishes; this is done adding a third edge with token: (a_2, a_0). The key is the positioning of the token which creates a *single-token* cycle. In *single-token* cycles, the actors execute sequentially. Considering cumulative resources, multiple activities could execute at the same time without conflicts. We extend the concept of *single-token* cycle to cumulative resources. The idea is that the activities finishing the execution of the iteration of the graph must be connected (with edges and tokens) with the ones that start the execution.

3.2 The Throughput

Since SDF graphs model periodic behaviors, real time constraints refer in general to the application throughput. From an intuitive point of view, this is best defined in terms of the throughput of the associated HSDFG; this refers to how often an activity fires and is inversely proportional to the weightest cycle.

Definition: Given a cyclic graph G, the *maximum-cycle-mean* is the maximum of the sum of the execution time of the activities over the number of tokens of the cycles in the graph. The throughput of the graph G is one over it's *maximum-cycle-mean*, that is the number of tokens over the sum of the execution times.

The **objective function** of the solver presented is to maximize the graph throughput.

We developed a throughput constraint whose purpose is to reduce the search space propagating over the graph by adding arcs and tokens, and computing the graph throughput. This constraint is based on the one proposed in [1] and

is notably extended for making it incremental. It encapsulates a filtering algorithm, originally proposed in [1]. However the algorithm turned out to be heavily resource-hungry. Its computation takes up to 90% of the search time, and the solver had a reduced scalability. Each time the graph is modified, the constraint receives a new description of the graph, and computes the throughput value over it. The solver reaches a feasible solution if and only if the throughput value found during the search (Upper Bound) is higher than the maximum between the value of the best solution found and the bound given by the problem instance (Lower Bound).

The filtering algorithm is based on a recursive formula which computes, starting from a source node, the weight of each path (execution times of the considered nodes) of the graph. As soon as a cycle is found, the throughput is computed. The final throughput value is the lowest found, that is the weightiest cycle.

4 Experimental Results

We have evaluated our model and the scalability of our code on various sets of realistic but synthetic instances. The graphs belong to two distinct classes of Synchronous Data-Flow Graph: cyclic and acyclic graphs. The class that better better fits our solution approach is the Cyclic one. Clearly, if a graph contains cycles, it has an implicit throughput upper bound defined by the longest cycle in the graph. In contrast, acyclic graphs have no implicit bound and expose the highest parallelism: this makes them the most challenging instances. The generated graph have been divided according to the number of nodes (from 10 nodes to 14 nodes). Table 3 presents median and maximum complete search times for cyclic (2nd and 3rd column) and acyclic (4th and 5th) istances. Times are presented in seconds with a time-limit of 900 seconds. As expected, the time to solve the most constrained iteration and the average running time grows exponentially with the size of the instances. However, the solution time is reasonable for realistic size graphs. Results are comparable with state of the art CP approaches, that in turn are much simpler w.r.t. our approach as they do not face periodicity and in some cases consider independent resources [10]. Moreover, the experiments show that the incremental filtering algorithm gains over one order of magnitude speed-up w.r.t. its non incremental version. The solver with the incremental algorithm runs from 2.174 to 5.579 times faster. As expected, the speed-up tends to increase with the dimension of the problem instance.

Node	% Non-Incr	% Incr
10	72.24	15.35
12	79.01	15.99
14	82.91	19.32

Fig. 2. Relative algorithms computation time

Node	CMedian	CMax	AMedian	AMax
10	0.12	118.84	2.58	303.44
11	0.96	186.48	14.48	472.77
12	2.67	446.39	30.15	675.56
13	8.33	647.14	75.08	759.32
14	18.64	825.14	189.43	834.32

Fig. 3. Search and Constraint execution times and speed-up

The problem faced is strongly NP-HARD and clearly the computational time grows up exponentially in the instance dimension. However, the reduced time for constraint computations in the incremental solver increases scalability and enables the solution of harder and larger problems. The table in figure 2 shows the relative amount of time that the algorithm computation absorbs during the search. It is evident that the new algorithm is definitely faster and lighter. Its impact on the search time is lower than 20% of the total time while the non-incremental version time stands over the 70%.

5 Conclusions

In this paper we presented a new framework for allocating and scheduling periodic applications modelled as Synchronous Data-Flow graphs with throughput and resource constraints. The framework relies on a different approach to scheduling applications where decisions to be taken are graph modifications that restrict the search space. Both search and constraint propagation works on graph modifications. We have devised a new incremental filtering algorithm for a throughput constraint. We proved its efficiency and speed-up w.r.t. the non-incremental version. We also proposed a cumulative resource constraint that uses and extends the Precedence Constraint Posting Approach to consider periodic applications. We now expect to gain the highest speed-ups from the optimization of the search strategy; hence future research will mostly focus on this topic.

References

1. Bonfietti, A., Lombardi, M., Milano, M., Benini, L.: Throughput constraint for synchronous data flow graphs. In: CPAIOR. (2009) 26–40
2. Blake, G., Dreslinski, R.G., Mudge, T.N.: A survey of multicore processors - a review of their common attributes. IEEE Signal Processing Magazine **26 issue 6** (Nov 2009) 26–37
3. Haid, W., Huang, K., Bacivarov, I., Thiele, L.: Multiprocessor SoC Software Design Flows - A focus on Kahn process networks. IEEE Signal Processing Magazine **26**(6) (November 2009) 64–71
4. Lee, E.A., Messerschmitt, D.C.: Synchronous dataflow. **75(9)** (September 1987) pp.1235–1245
5. Laborie, P.: Complete mcs-based search: Application to resource constrained project scheduling. IJCAI (2005) 181–186
6. Sriram, S., Bhattacharyya, S.: Embedded Multiprocessors Scheduling and Synchronization. Marcel Dekker, Inc (2000)
7. Baptiste, P., Pape, C.L., Nuijten, W.: Constraint-based scheduling: applying constraint programming to scheduling. Kluwer Academic Publisher (2001)
8. Policella, N., Cesta, A., Oddi, A., Smith, S.F.: From precedence constraint posting to partial order schedules. a csp approach to robust scheduling. AI Communications (2007) 163–180
9. Sriram, S., Lee, E.: Determining the order of processor transactions in statically scheduled multiprocessors. Journal of VLSI Signal Processing **220** (1996) 15:207
10. Hooker, J.N.: A hybrid method for planning and scheduling. (2004) 305–3162

Consistency Techniques for Hybrid Simulations

Student: Marco Bottalico[1],
Supervisor: Stefano Bistarelli[2],
Co-Supervisor: François Fages[3],
Tutor: Fabio Fioravanti[1]

[1] Dipartimento di Scienze, Università "G. d'Annunzio", Pescara, Italy
[bottalic,fioravanti]@sci.unich.it
[2] Dipartimento di Matematica e Informatica, Università di Perugia, Italy
bista@dmi.unipg.it
[3] EPI Contraintes, INRIA Paris - Rocquencourt, Le Chesnay Cedex, France
Francois.Fages@inria.fr

Abstract. In this paper, we investigate hybrid methods based on simulation of stochastic and deterministic models for biochemical systems, with consistency techniques in ordinary differential equations to have a preliminary vision on dissimilar methods to simulate different biochemical systems in Biocham.

Keywords: Ordinary differential equations, stochastic model, deterministic model.

1 Introduction

System biology is an interdisciplinary science, integrating experimental activity and mathematical modeling, which studies the dynamical behaviors of biological systems. An important problem in the modeling these systems is to characterize the dependence of certain properties on time and space. One frequently applied strategy is the description of the change of state variables by differential equations. If only temporal changes are considered, *ordinary differential equations* (ODEs) are used; for changes in time and space, partial differential equations are appropriate [6].

A variety of formalisms for modeling biological systems has been proposed in literature but in this paper we want to investigate only the consistency techniques in ordinary differential equations [5] and a new hybrid stochastic and deterministic model for biochemical systems [1].
There are two formalisms for mathematically describing the time behavior of a spatially homogeneous chemical system: the *deterministic approach* and the *stochastic approach*.
The deterministic approach regards the time evolution as a continuous, wholly predictable process which is governed by a set of coupled, ordinary differential equations (the "reaction-rate equations"). The stochastic approach regards the time evolution as a kind of random-walk process which is governed by a single

differential-difference equation (the "master equation"). Fairly simple kinetic theory arguments show that the stochastic formulation of chemical kinetics has a firmer physical basis than the deterministic formulation, but unfortunately the stochastic master equation is often mathematically intractable [8].

There is also a way to make exact numerical calculations within the framework of the stochastic formulation without having to deal with the master equation directly. We are talking about the Monte Carlo procedure to numerically simulate the time evolution of the given chemical system. Like the master equation, this "stochastic simulation algorithm" correctly accounts for the inherent fluctuations and correlations that are necessarily ignored in the deterministic formulation. Moreover this algorithm never approximates infinitesimal time increments dt by finite time steps Δt. The feasibility and utility of the simulation algorithm are demonstrated by applying it to several well-known model chemical systems, including the Lotka model, the Brusselator, and the Oregonator [8].

The goal of this paper is to show consistency techniques methods and hybrid stochastic/deterministic models to describe biochemical systems and their behaviour through the ordinary differential equations.

2 Related Work

In literature there are many contributions in the field of hybrid approach for problems which show a discrete and a continue behaviour at the same time.

In Biocham [3]: a software environment for modeling biochemical systems. It is based on two aspects: the analysis and simulation of boolean, kinetic and stochastic models and the formalization of biological properties in temporal logic. A model is defined by a set of reaction rules, possibly equipped with kinetic expressions, a list of parameter values and initial conditions. A specification that accounts for the relevant biological properties can also be added to the model as a list of temporal logic formulae. A single Biocham file can be used for boolean, continuous or stochastic analyses; for this reason it performs a hybrid approach for biological analysis.

3 Stochastic and Deterministic Models

The stochastic effects play an important role in biological processes leading to an increase in stochastic modelling attempts. The main problem related to the stochastic simulations regards times and computations which are very expensive [1].

The stochastic models have gained considerable attention when experiments conducted at the level of single cells showed the existence of a non-negligible level of noise in intracellular processes, like transcriptions and translation [7]. The dynamics of a stochastic system is described by the *chemical master equation* and in the 1976 Gillespie devised two exact algorithms to numerically simulate the stochastic time evolution of coupled chemical reactions, which are equivalent to solving the chemical master equation [8]. Only recently, modifications

to the original chemical master equation have been proposed to further speed up simulations. The most important methods involve the averaging over fast reactions [10], application of quasi-steady-state theory [13], grouping together reactions that occur in fast succession [2].

Another strategy is to model those processes that either involve large number of particles or have fast rates, in a deterministic way, keeping stochastic the remaining ones [1]. There are two recent algorithms to simulate biochemical systems in such hybrid framework that have been proposed [11, 15]. In both cases, the main idea is to first predict the time in which a stochastic event should be occur and then evolve the system of ordinary differential equations. At specific instant in time, the system is updated, and it is checked whether the stochastic event has to be performed or not. Instead in [1] the authors propose a rigorous mathematical ground for hybrid stochastic and deterministic modelling in a natural way. There are three different algorithms: the direct hybrid method, the first reaction hybrid method and the next reaction hybrid method. The main difference between the first two approaches and the second one is essentially one: they are based on a prediction correction heuristic for the realization of the stochastic part that can be seen as an approximation to the simultaneous solution of the system of ODEs which in [1] are precisely calculated.

Consider N chemical species $S_1, ..., S_N$ involved in M reactions $R_1, ..., R_M$. Chemical species are modelled in terms of number of molecules $X(t) = (X_1(t), ..., X_N(t))$. The reaction rate for each reaction R_j is specified by a so-called propensity function $a_j = a_j(X(t), t)$, which is equal to the product rate constant c_j and the number of possible combinations of reactant molecules involved in reaction R_j. Once a reaction R_j is performed, the number of molecules for each species is updated according to the state change vector v_j, i.e., $X(t) \leftarrow X(t) + v_j$ [1].

The **deterministic model** is based on the law of mass action, where a system of coupled ordinary differential equations (ODEs) is established for the time evolution of the number of molecules $X(t) \in \mathbb{R}_+^N$

$$\frac{d}{dt}X(t) = \sum_{j=1}^{M} v_j a_j(X(t), t) \tag{1}$$

with some initial value $X(t_0) \in \mathbb{R}_+^N$. While the system should be described as a vector of integers, this model needs real values for $X(t)$. This is however acceptable under the assumption of large number of molecules ($X_i(t) >> 1$) so that the relative error can be neglected [1].

The **stochastic model** is based on physical laws and the idea that chemical reactions are essentially random processes, the stochastic formulation of chemical reactions is given in terms of a Markov jump process $X(t) \in \mathbb{N}^N$ [9]. Its characterization is based on the probability $a_j(X(t), t)dt$ of a reaction R_j occurring in the next infinitesimal time interval $[t, t + dt]$. Denoting by $T_j(t)$ the time at which reaction R_j first occur after t, this amounts to write that

$$\mathbf{P}[T_j(t) \in [t, t + dt]|X(t)] = a_j(X(t), t)dt. \tag{2}$$

In [1] the authors consider a partition of the reactions $R_1, ..., R_M$ into those modelled stochastically (labeled with index $j \in \mathcal{S}$) and those modelled deterministically (labeled with index $j \in \mathcal{D}$). Now we can write the evolution equation for $X(t) \in \mathbb{R}^N$ which is given by the following hybrid system

$$dX(t) = \sum_{j \in \mathcal{D}} v_j a_j(X(t), t)dt + \sum_{j \in \mathcal{S}} v_j dN_j(t) \tag{3}$$

To partition the reactions the authors suggest some methods:

- run a fully stochastic realization and analyze the frequencies/propensities of each reaction;
- use biological insight (i.e. in [1] the authors say that seems reasonable to model gene regulatory parts stochastically, while metabolic reactions deterministically);
- for each reaction choose adaptively between two approaches using a criterion based on the number of the molecules and its propensity function.

To check if the algorithms based on hybrid model (direct hybrid method, first and next reaction methods) obtained good results they tested them in a intracellular growth of bacteriophage T7 derived by [14]. From the experiment appears that the hybrid simulations are about 100 times as fast as the fully stochastic ones without compromising the results accuracy (fig. 1).

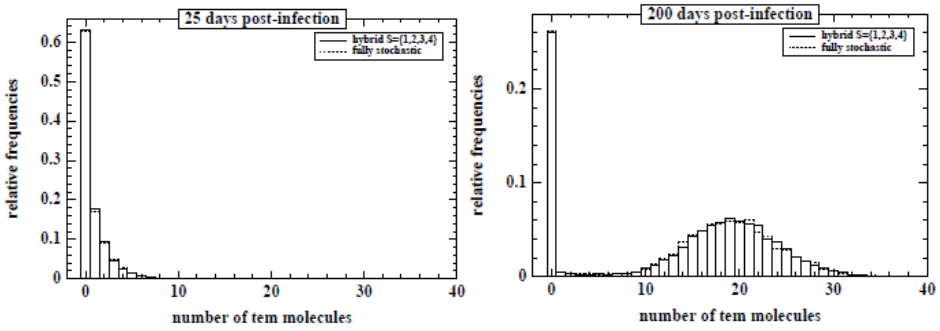


Fig. 1. Hybrid kinetics for the bacteriophage T7 model (reaction R_1, R_2, R_3 and R_4 modelled stochastically, reactions R_5 and R_6 modelled deterministically) compared to the the reference fully stochastic model (based on 10^4 realizations) [1].

4 Consistency Techniques in Ordinary Differential Equations

How we have explained in the previous section, the ordinary differential equations (ODEs) play a crucial role in the deterministic model. A first order (ODE) system O is a system of the form

$$
\begin{aligned}
u_1'(t) &= f_1(t, u_1(t), ..., u_n(t)) \\
u_2'(t) &= f_2(t, u_1(t), ..., u_n(t)) \\
&\vdots \\
u_n'(t) &= f_n(t, u_1(t), ..., u_n(t))
\end{aligned}
$$

In [5] the author uses the vector representation $u'(t) = f(t, u(t))$ or more simply $u' = f(t, u)$. At this point, there are two assumption:
(1) the function f is sufficiently smooth;
(2) the existence and uniqueness of a solution.
Now, given an initial condition $u(t_0) = u_0$ and for the second assumption, the solution of O is a function $s^* : \mathbb{R} \to \mathbb{R}^s$ satisfying O and the initial condition $s^*(t_0) = u_0$.

Although for some classes of ODEs the solution can be represented in closed form, most ODE systems cannot be solved explicitly [5]. The *discrete variable method* aim at approximating the solution $s^*(t)$ of any ODE system, not over a continuous range of t, but only at some points $t_0, t_1, ..., t_m$. This method include *one-step methods* and *multi-step methods*; in general these methods do not guarantee the existence of a solution within a given bound.
The *interval analysis method* instead, was introduced by Moore [12] in the 1966. These methods provide numerically reliable enclosures of the exact solution at points $t_0, t_1, ..., t_m$. To achieve the result, they typically apply a one-step Taylor interval method and make extensive use of automatic differentiation to obtain the Taylor coefficients[5].

The mayor problem of interval analysis methods on ODE systems is the explosion of the size of resulting boxes at point $t_0, t_1, ..., t_m$. For the author, there are two reasons for this explosion: at first this method have a tendency to accumulate errors from point to point, second the approximation of an arbitrary region by a box (wrapping effect) may introduce considerable loss of accuracy after a number of steps.

For all these reasons, in[4, 5] they show how to provide a unifying framework to extend traditional numerical techniques to intervals providing reliable enclosures. The first contribution is to extend explicit and implicit, one-step and multi-step methods to intervals. The second one is to generalize interval techniques into a two-step process: a forward process (to compute an enclosure) and a backward process (to reduce this enclosure).

5 Future Work

The importance of precise analysis to study and comprise biological phenomena involve different kind of models. On the one hand, it is necessary to describe some parts in a rigorous and accurate numerical method (for example methods based on ordinary differential equations or stochastic methods). In the other hand, the lack of evidence, drives the analysis on purely qualitative models (boolean or discrete models).

The goal is to implement in Biocham some techniques to realize hybrid simulation, combining different kinds and different nature models, in a qualitative and quantitative optical, with discrete and continue dynamics. The solution is to provide the specific language with a multi level description mechanism for the modelization; the second step is to distinguish in the formalism, the common characteristics from the details. At last we want to specify the criteria to change, during the simulation, the formalism.

References

1. A. Alfonsi, E. Cancès, G. Turinici, B. Di Ventura, and W. Huisinga. Exact simulation of hybrid stochastic and deterministic models for biochemical systems. Research Report RR-5435, INRIA, 2004.
2. K. Burrage, T. Tian, and P. Burrage. A multi-scale approach for simulation chemical reaction systems. *Progress Biophys Mol Biol*, 85:217–234, 2004.
3. L. Calzone, F. Fages, and S. Soliman. Biocham: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006.
4. Y. Deville, M. Janssen, and P. Van Hentenryck. Multistep filtering operators for ordinary differential equations. In J. Jaffar, editor, *CP*, volume 1713 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 1999.
5. Y. Deville, M. Janssen, and P. Van Hentenryck. Consistency techniques in ordinary differential equations. *Constraints*, 7(3-4):289–315, 2002.
6. K. Edda, H. Ralf, K. Axel, W. Christoph, and L. Hans. *Systems Biology in Practice: Concepts, Implementation and Application*. Wiley-VCH, 1 edition, May 2005.
7. M.B. Elowitz, A.J. Levine, E.D. Siggia, and P.S. Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002.
8. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.
9. D.T. Gillespie. *Markov Processes-An Introduction for Physical Scientists*. Academic Press, 1992.
10. E.L. Haseltine and J.B. Rawlings. Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *The Journal of Chemical Physics*, 117:6959–6969, 2002.
11. T.R. Kiehl, R.M. Mattheyses, and M.K. Simmons. Hybrid simulation of cellular behavior. *Bioinformatics*, 20(3):316–322, 2004.
12. R.E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.
13. C.V. Rao and A.P. Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the gillespie algorithm. *The Journal of Chemical Physics*, 118:4999–5010, 2003.
14. R. Srivastava, L. You, J. Summers, and J. Yin. Stochastic vs. deterministic modeling of intracellular viral kinetics. *Journal of Theoretical Biology*, 218(3):309–321, October 2002.
15. K. Takahashi, K. Kaizu, B. Hu, and M. Tomita. A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, 20:538–546, 2004.

A Semiring-based framework for fair resources allocation

Paola Campli (campli@sci.unich.it),
Supervisor: Stefano Bistarelli (bista@dipmat.unipg.it)
Tutor: Maria Chiara Meo (meo@sci.unich.it)
Co-Supervisor: Barry O'Sullivan (b.osullivan@cs.ucc.ie)

University G.D'Annunzio - Pescara, Italy

Abstract. In this paper we propose a general framework to model and solve the fair allocation problem (with indivisible objects) by using soft constraints. Our formal approach allows to model different allocation problems, ranging from goods and resources allocation to task and chore division. We use soft constraints to find a fair solution by respecting the agents's preferences; indeed these can be modeled in a natural fashion by using the Semiring-based framework for soft constraints. The fairness property is considered following an economical point of view, that is, taking into account the notions of *envy-freeness* (each player likes its allocation at least as much as those that the other players receive, so it does not envy anybody else) and *efficiency* (there is no other division better for everybody, or better for some players and not worse for the others).

1 Introduction and motivations

The problem of "fair division", that is, fairly dividing resources or costs among a set of people, is an important issue in real life scenarios; it can refer to several situations, such as inheritance and divorce settlements, division of health resources, computer networking resources, voting power, intellectual property licenses, costs for environmental improvements, etc. In these cases, formal protocols for division are needed. Many variations of the basic problem exist, for example, the situation with divisible resources is quite different from the situation with indivisible objects; the items to be divided can be goods or sometimes "bads" like chores or other burdens; some problems can involve the division of money to compensate a "non fair share", or a payoff in exchange for performing a chore. Other aspects to consider are the number of objects with respect to the number of people. If goods are scarce, an *auction* is needed and the items are assigned to (usually) one winner; in this case fairness methods are studied in repeated auctions to guarantee that not always the same player will be the winner.

But most of the variation comes from the fact that there are many reasonable ways to formalize "fairness" including *max-min fairness*, *proportional fairness*, *envy-free* fairness, etc. which may or may not lead to the same optimal allocation; if we take into account a *global view* this means looking at the overall allocation in terms of social welfare, while a *local view* focus on the agents preferences.

Although there is wide literature on fair division within the fields of economics, game theory, political science, mathematics, operations research and computer science,

it seems to lack a unified and general framework which allows to solve the different kinds of problems, each one with different objects (desirable or undesirable items), weights or preferences. Another issue encountered in previous works is that often it is not possible to find a solution and the problem remains unsolved; our approach might be applied in all these cases because the use of soft constraints allows to always find a solution; moreover we provide a general framework which can model several cases by choosing the appropriate semiring (see Sec. 3).

In this paper we investigate the allocation of indivisible objects which can be either goods or bads; thus, given a set of items and a set of people, each person states a weight for each object which, depending on the cases, can represent preferences or costs (such as time, money, resources etc.). According our model, the solution will be an *envy-free* allocation of objects to the agents, reminding that envy-freeness is a fairness property that guarantees that no agent would rather have one of the bundle allocated to any of the other agents, that is, each player prefers his bundle.

The paper is organized as follow: in section 2 the various aspects of fair division problems are illustrated; in section 3 we give a background on Soft Constraint Satisfaction Problems and semirings; in section 4 we show how to use the SCSP framework to model allocation problems; in section 5 the mapping between SCSP and allocation problems is provided; finally, section 6 contains references to similar works in the field of fair division and allocation problems, a short summary and our future works.

2 The problem of fair division

Fair division [6] is the problem of dividing one or several goods amongst two or more agents in a way that satisfies a suitable fairness criterion. Fair division has been studied in philosophy, political science, economics and mathematics for a long time, but is also relevant to computer science and multiagent systems (MAS), in which resource allocation is a central topic since the application or agents need resources to perform tasks. It is assumed that agents are autonomous. A solution needs to respect and balance their individual preferences; fairness definitions are required and once we have a well-defined fair division problem, we require an algorithm to solve it.

The elements of a Fair Division Problem are a set P of n players: $p_1, p_2, ..., p_n$ and a set of m objects O to be divided. The problem is to divide the set O into n shares ($o_1, o_2, ..., o_n$) so that each player gets a fair share of O. A *fair share* is any share that, in the opinion of the player receiving it, has a value that is at least $\frac{1}{n}$ of the total value of the set of goods O. It is crucial to understand that share value is subjective, and that each player may even have a different notion of how much the set to be divided is worth.

There are three types of fair division schemes: the *Continuous Fair Division Schemes*, in which the set O is infinitely divisible (cake, land, etc.) and shares can be adjusted by arbitrarily small amounts; the *Discrete Fair Division Schemes* where the set O is made up of indivisible objects (cars, houses, etc) and the *Mixed Fair Division Schemes*. In this paper, since we are dealing with indivisible objects, we will focus on the discrete case.

3 Constraint Satisfaction Problems, Semirings and Soft Constraints

The classic definition of a Constraint Satisfaction Problem (CSP) is as follows [10]. A CSP P is a triple $P = \langle X, D, C\rangle$ where X is an n-tuple of variables $X = \langle x_1, x_2, \ldots, x_n\rangle$, D is a corresponding n-tuple of domains $D = \langle D_1, D_2, \ldots, D_n\rangle$ such that x_i can assume values within a determined domain D_i, C is a t-tuple of constraints $C = \langle C_1, C_2, \ldots, C_t\rangle$. A constraint C_j is a pair $\langle Rl_{S_j}, S_j\rangle$ where Rl_{S_j} is a relation on the variables in $S_i = \textit{scope}(C_i)$. A solution to the CSP P is an n-tuple $A = \langle a_1, a_2, \ldots, a_n\rangle$ where $a_i \in D_i$ and each C_j is satisfied in that Rl_{S_j} holds on the projection of A onto the scope S_j . In a given task one may be required to find the set of all solutions, ***sol(P)***, to determine if that set is non-empty or just to find any solution, if one exists. If the set of solutions is empty the CSP is unsatisfiable.

A *c-semiring* [5, 3] S (or simply semiring in the following) is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ where A is a set with two special elements $\mathbf{0}, \mathbf{1} \in A$ (respectively the bottom and top elements of A) and with two operations $+$ and $\times$ that satisfy certain properties: $+$ is defined over (possibly infinite) sets of elements of A and is commutative, associative and idempotent; it is closed, $\mathbf{0}$ is its unit element and $\mathbf{1}$ is its absorbing element; $\times$ is closed, associative, commutative and distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element (for the exhaustive definition, please refer to [5]). The $+$ operation defines a partial order $\leq_S$ over A such that $a \leq_S b$ iff $a + b = b$; intuitively $a \leq_S b$ if b represents a value *better* than a. Other properties related to the two operations are that $+$ and $\times$ are monotone on $\leq_S$, $\mathbf{0}$ is its min and $\mathbf{1}$ its max, $\langle A, \leq_S\rangle$ is a complete lattice and $+$ is its lub.

A *soft constraint* [5, 3] may be seen as a constraint where each instantiation of its variables has an associated preference. Given $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and an ordered set of variables V over a finite domain D, a soft constraint is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the semiring. Using this notation $\mathcal{C} = \eta \rightarrow A$ is the set of all possible constraints that can be built starting from S, D and V. Any function in $\mathcal{C}$ depends on the assignment of only a finite subset of V. For instance, a binary constraint $c_{x,y}$ over variables x and y, is a function $c_{x,y} : V \rightarrow D \rightarrow A$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$ (the *support*, or *scope*, of the constraint). Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the assignment $v := d_1$. Notice that $c\eta$ is the application of a constraint function $c : V \rightarrow D \rightarrow A$ to a function $\eta : V \rightarrow D$; what we obtain is a semiring value $c\eta = a$. $\bar{\mathbf{0}}$ and $\bar{\mathbf{1}}$ represent the constraint functions associating $\mathbf{0}$ and $\mathbf{1}$ to all assignments of domain values; the $\bar{a}$ function always returns the value a.

Given the set $\mathcal{C}$, the combination function $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times c_2\eta$ [5, 3]. The $\otimes$ builds a new constraint which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples. Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* [5, 3, 4] of c over $V - \{v\}$, written $c \Downarrow_{(V\setminus\{v\})}$ is the constraint c' such that $c'\eta = \sum_{d\in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support.

A SCSP [3] is a tuple $P = \langle X, D, C, A\rangle$ where X is a set of variables, D is the domain of the variables and C is a set of constraints over X associating values form a

c-semiring A. The *best level of consistency* notion defined as $blevel(P) = Sol(P) \Downarrow_{\emptyset}$, where $Sol(P) = \bigotimes C$ [3]. A problem P is α-consistent if $blevel(P) = \alpha$ [3]; P is instead simply "consistent" iff there exists $\alpha >_S \mathbf{0}$ such that P is α-consistent [3]. P is inconsistent if it is not consistent.

4 The SCSP framework for allocations problems

In this section we define a quantitative framework to model fair division problems, where each assignment of objects to people have an associated preference/weight and, consequently, modeling this kind of problems as Soft CSPs (see Sec. 3) leads to an allocation of goods to people that optimize the criteria defined by the chosen semiring.

For instance, the *Weighted* semiring $\langle \mathbb{R}^+, min, \hat{+}, 0, 1 \rangle$, where $\hat{+}$ is the arithmetic plus ($\mathbf{0} = \infty$ and $\mathbf{1} = 0$), can be used to model the undesirable objects case (such as chore division) by expressing the (e.g. money) cost for performing a chore; the optimum solution in this scenario corresponds to an allocation with minimum total cost.

The *Fuzzy* semiring $\langle [0..1], max, min, 0, 1 \rangle$ is well suited for modeling the players's preferences with respect to each good; the solution we obtain with this semiring corresponds in choosing the highest among the minimum preferences.

The *Probabilistic* semiring $\langle [0..1], max, \hat{\times}, 0, 1 \rangle$ can be used when preferences are unknown, thus, weights corresponds to probabilities; we can express for instance, that person p_1 prefers object o_3 with probability 0.4. The arithmetic $\hat{\times}$ is used to compose the probability values together (since we assume that preferences and thus probabilities are independent).

By using the *Boolean* semiring $\langle \{true, false\}, \vee, \wedge, false, true \rangle$ we can solve the non weighted allocation problems, that is, each person state only the goods he/she desires (or the tasks he is able to perform).

5 Mapping Allocation Problems to SCSPs

In this section we show a mapping from the allocation problem to SCSPs. An allocation problem is formed by a set of m indivisible objects (or items) $O = \{o_1, o_2, \ldots, o_m\}$ and a set of people (or players) $P = \{p_1, \ldots, p_n\}$. Each player has their own preferences or costs regarding the allocation of goods/tasks to be selected. The problem consists in partitioning the set of objects in n subsets (or bundles) in a way that each person receives a (non-empty) bundle that satisfies a suitable fairness criterion. In order to model this problem with a SCSP, we define a variable for each object $o_i \in O$, i.e. $V = \{o_1, o_2, \ldots, o_m\}$ and the domain of each variable is the set of People in P: $D = \{p_1, \ldots, p_n\}$, meaning that an object can be assigned to a person in the set P; for example $o_1 = p_2$ means that player p_2 receives object o_1. A soft constraint associates a semiring value for each assignment of the variables in its scope, which represent the preference of the player for a given item; if no weights are considered , the corresponding variable assignment is *not admitted* or *admitted* and the values **0** or **1** of the boolean semiring set are respectively returned.

Example 1. As a simple example, suppose we must assign 3 objects (o_1, o_2, o_3) to 2 players (p_1, p_2). The corresponding SCSP, by using (for instance) a Fuzzy semiring, has three variables: o_1, o_2 and o_3, each with the domain $D = \{p_1, p_2\}$; we define the following unary constraints: $C_{o1} := \{(p_1, 0.7); (p_2, 0.2); \}$ meaning that object 1 can be assigned either to person 1 (who has a preference of 0.7 for this objects) or person 2 (with preference 0.2); $C_{o2} := \{(p_1, 0.3); (p_2, 0.1); \}$ that is, object 2 can be assigned either to person 1 or 2 with preferences 0.3 and 0.1 respectively; $C_{o3} := \{(p_2, 0.7)\}$ meaning that object 3 can only be assigned to person 2 who desires the object with preference 0.7;

the solution is illustrated in the table below:

o1	o2	o3		$Sol(P)$
p1	p1	p1	not allowed	
p1	p1	p2	$0.7 \times 0.3 \times 0.7$	**0.3**
p1	p2	p1	not allowed	
p1	p2	p2	$0.7 \times 0.1 \times 0.7$	0.1
p2	p1	p1	not allowed	
p2	p2	p1	not allowed	
p2	p2	p2	$0.2 \times 0.1 \times 0.7$	0.1
p2	p1	p2	$0.2 \times 0.3 \times 0.7$	0.2

The (unique) optimal solution of this problem is $o_1 = p_1, o_2 = p_1 o_3 = p_2$ (with preference 0.3).

Depending on the cases, the solution provided might not be fair if we only use the previous method. For example, with different preferences, the solution (p_2, p_2, p_2) could be returned, which is certainly unfair, since player 1 does not get any object and envies player 2. For this reason, we need to specify additional constraints in order to solve the allocation problem and guarantee the *envy-freeness* property of fairness. Let x_{ij} be a boolean variable which is equal to 1 if item j is assigned to player i and 0 otherwise and let $u_i(B_i)$ be the value for person i of the set of objects (B_i) assigned to him; this value represents the valuation of the bundle (that is, the subset of items) for each person; an issue encountered in this case is that requesting an input to the agents for every possible combination of goods is NP-hard, in fact for m object there are 2^m valuations for each of the n players. In order to reduce the size of the problem, we can automatically calculate the value of the bundle, by specifying in the problem if the valuations are *additive* (thus, the value is obtained by summing the weights of the single objects in the bundle), *super-additive* (the value of the bundle is greater than the values of the single objects), *sub-additive* (the value of the bundle is lower than the values of the single objects) or *maximal* (the value of the bundle corresponds to the maximum weight among the objects in the bundle); in this way we can compute the value of the entire bundle $u_i(B_i)$; the type of valuation depends on the kind of goods; for example if the items considered are *complementary* (e.g. printers and ink cartridges) the valuation of the bundle might be super-additive, or conversely, if the goods are *substitute* (e.g. petroleum and natural gas), the valuation might be sub-additive. The defined constraints are the following: [1]

1. Each object must be assigned to at most one person $\forall j \quad \sum_i x_{ij} = 1$;
2. Each person must receive at least one item: $\forall i \quad \sum_j x_{ij} \geq 1$;
3. *Envy-freeness constraint.* Each person does not prefer the set of objects assigned to the other players: $\forall i \quad u_i(Bi) \geq u_i(Bj) \quad$ for each $j \neq i$;

[1] constraints 1 and 2 are based on those used in the Santa Claus Problem's paper [2]

Moreover, since we are assuming that the number of objects is greater (or equal) than the number of people, our solution is also efficient, as shown in [1], which proves that when $m \geq n$ envy-freeness implies efficiency.

6 Related works, conclusions and future works

In the indivisible resource area, most of the issues are based on the *Santa Claus problem* [2], in which the goal is to distribute n presents among k kids, in such a way that the least lucky kid is as happy as possible; a linear programming is used with the *Max-min fairness* objective function. Another variant is represented by the "housemate problem" [9], where goods are associated with bads. The problem consists in assigning different rooms to people sharing a house according the bid they submit, but also to determine a price to be paid by each roommate for his assigned room. Concerning Chore division, the problem of dividing an undesirable object has been investigated only for divisible goods, where cake cutting algorithms have been adapted in order to deal with chores instead of desirable goods. It is supposed that chores are infinitely divisible [8] and valuations over bundles are additive. Other works view the problem from a computational perspective and are based on approximation algorithms with the purpose of finding a solution closest to the optimum [7].

We investigated on the use of the semiring-based framework for soft constraints in order to model and solve the fair allocation of objects problem. According the chosen semiring, we can easily represent the different set of preferences, their combination and the various kind of objects. In the future we plan to use the framework for the Stable Marriage Problem, which can be casted in a particular fair allocation problem involving the same number of objects and people.

References

1. Ahmet Alkan, Gabrielle Demange, and David Gale. Fair allocation of indivisible goods and criteria of justice. *Econometrica*, 59(4):1023–1039, 1991.
2. Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 31–40, New York, NY, USA, 2006. ACM.
3. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.
4. S. Bistarelli, U. Montanari, and F. Rossi. Soft concurrent constraint programming. *ACM Trans. Comput. Logic*, 7(3):563–589, 2006.
5. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.
6. Ulle Endriss. Lecture notes on fair division. 2009.
7. Uriel Feige. On allocations that maximize fairness. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 287–293, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.
8. Elisha Peterson and Francis Edward Su. Four-person envy-free chore division. *Mathematics Magazine*, 2002.
9. Richard F. Potthoff. Use of linear programming to find an envy-free solution closest to the bramskilgour gap solution for the housemates problem. *Group Decision and Negotiation*, 11:405–414, 2002.
10. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., New York, NY, USA, 2006.

Symmetries and Lazy Clause Generation

Geoffrey Chu[1] (student), Maria Garcia de la Banda[2], Chris Mears[2], and Peter J. Stuckey[1] (supervisor)

[1] National ICT Australia, Victoria Laboratory,
Department of Computer Science and Software Engineering,
University of Melbourne, Australia
{gchu,pjs}@csse.unimelb.edu.au
[2] Faculty of Information Technology,
Monash University, Australia
{chris.mears,mbanda}@infotech.monash.edu.au

Abstract. Lazy clause generation is a powerful approach to reducing search in constraint programming. This is achieved by recording sets of domain restrictions that previously lead to failure as new clausal propagators. Symmetry breaking approaches are also powerful methods for reducing search by recognizing that parts of the search tree are symmetric and do not need to be explored. In this paper we show how we can successfully combine dynamic symmetry breaking methods with lazy clause generation. Further, we show that the more precise nogoods generated by a lazy clause solver allows our combined approach to exploit redundancies that cannot be exploited via any previous symmetry breaking method, be it static or dynamic.

1 Introduction

Lazy clause generation [5] is a hybrid approach to constraint solving that combines features of finite domain propagation and Boolean satisfiability. Finite domain propagation is instrumented to record the reasons for each propagation step. This creates an implication graph like that built by a SAT solver, which may be used to create efficient nogoods that record the reasons for failure. These nogoods can be propagated efficiently using SAT unit propagation technology. The resulting hybrid system combines some of the advantages of finite domain constraint programming (high level model and programmable search) with some of the advantages of SAT solvers (reduced search by nogood creation). Lazy clause generation provides state of the art solutions to a number of combinatorial optimization problems such as Resource Constrained Project Scheduling Problems [6].

Symmetry breaking methods aim to speed up execution by pruning parts of the search tree known to be symmetric to those already explored. This can be achieved via both static methods like lexical graphical constraints, and dynamic methods like Symmetry Breaking During Search (SBDS) [3] and Symmetry Breaking by Dominance Detection (SBDD) [1], [2]. Dynamic approaches have the benefit that they do not interfere with the branching heuristic. In this paper, we examine how to combine lazy clause generation with SBDS and show that we can exploit types of symmetries that could not previously be exploited by any other method.

2 Lazy Clause Generation

Lazy clause generation is a hybrid of finite domain and SAT solving where each FD propagator is extended to be able to explain its propagations. An integer variable x in problem $P = (C, D)$ with initial domain $D(x) = [\, l \,..\, u \,]$ is channelled with a set of Boolean variables representing $\{[\![x = d]\!] \mid l \leq d \leq u\} \cup \{[\![x \leq d]\!] \mid l \leq d < u\}$. Each domain change inferred by a propagator f can be expressed as fixing the value of one of these Boolean variables. The inference must be explain in terms of the other boolean variables associated with the variables involved in the propagator. An *explanation* for a domain change represented by literal l is of the form $S \rightarrow l$ where S is a set of literals that are currently true. We require that each propagator f return an explanation for every domain change it makes. For example the propagator for $x \neq y$ may explain the inference that $y \neq 3$ given $x = 3$ as $[\![x = 3]\!] \rightarrow \neg[\![y = 3]\!]$. The explanations are stored, forming an *implication graph* like that in a SAT solver.

A *nogood* N is a set of literals such that $C \wedge D \Rightarrow \neg \wedge_{l \in N} l$. That is, in all solutions of P, the conjunction of the literals in N is false. Lazy clause solvers derive nogoods from each conflict by analysing the implication graph. We begin with the explanation of failure as the working clause. While the working clause has more than 1 literal on the last decision level, we eliminate the literal which was fixed last by resolving the working clause with the explanation clause for that literal. When only one literal at the last decision level remains, we have what is called the 1UIP (First Unique Implication Point) nogood. Another possible nogood we could derive is the *choice* nogood, where we simply take the entire set of decision literals leading to the failed subproblem as the nogood. Nogoods are added to the solver as clausal propagators.

3 Symmetries and Nogoods

Nogoods are globally valid constraints, thus symmtries can be applied to nogoods to derive additional correct nogoods.

Example 1. Consider the following constraint problem $P = (C, D)$ where $C \equiv \{\sum_{i=1}^{5} x_i \leq 12,\ \mathit{alldiff}([x_1, x_2, x_3, x_4, x_5])\}$ and $D(x_i) = [\, 1 \,..\, 8 \,], 1 \leq i \leq 5$. This problem has variable interchangeability symmetries since each of $\{x_1, x_2, x_3, x_4, x_5\}$ are indistinguishable. Suppose the subproblem P' with $x_1 = 1, x_2 = 2$ fails, then we can derive the choice nogood $\{x_1 = 1, x_2 = 2\}$. Clearly any symmetric version is also a correct nogood, for example $\{x_2 = 1, x_1 = 2\}$ or $\{x_3 = 1, x_5 = 2\}$. □

Dynamic symmetry breaking techniques like SBDS and SBDD can be thought of as propagating symmetric versions of the choice nogoods derived at each node. SBDS works as follows. Suppose we reach subproblem P' via the decision sequence $d_1, d_2, \ldots, d_n, d_{n+1}$, where each d_i is a literal representing the decision assignment at the ith level. If P' fails, we can derive the following choice nogood: $\{d_1, \ldots, d_n, d_{n+1}\}$. SBDS treats the nogoods as one-directional, i.e. $\{d_1, \ldots, d_n\} \rightarrow \neg d_{n+1}$. This constraint only propagates if $d_1, \ldots, d_n$ are true in which case it asserts $\neg d_{n+1}$. Suppose P'' was the parent subproblem of P'. When we backtrack to P'', i.e. to the point where we undo d_{n+1}, we do the

following. For each symmetry ρ, we post the symmetric one-directional nogood $\{\rho(d_1), \ldots, \rho(d_n)\} \rightarrow \neg\rho(d_{n+1})$ as a local constraint in P'', ignoring those ρ for which $\exists i$ s.t. $\neg\rho(d_i)$ is entailed in P'', as such nogoods are redundant within subproblem P''. From this point on, we refer to normal SBDS as SBDS-choice (since it uses choice nogoods).

4 Symmetries and Lazy Clause Generation

Naively, we can simply add SBDS-choice to a lazy clause solver. However, we show that using the 1UIP nogoods normally derived by a lazy clause solvers lead to even better performance.

4.1 SBDS-1UIP

It is easy to see how to adapt SBDS to use 1UIP nogoods. Suppose subproblem P' fails and we derive the 1UIP nogood $\{l_1, \ldots, l_n, l_{n+1}\}$ where l_{n+1} is the asserting literal. When we backtrack to P'', the parent subproblem of P', we do the following. For each symmetry ρ, we post the symmetric (one-directional) nogood $\{\rho(l_1), \ldots, \rho(l_n)\} \rightarrow \neg\rho(l_{n+1})\}$, ignoring those ρ for which $\exists i$ s.t. $\neg\rho(l_i)$ is entailed in P''.

It is possible to show that SBDS-1UIP exploits strictly more symmetries than SBDS-choice. We omit the proof for lack of space. This means that SBDS-1UIP can prune even more than complete methods like SBDS-choice. Roughly speaking, SBDS-choice can only exploit symmetries on the already labelled parts of the problem, whereas SBDS-1UIP can exploit symmetries in the unlabeled parts of the problem as well.

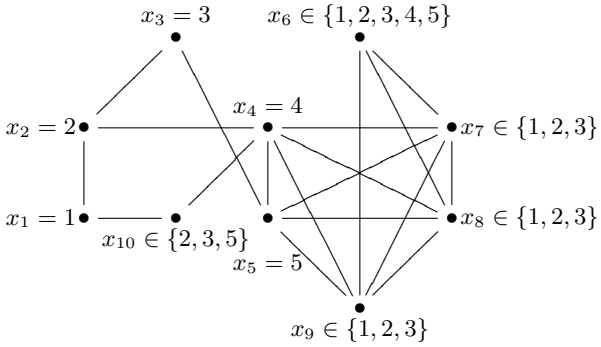


Fig. 1. A graph colouring problem where we can exploit additional symmetries

Example 2. Consider the graph colouring problem shown in Figure 1, where we are trying to colour the nodes with at most 5 colours. After making the decisions $x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5$, we have domains as shown in

Figure 1. Suppose we label $x_6 = 1$ next. Then propagation gives $x_7 \in \{2, 3\}, x_8 \in \{2, 3\}, x_9 \in \{2, 3\}$. Now, suppose we try $x_7 = 2$. This forces $x_8 = 3, x_9 = 3$, which conflicts. The (one-directional) 1UIP nogood from this conflict is $\{x_8 \neq 1, x_8 \neq 4, x_8 \neq 5, x_9 \neq 1, x_9 \neq 4, x_9 \neq 5\} \rightarrow x_7 \neq 2$. After propagating this nogood, we have $x_7 = 3$, which after further propagation, once again conflicts. At this point, we backtrack to before x_6 is labelled and derive the nogood $\{x_7 \neq 4, x_7 \neq 5, x_8 \neq 4, x_8 \neq 5, x_9 \neq 4, x_9 \neq 5\} \rightarrow x_6 \neq 1$.

Now, let's examine what SBDS-1UIP can do at this point. It is clear that if we apply the value symmetries $\ll 1 \gg \leftrightarrows \ll 2 \gg$ or $\ll 1 \gg \leftrightarrows \ll 3 \gg$ to this nogood, the LHS remains unchanged while the RHS changes. Therefore, we can post these two symmetric nogoods and immediately get the inferences $x_6 \neq 2$ and $x_6 \neq 3$. On the other hand, SBDS-choice can't do anything. The choice nogood is $\{x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4, x_5 = 5\} \rightarrow x_6 \neq 1$, and it is easy to see that no matter which value symmetry we use on it, the LHS will have a set of literals incompatible with the current set of decisions and thus cannot imply the RHS. □

The kind of redundancy we exploit here is extremely difficult to exploit. Roughly speaking, we can say that we are exploiting the symmetry that exists in the sub-component of a subproblem which is the actual cause of failure. In this case, they are the variables x_6, x_7, x_8, x_9, their current domains in the subproblem, and the constraints linking them. Neither static symmetry breaking constraints or conditional symmetry breaking constraints are capable of exploiting this symmetry. It is only because a lazy clause solver gives us such precise information about which variables are involved in failures that SBDS-1UIP can exploit this.

5 Experiments

We now perform experiments on the the Concert Hall Scheduling problem and the Graph Colouring problem. We take the benchmarks used in [4].

We implemented SBDS in Chuffed, which is a state of the art lazy clause solver. We run Chuffed with three different versions of SBDS. The first version is choice, where we use symmetric versions of choice nogoods. The second is 1UIP, where we use symmetric versions of 1UIP nogoods. The third version we call crippled, where we use symmetric versions of 1UIP nogoods, but only those nogoods derived from symmetries where choice could also exploit the symmetry. We also compare against Chuffed with no symmetry breaking (none), and with static symmetry breaking constraints (static). Finally, we compare against an implementation of SBDS in [4], which is called Lightweight Dynamic Symmetry Breaking (LDSB). LDSB is implemented on the Eclipse constraint programming platform and was the fastest implementations of dynamic symmetry breaking on the two problems we examine, beating GAP-SBDS and GAP-SBDD by significant margins.

All versions of Chuffed are run on Xeon Pro 2.4GHz processors. The results for LDSB were received from the authors of [4], where Eclipse LDSB was run on quad-core Intel Xeon E5310 1.66GHz processors. We group the instances by size, so that the times displayed are the average run times for the instances of each

Table 1. Comparison of three SBDS implementations in Chuffed, static symmetry breaking in Chuffed, and LDSB in Eclipse, on the Concert Hall Scheduling problem

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
20	259.8	686018	0.04	84	0.05	130	0.07	350	0.05	134	0.29	3283
22	381.5	749462	0.07	181	0.08	299	0.17	1207	0.07	183	0.73	7786
24	576.9	1438509	0.10	275	0.11	316	0.78	3426	0.15	486	2.70	12611
26	483.4	1189930	0.10	282	0.19	677	2.26	5605	0.25	685	2.71	12724
28	530.7	1282797	0.68	1611	1.12	2613	3.64	10530	0.42	1041	9.94	57284
30	581.3	1251980	0.27	761	0.53	2042	19.52	48474	0.52	2300	121.50	722668
32	542.4	936019	0.40	1522	1.01	4845	21.48	65157	1.31	5712	97.90	641071
34	600.0	1039051	1.10	2636	3.22	8761	19.86	48837	1.60	4406	72.73	425718
36	600.0	1223864	1.40	3156	5.02	13606	59.70	131142	2.37	5707	171.14	938439
38	600.0	1027778	1.91	5053	12.56	26556	82.77	178170	3.51	10518	268.05	1211086
40	600.0	1447604	2.96	6648	10.27	27028	102.1	219454	6.40	18169	240.84	1220934

Table 2. Comparison of three SBDS implementations in Chuffed, static symmetry breaking in Chuffed, and LDSB in Eclipse, on the Graph Colouring problems

Uniform

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
30	140.7	282974	0.00	14	0.06	474	0.26	3049	0.02	277	19.63	56577
32	211.4	390392	0.00	17	0.00	146	0.24	3677	0.00	84	14.11	27178
34	213.9	272772	0.00	25	0.29	1182	3.53	11975	0.03	433	22.06	30127
36	195.9	296358	0.00	36	0.04	467	6.91	23842	0.01	200	35.66	85505
38	224.0	297138	0.00	55	0.04	516	23.55	69480	0.03	526	51.18	107574
40	250.9	423326	0.00	83	0.31	1879	21.07	78918	0.06	878	84.16	185707

Biased

Size	none		1UIP		crippled		choice		static		LDSB	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Nodes
20	13.25	39551	0.00	27	0.00	32	0.01	639	0.00	29	0.72	1376
22	11.53	63984	0.00	25	0.00	34	0.02	727	0.00	25	0.16	538
24	66.60	154409	0.00	35	0.00	47	0.07	1992	0.00	32	1.91	2114
26	74.77	277290	0.00	55	0.00	93	0.12	3385	0.00	104	9.56	34210
28	130.5	280649	0.00	62	0.00	84	0.58	6402	0.00	103	9.14	37738
30	267.6	480195	0.00	101	0.01	239	10.48	43835	0.01	359	57.18	215932
32	331.7	600772	0.01	232	0.24	1597	9.98	44216	0.16	1864	110.16	288707
34	219.9	387213	0.20	806	0.40	1946	10.26	47470	0.45	1730	64.14	165943
36	442.6	709888	0.01	317	0.04	857	27.39	113252	0.80	3226	152.62	327472
38	382.5	631403	0.10	798	1.01	5569	31.63	138787	4.12	9413	193.40	508164
40	465.6	531285	0.02	410	0.36	2660	24.68	91847	0.12	2133	196.02	476486

size. A timeout of 600 seconds was used. Instances which timeout are counted as 600 seconds. The results are shown in Tables 1 and 2. Chuffed none and Eclipse LDSB fails to solve many instances before timeout, and choice fails to solve a few instances. 1UIP, crippled and static all solve every instance in the benchmarks.

It is clear from the results that SBDS-1UIP is far superior to SBDS-choice. Comparison between SBDS-1UIP and SBDS-crippled shows that the additional symmetries that we can only exploit with SBDS-1UIP indeed gives us reduced search and additional speedup. Comparison with static shows that dynamic symmetry breaking can be superior to static symmetry breaking on appropriate problems. The comparison with LDSB shows that lazy clause solvers can be much faster than normal CP solvers, and that they retain this advantage when integrated with symmetry breaking methods.

The total speed difference between 1UIP and LDSB is up to 2 orders of magnitude for the Concert Hall problems and up to 4 orders of magnitude for the Graph Colouring problems. Most of this speedup can be explained by the dramatic reduction in search space, which is apparent from the node counts in the results table. The redundancies exploited by lazy clause solvers are different from those redundancies caused by symmetries, and it is very clear here that by exploiting both at the same time with SBDS-1UIP, we get much higher speedups than possible with either of them separately. It should also be noted that CHUFFED with symmetry breaking constraints is also quite competitive.

6 Conclusion

In this paper we have examined how we can extend SBDS to make use of the better nogoods generated by lazy clause solvers. We have built a prototype implementation combining SBDS with lazy clause generation, which we call SBDS-1UIP. The resulting system can exploit types of redundancies previously impossible to exploit, and can outperform LDSB by several orders of magnitudes on some problems.

Acknowledgments. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

References

1. T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference*, pages 93–107, 2001.
2. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 77–92, 2001.
3. I. Gent and B.M. Smith. Symmetry breaking in constraint programming. In *14th European Conference on Artificial Intelligence*, pages 599–603, 2000.
4. C. Mears. *Automatic Symmetry Detection and Dynamic Symmetry Breaking for Constraint Programming*. PhD thesis, Clayton School of Information Technology, Monash University, 2010.
5. O. Ohrimenko, P.J. Stuckey, and M. Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009.
6. A. Schutt, T. Feydy, P.J. Stuckey, and M. Wallace. Why cumulative decomposition is not as bad as it sounds. In I. Gent, editor, *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *LNCS*, pages 746–761. Springer-Verlag, 2009.

A Soft Constraint for Cumulative Problems with Over-loads of Resource

Alexis De Clercq,
supervised by Thierry Petit, Nicolas Beldiceanu, and Narendra Jussien

Mines-Nantes, LINA UMR CNRS 6241,
4, rue Alfred Kastler, FR-44307 Nantes, France.
{Alexis.De-Clercq,Thierry.Petit,Nicolas.Beldiceanu,
Narendra.Jussien}@mines-nantes.fr

Abstract. This paper deals with cumulative problems where the resource capacity may be over-loaded, using constraint programming. We propose a generic soft constraint and we show its use in several practical cases. We adapt to our constraint the most recent edge-finding filtering algorithm of the Cumulative constraint.

1 Introduction

Scheduling problems consist of ordering activities. In *cumulative scheduling*, each activity a_i requires for its execution the availability of a certain amount of renewable resource. a_i has an earliest starting time est_{a_i}, a latest starting time lst_{a_i}, an earliest completion time ect_{a_i} and a latest completion time lct_{a_i}. Given a set of activities Ω, $est_\Omega = \min_{j\in\Omega}(est_j)$, $lst_\Omega = \min_{j\in\Omega}(lst_j)$, $ect_\Omega = \max_{j\in\Omega}(ect_j)$, $lct_\Omega = \max_{j\in\Omega}(lct_j)$. In Constraint Programming, activities are represented by variables. Many problems can be encoded using the Cumulative constraint [1].

Definition 1. *Let A be a set of n non-preemptive activities. For each $a_i \in A$, $start[a_i]$ is the variable representing its starting point in time. Its domain is the interval $D(start[a_i]) = [est_{a_i}, lst_{a_i}]$. $dur[a_i]$ is its duration variable, s.t. $D(dur[a_i]) = [\underline{d}_i, \overline{d}_i]$. Variable $res[a_i]$ (height of a_i) represents the discrete amount of resource consumed by a_i, s.t. $D(res[a_i]) = [\underline{r}_i, \overline{r}_i]$. We assume here $\underline{r}_i \geq 0$.*

We have $ect_{a_i} = est_{a_i} + \underline{d}_i$ and $lct_{a_i} = lst_{a_i} + \overline{d}_i$. The *energy* of an activity is $e_{a_i} = \underline{r}_i * \underline{d}_i$. The energy of a set of activities $\Theta \subseteq A$ is $e_\Theta = \sum_{a_i\in\Theta} e_{a_i}$.

Definition 2. *Given one resource with a capacity limited by capa and a set A of n activities, at each point in time t the cumulated height h_t of the activities overlapping t is $h_t = \sum_{a_i\in A, start[a_i]\leq t<end[a_i]} \underline{r}_i$. The* Cumulative*$(A, capa)$ constraint* [1] *enforces C1: For each activity $a_i \in A, start[a_i] + dur[a_i] = end[a_i]$, and C2: At each point in time t, $h_t \leq capa$.*

In this article, we study the case of cumulative problems where the *time horizon* (maximum latest completion time among all activities) is fixed, so as some problems have no solution if no over-loads on the resource capacity are tolerated. In this case, since the user wants to obtain a solution, over-loads are accepted

providing that some constraints expressing when and how such over-loads appear are satisfied. In this context, Section 2 presents SoftCumulative, a generalization of the SoftCumulativeSum constraint [7], and some practical examples of problems. Section 3 presents a filtering technique for SoftCumulative, including an adaptation of the $O(kn\log(n))$ Vilím's Edge-finding algorithm [12].

2 SoftCumulative and its Applications

Definition 3. *Let A be a set of activities s.t. starting times are scheduled between time 0 and $m-1$, a maximum capacity capa and:*

- *A sequence $P = [p_0, \ldots, p_k]$ of $k+1$ distinct integers sorted in increasing order and s.t. $p_0 = 0$ and $p_k = m$, which defines a partition of $[0, m[$ in sub-intervals s.t. each pair $(j, j+1)$, $j < k$, defines an interval $[p_j, p_{j+1}[$.*
- *A sequence of integer cost variables $Cost = [cost_0, \ldots, cost_{k-1}]$ one-to-one mapped with $P \backslash \{p_k\}$, s.t. variable $cost_j$ corresponds to the interval $[p_j, p_{j+1}[$.*
- *A sequence of ideal capacities $IC = [ic_0, \ldots, ic_{k-1}]$ one-to-one mapped with $P \backslash \{p_k\}$, s.t. $\forall\ ic_j \in IC, ic_j \leq capa$.*
- *A variable obj, a flag* CTR $\in$ (SUM-MAX, SUM-SUM, MAX-MAX, MAX-SUM).

The SoftCumulative*$(A, capa, P, Cost, IC, obj,$*CTR*) enforces:*

- *C1 and C2 (see* Definition 2*).*
- *C3: A constraint on each variable $cost_j \in Cost$: If* CTR $=$ SUM-MAX *or* MAX-MAX*, it is $cost_j = \max(0, \max_{t \in [p_j, p_{j+1}[}(h_t - ic_j))$, otherwise it is $cost_j = \sum_{t \in [p_j, p_{j+1}[} \max(0, h_t - ic_j)$, with $j < k$.*
- *C4: An objective constraint depending on* CTR*: If* CTR $=$ MAX-MAX *or* MAX-SUM *it is $obj = \max_{j \in [0,k-1]} cost_j$, otherwise it is $obj = \sum_{j \in [0,k-1]} cost_j$*

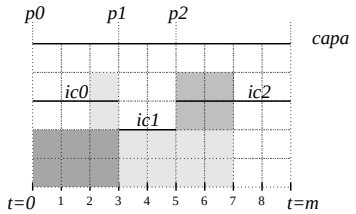


Fig. 1: SoftCumulative with 3 fixed activities: a_1 starts at 0 and ends at 3, $res[a_1] = 2$. a_2 starts at 2 and ends at 7, $res[a_2] = 2$. a_3 starts at 5 and ends at 7, $res[a_3] = 2$. If CTR is SUM-MAX then $cost_0 = 1, cost_1 = 0, cost_2 = 1$, $obj = 2$. If it is MAX-MAX then $cost_0 = 1, cost_1 = 0, cost_2 = 1$, $obj = 1$. If it is MAX-SUM then $cost_0 = 1, cost_1 = 0, cost_2 = 2$, $obj = 2$. If it is SUM-SUM then $cost_0 = 1, cost_1 = 0, cost_2 = 2$, $obj = 3$.

Figure 1 shows a SoftCumulative with three fixed activities and over-loads in the first and in the third interval of the partition of $[0, 9[$ given by $P = [0, 3, 5]$.

It is possible to add practical constraints on variables in $Cost$ to control over-loads over ideal capacities IC. Consider for instance cumulative problems where time unit is one hour, intervals defined by P have a length of one day (*e.g.*, 8 hours, except on Saturday, 4 hours) and the resource is the salaries, electricity or maintenance cost. For each day j the available budget is defined by ic_j.

$\diamond$ *Smooth changes w.r.t. over-loads* A frequent requirement consists of limiting the number of huge changes w.r.t. over-loads. It can be expressed by adding to SoftCumulative some instances of Smooth(N, tol, X) [2], where N is a variable, tol an integer and $X = [x_0, \ldots, x_n]$ a sequence of variables, which imposes that the number of times that $|x_{i+1} - x_i| > tol$ is equal to the value of N. In our case, X is $Cost$. In practice, for example, when a company hires extra-employees, their number should not vary too much from one period to another.

$\diamond$ *Fair distribution of over-loads* The fair distribution of over-loads, for instance during one month, can be expressed by adding to SoftCumulative some balancing constraints [6, 9] on variables in $Cost$, or by using cardinality constraints [8] (for instance, at most one over-loaded day each week).

$\diamond$ *Dependencies w.r.t. over-loads* In some problems, dependencies exist w.r.t. over-loads in different days. For instance, one may impose that if the maximum over-load on May 18th is > 3, then the maximum over-load on May 19th should be ≤ 1. Dependencies can be encoded by primitive constraints on pairs of variables in $Cost$ or, if they are more complicated, thanks to an automaton [5].

3 Filtering algorithm for SoftCumulative

The objective obj, the sequence $Cost$ and the set A are variables. We describe bound-propagation techniques for these variables. A fail is detected when a domain has a lower-bound strictly greater than its upper-bound. For sake of space, we focus in this section on sum-max, the most useful parameter ctr for SoftCumulative. Variable obj is pruned according to the objective constraint $obj = \sum_{j \in [0,k-1]} cost_j$, which can be also back-propagated on variables in $Cost$.

3.1 Pruning cost variables

Similarly to SoftCumulativeSum [7], minimum values of domains of variables in $Cost$ can be pruned according to the cumulative profile computed with *compulsory parts* of activities [4]. Similarly to the filtering algorithms of Cumulative and SoftCumulativeSum, *sweep* [3] can be used to compute this profile. Then, each minimum of a variable $cost_j \in Cost$ can be updated thanks to the difference between the maximum height of the profile within the interval corresponding to $cost_j$ and the value ic_j, iff such a difference is positive. W.r.t. maximum values of variables in $Cost$, they can be reduced either by external events (for instance the constraints Smooth, balancing constraints or automaton described in section 2), or thanks to a back-propagation of the objective constraint. Obviously, such reductions have a significant impact on the pruning of starting date variables described in next section.

3.2 Pruning starting dates from upper bounds of cost variables

This section presents a filtering technique inspired from the $O(kn\log(n))$ Edge-finding filtering algorithm proposed by Vilím [12] for Cumulative, in order to prune, within SoftCumulative, the variable $start[a_i]$ for each activity $a_i \in A$.

Principle of energetic reasoning The principle of energetic reasoning is to compare the resource necessarily required by a set of activities within a given interval of points in time with the available resource within this interval. In classical cumulative problems, this area of available resource within an interval $[t_s, t_e[$ of k points in time is equal to $k * capa$. In the context of SoftCumulative, we consider maximum values of variables in $Cost$ instead of $capa$. W.l.o.g., the considered intervals are not necessarily the intervals given by the set P. Thus, we define formally the free area of an arbitrary interval w.r.t. the ideal capacities and to the maximum cost on each interval of P.

Notation 1 *Given a* SoftCumulative$(A, capa, P, Cost, IC, obj, \text{CTR})$*, t_s and t_e two time points in $[0, m[$ s.t. $t_s < t_e$, we denote by $succ(t_s)$ the smallest index $i \in [0, k]$ such that p_i is greater than or equal to t_s and $prev(t_e)$ the greatest index $j \in [0, k]$ such that p_j is strictly less than t_e. Given $p_j \in P$, we use the notation $capa_j = \min(capa, ic_j + \max(D(cost_j)))$.*

Definition 4. *Given t_s and t_e in $[0, m[$ s.t. $t_s < t_e$, $Area_{(t_s,t_e)}$ is the number of free resource units in $[t_s, t_e[$ according to the upper bounds of variables in Cost:*

- *If $prev(t_e) < succ(t_s)$ then $Area_{(t_s,t_e)} = (t_e - t_s) * capa_{prev(t_e)}$.*
- *If $prev(t_e) \geq succ(t_s)$ then $Area_{(t_s,t_e)} = (succ(t_s) - t_s) * capa_{succ(t_s)-1} + (t_e - prev(t_e)) * capa_{prev(t_e)} + \sum_{j \in [succ(t_s), prev(t_e)-1[}(p_{j+1} - p_j) * capa_j$.*

Vilím's edge finding algorithm [12] is divided into two phases. First, it detects precedences among activities, and then it prunes starting dates of activities. These two phases use the notion of *energy envelope*. In the case of SoftCumulative, given a set of activities Θ[1], the energy envelope is computed according to the maximum values of variables in $Cost$, instead of $capa$.

Definition 5. *Let Θ be a set of activities s.t. $\Theta \subseteq A$. The* energy envelope *of Θ is: $Env(\Theta) = \max_{\Omega \subseteq \Theta}(Area_{(p_0, est_\Omega)} + e_\Omega)$.*

Detecting precedence relations We use the definition given in [12].

Definition 6 (Precedences [12]). *An activity $a_i \in A$ "ends before the end" of an activity $a_j \in A$ iff, in all solutions $end[a_i] \leq end[a_j]$. It is denoted by the relation $a_i \lessdot a_j$. It can be extended to a set of activities: a set of activities Θ can end before the end of an activity a_j in the same way. We can write $\Theta \lessdot a_j$.*

To detect precedences w.r.t. an activity a_j, we consider all the activities a_i having a latest completion time greater than the latest completion time of a_i.

[1] Since our reasoning is exclusively based on energy, determining the relevant set Θ will be identical to the Cumulative case [12], both for detecting precedences and pruning start variables of activities.

Definition 7. *The* Left Cut *of A by an activity $a_j \in A$ is a set of activities such that: $LCut(A, a_j) = \{a_i, a_i \in A \ \& \ lct_{a_i} \leq lct_{a_j}\}$.*

Given a set Θ of activities in A (which should be a Left Cut), the energy envelope is used to compute a lower bound $lb(ect_\Theta)$ for the earliest completion time ect_Θ of Θ. In [12], this point in time is simply: $lb(ect_\Theta) = \lceil Env(\Theta)/capa \rceil$. In the case of SoftCumulative, it is also possible to compute this lower bound. We can determine the greatest index in variables $Cost$ corresponding to a point in time less than or equal to $lb(ect_\Theta)$.

Definition 8. *Given a* SoftCumulative$(A, capa, P, Cost, IC, obj,ctr)$*, $P = [p_0, \ldots, p_{k-1}]$, and x a positive integer, $inf(x)$ is the greatest index $j \in [0, k-1]$ s.t. $x \leq Area_{(p_0,p_j)}$ (if it exists).*

Proposition 1. *Given a Left Cut Θ, a lower bound of ect_Θ is:*

$$lb(ect_\Theta) = p_{inf(Env(\Theta))} + \left\lceil \frac{Env(\Theta) - Area_{(p_0, p_{inf(Env(\Theta))})}}{capa_{inf(Env(\Theta))}} \right\rceil$$

Proof. (Sketch) $Env(\Theta) - Area_{(p_0,p_{inf(Env(\Theta))})}$ is the energy that remains from $Env(\Theta)$ when all the free area between p_0 and $p_{inf(Env(\Theta))}$ is filled. By dividing this quantity by $capa_{inf(Env(\Theta))}$, we obtain the smallest number of points in time to add after $p_{inf(Env(\Theta))}$. □

Vilím introduced in [11], [12] a tree data structure to efficiently compute the energy envelope $Env(\Theta)$ for a given set $\Theta \in A$, the Θ-tree. In a Θ-tree, leaves are the activities sorted by their earliest starting dates ($est_{a_i}, \forall i \in A$). Each node v of the Θ-tree holds two values (see [10], [12] for their computation) : (1) e_v, the total energy contained in that node, that is, the sum of the energies of direct descendants. (2) Env_v, the total energy envelope of the set of descendants leaves. The formulas given in [12] do not need to be adapted to our case because they are based on energy in each node. The capacities are not involved in the computation. Once the energy envelopes are available, from [12], it is possible to detect precedences thanks to the following rule: $lb(ect(LCut(A, a_j) \cup \{a_i\})) > lct_{a_j} \Rightarrow LCut(A, a_j) \lessdot a_i$. Finally, the algorithm used to compute precedence detections uses "$\Theta - \Lambda$-tree" to obtain a precedence list for each activity with at least one activity that ends before its end. It is an extension of the Θ-tree. The principle is the same, and the formulas can be directly re-used in our framework.

Pruning the starting date variables An adjustment can be made on the earliest starting time for each activity a_i that ends necessarily after the end of a set of activities Θ. Θ can only overlap the interval $[est_\Theta, lct_\Theta]$, by definition. If it overlaps more than the area available while a_i is processed then the remaining area should be placed before a_i, so that est_{a_i} can be updated. The earliest starting time can only be revised if we consider a set Θ that can compete[2] with

[2] Θ *competes* with a_i iff, within $[est_\Theta, lct_\Theta]$, reducing the available resource by the resource necessary for a_i makes that e_Θ does not fit in this interval.

a_i (that is : $e_\Theta > Area_{(est_\Theta, lct_\Theta)} - \underline{r}_i * (lct_\Theta - est_\Theta)$). In the original algorithm [12], the revised earliest starting time of a_i is:

$$est_{a_i} = est_\Theta + \left\lceil \frac{e_\Theta - (capa - \underline{r}_i)(lct_\Theta - ect_\Theta)}{\underline{r}_i} \right\rceil$$

In the case of SoftCumulative, we have to consider $Area_{(est_\Theta, lct_\Theta)}$. The revised earliest starting time of a_i is:

$$est_{a_i} = est_\Theta + \left\lceil \frac{e_\Theta - Area_{(est_\Theta, lct_\Theta)} + \underline{r}_i * (lct_\Theta - est_\Theta)}{\underline{r}_i} \right\rceil$$

4 Conclusion and Perspectives

We presented SoftCumulative, which generalizes the SoftCumulativeSum [7]. We showed some practical applications of this constraint. We adapted the $O(kn\log(n))$ Edge-finding filtering algorithm proposed by Vilím [12] for Cumulative to SoftCumulative. Currently, as it is usually done for soft global constraints, we do not prune the starting time variables of activities from events occurring on the *lower* bounds of domains of variables in $Cost$, although it may be useful (*e.g.*, w.r.t. the example with Smooth in Section 3). Our perspectives are to include that pruning and to experiment our constraint.

References

1. A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathl. Comput. Modelling*, 17(7):57–73, 1993.
2. N. Beldiceanu and M. Carlsson. Revisiting the cardinality operator and introducing the cardinality-path constraint family. *Proc. ICLP*, 2237:59–73, 2001.
3. N. Beldiceanu and M. Carlsson. A new multi-resource *cumulatives* constraint with negative heights. *Proc. CP*, pages 63–79, 2002.
4. A. Lahrichi. The notions of Hump, Compulsory Part and their use in Cumulative Problems. *C.R. Acad. sc.*, t. 294:20ç–211, 1982.
5. G. Pesant. A regular language membership constraint for finite sequences of variables. *Proc. CP*, 3258:482–495, 2004.
6. G. Pesant and J.-C. Régin. Spread: A balancing constraint based on statistics. *Proc. CP*, pages 460–474, 2005.
7. T. Petit and E. Poder. Global propagation of side constraints for solving over-constrained problems. *To appear in the Annals of 0perations Research*, 2010.
8. T. Petit, J-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. *Proc. IEEE-ICTAI*, pages 358–365, 2000.
9. P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. The deviation constraint. *Proc. CPAIOR*, 4510:260–274, 2007.
10. P. Vilím. Max energy filtering algorithm for discrete cumulative resources. *Proc. CPAIOR*, 5547:294–308, 2009.
11. Petr Vilím. *Global Constraints in Scheduling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, August 2007.
12. Petr Vilím. Edge finding filtering algorithm for discrete cumulative resources in O($kn\log(n)$). *Proc. CP*, pages 802–816, 2009.

Optimal Stopping rule-based algorithms for Computing Sub-Optimal solutions in Satisfiability problems with Preferences

Student: Emanuele Di Rosa[1].
Supervisors: Enrico Giunchiglia[1] and Barry O'Sullivan[2]

[1] DIST, Università di Genova, Viale Causa, 13 – 16145 Genova, Italy
{emanuele,enrico}@dist.unige.it
[2] Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
b.osullivan@4c.ucc.ie

Abstract. Satisfiability problems with preferences allow to enrich the expressive power of the Boolean Satisfiability problem (SAT) and make it able to handle qualitative/quantitative preferences on literals/formulas, defining an optimization problem. In some cases, it is not strictly necessary to compute an optimal solution, but it is enough to compute a sub-optimal solution, i.e. of high quality, and, possibly, provide a lower bound on the probability to find an optimal solution. The $\frac{1}{e}$-*rule* is the optimal stopping rule for the secretary problem, and guarantees to find an optimal solution with probability at least $\frac{1}{e}$. In this paper, (1) we show how to apply the $\frac{1}{e}$-*rule* for solving satisfiability problems with preferences; (2) we show that its theoretic success percentage of about 37% is greater than 90% on random benchmarks; (3) we show that the performance of the $\frac{1}{e}$-*rule* on structured benchmarks is sometimes many order of magnitude worse than that of complete search-based algorithms, and we explain the reasons why. (4) We propose an algorithm approximating the idea underlying the $\frac{1}{e}$-*rule*, which needs the generation of just two solutions: the experimental evaluation shows that the average success percentage of the proposed algorithm approximates very well the theoretic one of the $\frac{1}{e}$-*rule*, since it is about 48.33% on randomly generated instances.

1 Introduction

Boolean satisfiability (SAT) solvers return one assignment satisfying the input set of clauses, assuming at least one exists. However, in many cases, it is not enough to compute assignments satisfying all the input clauses: Indeed, the returned assignments have also to be "optimal" i.e. they have to satisfy as many other constraints –expressed as preferences– as possible. Satisfiability problems with preferences allow to enrich the expressive power of the SAT problem and make it able to handle qualitative/quantitative preferences on literals/formulas, defining an optimization problem. Three complete search-based approaches have been presented in order to solve these kinds of problems: OPTSAT-HS [2], OPTSAT-BF [3], and SAT&PREF [4]. Even though complete search-based approaches guarantee that the returned solution is optimal, they may require too much computation time to return an optimal solution (OPTSAT-HS modifies the SAT solver

heuristic and may have an exponential degradation in its performance [3]) or may generate an exponential number of intermediate solutions before finding the optimal one (as OPTSAT-BF and SAT&PREF). Moreover, in some cases, we do not necessarily need an optimal solution, but we only want to compute a sub-optimal solution of high quality by applying the simplest method, and, if possible, providing guarantees about a lower bound on the probability to find an optimal solution. The $\frac{1}{e}$-*rule* is the optimal stopping rule for the secretary problem, and guarantees to find an optimal solution (the best secretary) in at least the 37% of the cases [1]. It is constituted by two phases. In the former phase, it interviews and rejects $\frac{n}{e}$ applicants (where n is the known total number of applicants), and keeps the score related to the best one interviewed so far: this score becomes a lower bound concerning the quality of the next secretary to interview. In the latter phase, it interviews and rejects secretaries until the one with a score greater than the lower bound is found.

In this paper: (1) we show how to apply the $\frac{1}{e}$-*rule* for solving satisfiability problems with preferences, which does not require any modification in the internals of SAT solvers (as in OPTSAT-HS and in SAT&PREF), neither to add new formulas to improve the quality of the last solution computed (as in OPTSAT-BF and in SAT&PREF); (2) we show that its theoretic success percentage of about 37% is greater than 90% on random benchmarks; (3) we show that the performance of the $\frac{1}{e}$-*rule* on structured benchmarks is sometimes many order of magnitude worse than that of complete search-based algorithms, and we explain the reasons why. (4) We propose an algorithm approximating the ideas underlying the $\frac{1}{e}$-*rule*, which needs the generation of just two solutions: the first one approximates the first phase of the $\frac{1}{e}$-*rule*, while the second solution approximates the second phase. The experimental evaluation shows that the average success percentage of the proposed algorithm approximates very well that of the $\frac{1}{e}$-*rule*, since it is about 48.33% on randomly generated instances.

2 Satisfiability problems with Preferences

In three papers [2–4], complete search-based algorithms to compute an optimal solution w.r.t. a given set of qualitative preferences, expressed as a partially ordered set of literals $\langle S, \prec \rangle$, have been shown: Intuitively, S represents the preferences that we would like to have satisfied, $\prec$ models their relative importance, and a model μ of a formula φ (i.e. an assignment satisfying φ) is optimal if it is a minimal element of the partial order on the models of φ induced by $\langle S, \prec \rangle$ (see [2]). For example, let M, B, and C be Boolean variables for *Moto*, *Bike*, and *Car*, respectively; the preference $\langle \{M, B, \overline{C}\}, \{M \prec B\} \rangle$ models the fact that we would like to have both *Moto* and *Bike*, and that we would like to avoid *Car*; moreover, having *Moto* is more important than having *Bike*. If we are given a set φ of constraints imposing that we can buy exactly one among *Moto*, *Bike*, and *Car*, the resulting partial order on the models of φ induced by the preference is $\{M, \overline{B}, \overline{C}\} \prec \{\overline{M}, B, \overline{C}\} \prec \{\overline{M}, \overline{B}, C\}$ and the optimal model is $\{M, \overline{B}, \overline{C}\}$. Since qualitative/quantitative preferences on literals/formulas and also their mixing, can be reduced to the basic framework of qualitative preferences on literals [2], in the following, for simplicity, we consider only these kinds of problems.

3 The Secretary Problem and the $\frac{1}{e}$-*rule*

In the secretary problem there is a single secretarial position to fill, and n applicants for the position; the value of n is apriori known. The applicants are interviewed sequentially in a random order, with each order being equally likely. After each interview, the applicant can be compared with the previous ones, and has to be accepted or rejected. The decision to accept or reject an applicant can be based only on the relative ranks of the applicants interviewed so far, and rejected applicants cannot be recalled. The object is to select the best applicant. The $\frac{1}{e}$-*rule* is the optimal stopping rule for the secretary problem, and says to interview and reject the first $\frac{n}{e}$ applicants (where e is the base of the natural logarithm), and accept the next applicant who is better than all the applicants previously interviewed. As shown in [1], as n goes to infinite, the probability of selecting the best applicant from the pool goes to $\frac{1}{e}$, i.e. about 37% of the times: it provides a useful theoretic lower bound on the success probability.

4 Applying the $\frac{1}{e}$-*rule* for Computing Sub-Optimal solutions in Satisfiability problems with Preferences

In order to apply the $\frac{1}{e}$-*rule* to satisfiability problems with preferences, we need (i) to count the exact number n of models of the input formula before starting the algorithm (e.g. by the exact model counter cachet [5]); (ii) a random heuristic to choose decision variables to assign and a random polarity (these are default options in modern sat solvers) to obtain at each generation a different model, and simulate a random order on models; (iii) a test that enables us to say if the new model found is better than our current best model found with respect to preferences. Last point is carried out because of the *preference formula* ψ for a model μ wrt $(S, \prec)$, presented in [3]. An assignment μ' is preferred to μ wrt $(S, \prec)$ iff μ' satisfies the *preference formula*. In figure 1 we report the function $\frac{1}{e}$-*rule* that applies the $\frac{1}{e}$-*rule* to satisfiability problems with qualitative preferences on literals. It returns FALSE if the input formula φ is unsatisfiable, returns TRUE if it exists at least a model of φ, and, in this case, prints the model that is better than the best model found within the generation of $\frac{n}{e}$ models. In figure 1 we used the following functions:

- *countModels* returns the exact number of models of the input formula φ.
- *generateModel* returns one model of φ, if it exists, by using any SAT solver as a black box, which uses both random heuristic and random polarity.
- *isBetter* returns TRUE if model μ is better than μ_B wrt preferences; FALSE otherwise. For this purpose, it is enough to add the clauses of the formula ψ in φ and all literals in μ_B (as unit clauses), in an empty instance of any SAT solver; then we perform a simple Boolean constraint propagation.
- *BlCl* returns a blocking clause, i.e. a clause containing the negated version of all literals in μ, that allows to cut the last generated model μ off from φ.

With respect to the standard rule we keep μ_B as a way to know the quality of the best model generated so far: this is due to the fact that in this framework it is easier and more practical to directly compare two models, instead of calculating a score on them, and then compare them according to that score. Notice that all functions but *isBetter* are already implemented and available.

function $\frac{1}{e}$-rule(φ,S,$\prec$)
1 $n = countModels(\varphi)$;
2 $\mu_B = generateModel(\varphi)$;
3 **if** $(\emptyset \in \mu_B)$ **return** FALSE; // φ is $unsatisfiable$
4 **return** $\frac{1}{e}$-ruleCore(φ,S,$\prec$,$\lceil \frac{n}{e} \rceil - 1$,$\mu_B$,FALSE)

function $\frac{1}{e}$-ruleCore(φ,S,$\prec$,$numGen$,μ_B,$newBest$)
5 $\mu = generateModel(\varphi)$;
6 **if** $(\emptyset \in \mu)$ Print(μ_B); **return** TRUE; // μ_B is $Optimal$
7 **if** $(isBetter(\mu, \mu_B, S, \prec))$ $\mu_B = \mu$; $newBest$ = TRUE;
8 **if** ($numGen <= 0$ **and** $newBest$) Print(μ); **return** TRUE;
9 **return** $\frac{1}{e}$-ruleCore($\varphi \cup BlCl(\mu)$,S,$\prec$,$numGen$-1,μ_B,FALSE)

Fig. 1. The $\frac{1}{e}$-*rule* for solving satisfiability problems with qualitative preferences on literals.

5 The algorithm approximating the $\frac{1}{e}$-*rule*

Even though the $\frac{1}{e}$-*rule* is easy to apply and does not require any modification in the SAT solvers, it has two main drawbacks: (1) the computation of the exact number of models of the input formula before the start of the "core" algorithm; (2) the generation of at least $\frac{n}{e}$ models. Intuitively, the reason why the first phase of the $\frac{1}{e}$-*rule* generates at least $\frac{n}{e}$ models is based on the idea that by generating so many models, it is likely to find a model that has a good quality; in such a way, it is possible to use the information (or the score) associated to its quality, as a lower bound for the quality of the models that will be generated in the second phase. In such a way, when is generated a model that is better than the best model found within $\frac{n}{e}$ generations, it is likely to be optimal. The proposed algorithm takes advantage from the result presented in [4]: imposing to the sat solver heuristic to assign the polarity according to the preferences, the first solution computed has a high quality. Moreover, the SAT&PREF does not present the drawbacks of the OPTSAT-HS approach, and limits the ones of the OPTSAT-BF approach. Of course, it is still possible for SAT&PREF the generation of exponentially many intermediate models before finding an optimal solution. Thus, we propose an approximated algorithm (APPRAL from here on) that is a slightly modified version of the SAT&PREF algorithm, where only the first two models of the SAT&PREF algorithm are generated. Our hypothesis is that, the generation of the first model by SAT&PREF approximates the generation of the $\frac{n}{e}$ models in the $\frac{1}{e}$-*rule* (the first phase), and that the generation of the second model approximates the second phase of the $\frac{1}{e}$-*rule*.

6 Experimental Analysis

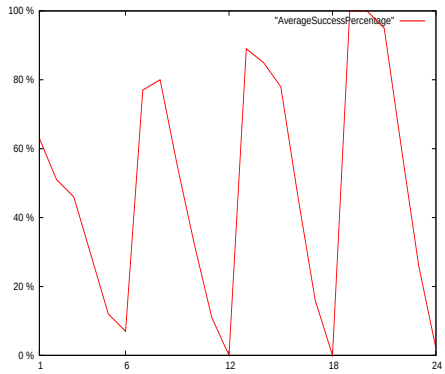
We started our evaluation considering 100 randomly generated 3-SAT formulas with 60 variables and 240 clauses, and 100 randomly generated preferences $(S, \prec)$, with $\prec = \emptyset$ and three kinds of polarity for the literals in the preference: random, all positive and all negative, respectively (indicated with R, P, and N

in Table 2(a)). We counted the exact number of models by Cachet [5] and the number of optimal models wrt the preferences as in [2], for all formulas: the average percentage to find an optimal model using a random algorithm is about 0.87%, 1.32%, and 1.03%, for random, positive, and negative polarity, respectively. The results reported in Table 2(a) are obtained running the experiments on a Linux box equipped with a Pentium IV 3.2GHz processor and 1GB of RAM, and show that both the $\frac{1}{e}$-*rule* and ApprAl compute an optimal solution with a high percentage (about 90%). However, the $\frac{1}{e}$-*rule* needs to compute always at least 2345.12 solutions (ANM column, i.e. average number of models) before finding an optimal solution, while ApprAl needs, on average, less than 2 solutions: this confirms our initial hypothesis. Notice that for the $\frac{1}{e}$-*rule* each instance has been run 10 runs, since it uses a random heuristic and a random polarity: in Table 2(a), * indicates that an optimal solution is found for all runs; $\hat{}$ indicates that an optimal solution is found at least 6 out of 10 times. Moreover, we ran the $\frac{1}{e}$-*rule* on structured benchmarks representing 21 Partial Min-ONE problems as in [3], where complete search-based algorithms compute an optimal solution for all instances and in a average time of less than 15 seconds. On these structured instances the most of the computation resources used by the $\frac{1}{e}$-*rule* are dedicated to the preliminary step to count models by Cachet: we obtained 10 timeout (set to 200 seconds) and 4 memory-out (set to 1GB) out of 21 instances, only in the counting process; when it terminates within the timeout, there are even three order of magnitude wrt the best complete solver. Instead, ApprAl managed to find an optimal model in 19 out of 21 instances. In order to test the success percentage of the algorithms on instances with $\langle S, \prec \rangle$ of increasing size, we considered the 2400 randomly generated instances with 200 variables and 800 clauses, as in [4]. We remember that $\prec$ is the transitive closure of the directed acyclic graph (DAG) whose nodes are the literals in S and with an arc between two nodes with probability $\rho = 0, T/2, T, 2T, 4T, 1$, where $T = \ln(|S|)/|S|$: Fixing $\rho = 0$ generated problems with an empty partial order, while increasing ρ corresponds (on average) to increasing $| \prec |$ up to the point in which $\rho = 1$, corresponding to a totally ordered set of preferences. The average success percentage to find an optimal solution for ApprAl, calculated on all 2400 instances, is about 48.33%; in particular, the 58.36% of the times the first solution computed is already optimal, while the second solution computed is optimal in the remaining 41.64%. We analyzed in more detail the success percentage obtained, and the results are shown in Figure 2. Here, on the x-axis we have 24 points, each corresponding to a pair $\langle |S|, \rho \rangle$: The pairs are first ordered according to $|S|$ and then to ρ. Thus, the first point has $|S| = 25\%$ of the variables, and $\rho = 0$, the second point has again $|S| = 25\%$ of the variables but $\rho = T/2$, and analogously for the others. Thus, points 6, 12, 18, 24 have $\rho = 1$, i.e., they are the points corresponding to a totally ordered set of preferences. The y-axis is the average success percentage for ApprAl on the 100 instances having the same $\langle |S|, \rho \rangle$: all instances are solved within the timeout, fixed to 600 seconds. Notice that, *only* for the point 19 of the x-axis, the first solution computed is guaranteed to be optimal (and the 100% is reached): all variables

are in the set S of preferences, $\prec$ is empty, and the heuristic of the SAT solver assigns decision variables according to the polarity specified in the preferences. Nonetheless, the approximated algorithm reaches a success percentage greater than 60%, especially for points corresponding to $\rho = 0, T/2$.

	Algo	Succ.Perc.		T	ANM
		*	^	(sec)	*
R	$\frac{1}{e}$-*rule*	89%	96%	0,29	2394,09
	ApprAl	87%	-	0,00	1,22
P	$\frac{1}{e}$-*rule*	89%	98%	0,28	2345,12
	ApprAl	89%	-	0,00	1,19
N	$\frac{1}{e}$-*rule*	90%	97%	0,29	2499,51
	ApprAl	94%	-	0,00	1,18

(a) A comparison between the $\frac{1}{e}$-*rule* and ApprAl on 100 random 3-SAT formulas and preferences with empty partial order.



(b) The average success percentage for 2400 random instances and $\langle S, \prec \rangle$ of increasing size.

Fig. 2. Results of the average success percentage for randomly generated 3-SAT instances.

7 Future work

In the future work we will analyse the amount of the cpu-time saved and the success percentage on structured problems, with respect to the complete approaches. Moreover, we plan to apply the $\frac{1}{e}$-*rule* in dynamic environments, e.g. in sat-based planning with uncertainty on the initial state.

References

1. F. T. Bruss, 'A Unified Approach to a Class of Best Choice Problems with an Unknown Number of Options', in *The Annals of Probability*, Vol. 12, No. 3, 1984, pp. 882–889.
2. E. Di Rosa, E. Giunchiglia, and M. Maratea, 'Solving Satisfiability Problems with Preferences', accepted to the *Journal Constraints*, in press and available here: http://www.star.dist.unige.it/~emanuele/Data/10constraints.pdf.
3. E. Di Rosa, E. Giunchiglia, and M. Maratea, 'A new approach for solving satisfiability problems with qualitative preferences', in *Proc. ECAI'08*, pp. 510–514.
4. E. Di Rosa, E. Giunchiglia, and B. O'Sullivan, 'Combining approaches for solving Satisfiability problems with Preferences and their Evaluation', available here: http://www.star.dist.unige.it/~emanuele/Data/10DiRosa_Giunchiglia_OSullivan.pdf.
5. T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi, 'Combining component caching and clause learning for effective model counting', in *Proc. SAT'04*.

Synthesis of Search Algorithms from High-level CP Models

Samir A. Mohamed Elsayed (Student), Laurent Michel (Advisor)

Computer Science Department, University of Connecticut.

Abstract. The ability to specify CP programs in term of a declarative model and a search procedure is a central feature primarily responsible for the industrial CP successes. However, writing search procedures is often difficult for novices or people accustomed to mathematical programming tools where this step is absent. Several attempts have been made to produce generic black-box searches that would be suitable for the vast majority of benchmarks. This paper offers an alternative viewpoint and argues for the synthesis of a search from the declarative model to exploit the problem instance structures. The intent is not to eliminate the search altogether. Instead, it is to have a default that performs adequately in the majority of cases while retaining the ability to write full-fledged procedures for experts. Preliminary empirical results demonstrate that the approach is viable delivering search procedures approaching and sometimes rivaling hand-crafted code produced by experts.

1 Introduction

Constraint programming (CP) techniques are successfully used in various industries and quite successful when confronted with hard constraint satisfaction problems. Parts of this success can be attributed to the considerable amount of flexibility that arise from the ability to write completely custom search procedures. Indeed, constraint programming has often been described from the basic belief that

$$CP = Model + Search$$

where the model is responsible for providing a declarative specification of the constraints that solutions of the problem must satisfy and the search is a specification of how to explore the search space to produce such a solution. In a number of languages designed for constraint programming, the search can be quite sophisticated. It can often concisely specify variable and value selection heuristics, search phases [5], restarting strategies [3], large neighborhood search [1], exploration strategies like depth-first-search, best-first search or limited discrepancy search [4] to name just a few.

This capability is in stark contrast with, for instance, mathematical programming where the search is a so-called *black-box* that can only be controlled through a collection of parameters affecting pre-processing, cut generation or

the selection of predefined global heuristics. Users of mathematical programming are accustomed to solely rely on modeling techniques and reformulations to indirectly influence and hopefully strengthen the search process effectiveness.

Unsurprisingly, many users of one technology often bring their baggages, habits and expectations when discovering a new technology like CP. Too often, newcomers overlook the true potential of open search specification and fail to exploit it. The observation prompted a number of efforts to rethink constraint programming tools and mold them after mathematical programming tools by *eliminating open search procedures* in favors of intelligent black-box procedures. Efforts of this type include [5] and [2] while others, e.g., [7] elected to provide a number of predefined common heuristics.

Our contention in this paper is that it is possible to get the best of both world, retaining the ability to write custom search procedures and *synthesizing* model-specific search procedures that are competitive with procedures handcrafted by experts. The central contribution of this paper is Cp-as, a model-driven automatic search procedure generator. Cp-as is written in Comet [9], an object-oriented programming language with support for constraint-programming at large and finite domains solving in particular. Cp-as analyzes a CP model instance at runtime, examining the variable declarations, the arithmetic and logical constraints as well as the global constraints to synthesize a procedure that is likely to perform reasonably well on this model instance. Cp-as is evaluated on a collection of CP models that require custom searches such as scene allocation, progressive party and warehouse allocation. The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 provides details about the synthesis process of Cp-as. Experimental results are reported in section 4. Finally, section 5 concludes.

2 Related Work

Minion [2] offers a black-box search and combines it with matrix based modeling, aiming for raw speed alone to produce 'model and run' solutions. The idea of deriving the search from the model appeared in [10] for CBLS. Aeon [6] is closely related and focuses exclusively on scheduling. Given a scheduling model specified in a high-level modeling language, Aeon recognizes and classifies its structures, and synthesizes an appropriate search algorithm. The classification result drives the selection of a search template. Aeon provides synthesizers for Constraint-based Scheduling (complete search) or Constraint-based local search (incomplete). Experimental results indicate that this approach may be competitive with state-of-the-art search algorithms. This paper extends this line of thinking with synthesis of search for *general non-scheduling* CP models.

3 The Synthesis Process

Cp-as, which is written in Comet generates search procedures for Comet models. Cp-as is defined in term of an extensible collection of rules meant to recognize

features of the model for which good heuristics are known. Each rule can issue a set of *recommendations* where a recommendation is characterized by a score indicating its fitness, a subset of variables to which it applies and three heuristics to be used for labeling these variables, namely: a variable, value and symmetry-breaking heuristic. CP-AS applies all the rules to a model to obtain a set of recommendations which fully specify the search procedure. Recommendations are applied in decreasing order of scores, hence if several recommendations apply to the same subset of variables, the highest-scoring one will take precedence. This section details this process.

3.1 Preliminaries

A Constraint Satisfaction Problem (CSP) is a triplet <X,D,C>, where X is a set of variables, D is a set of domains, and C is a set of constraints. Each $x_i \in X$ is associated with a domain $D_i \in D$. An assignment α associates a value $v \in D_i$ to each variable x_i, i.e., $\alpha(x_i) = v \in D_i$. A constraint $c \in C$, over variables $x_i, \cdots, x_j$, specifies a subset of the Cartesian product $D_i \times \cdots \times D_j$ indicating mutually-compatible variable assignments. A tuple $v = (v_i, \cdots, v_j)$ satisfies a constraint $c(x_i, \cdots, x_j)$ if $v \in c$. A solution α is a complete assignment that satisfies all the constraints. A Constraint Optimization Problem (COP) <X,D,C,f> is a CSP with an objective function f.

A variable selection heuristic h_x maps a subset of variables of cardinality k to a permutation of those variables: $h_x : 2^X \to \mathbb{N} \to X$. For instance, the familiar first-fail variable selection heuristic over a set of variables X returns a permutation function $\pi : \mathbb{N} \to X$ of the naturals $0..k-1$ that satisfies

$$\forall i, j \in 0..k-1 : i < j \Rightarrow |D_{x_{\pi(i)}}| \leq |D_{x_{\pi(j)}}|$$

A value selection heuristic h_v is a function that maps a set of finite values of cardinality k to a permutation: $h_v : 2^{\mathbb{Z}} \to \mathbb{N} \to \mathbb{Z}$. For instance, the min-value heuristic for x_i with domain D_i of cardinality k specified as $h_v(D_i)$ denotes the permutation function π satisfying

$$\forall a, b \in 0..k-1 : a < b \Rightarrow D_i(\pi(a)) \leq D_i(\pi(b))$$

A value symmetry breaking heuristic h_s is a function that maps a set of finite values of cardinality k to a subset of non-symmetric values: $h_s : 2^{\mathbb{Z}} \to 2^{\mathbb{Z}}$.

3.2 Rules and Recommendations

Given a CSP $M = \langle X, D, C \rangle$, a CP-AS rule $r \in \mathit{Rules}$ is a quadruple $\langle \mathcal{S}, \mathcal{P}, \mathcal{V}, \mathcal{H} \rangle$ where $\mathcal{S}$ is a scoring function $\mathcal{S} : M \to \mathbb{R}$ and $\mathcal{P}$ is the priority assigned to the rule. All scores are normalized in the 0..1 range with 1 representing the strongest fit. $\mathcal{V} : 2^X \to 2^X$ is a function that returns the subset of variables to be subjected to the search heuristic recommended by the rule. Finally, $\mathcal{H}$ is a triple of heuristic functions $\langle h_x, h_v, h_s \rangle$ as specified above.

A recommendation $\langle S, P, V, H \rangle$ results from applying a rule $r = \langle \mathcal{S}, \mathcal{P}, \mathcal{V}, \mathcal{H} \rangle$ to a CSP $M = \langle X, D, C \rangle$ and captures a score $S = \mathcal{S}(M)$, a priority $P = \mathcal{P}$, a subset of variables $V = \mathcal{V}(X)$, and a triple of heuristics $h_x(V), h_v, h_s$.

3.3 Rules Library

CP-AS rules are meant to exploit modeled structures and properties such as global constraints, domain sizes, or variables degree to name just a few.
Degree rule: Intuitively, the rule determines whether the static degree of variables in the constraint hyper-graph are sufficiently diverse. If the degrees are all very similar, a degree based heuristic is inappropriate. Conversely, a very diverse set of degrees indicates a a strong fit. Therefore, the rule defines

- $\mathcal{S} = \sigma(\{deg(x) : x \in X\})/\max_{x \in X} deg(x)$ where σ stands for standard deviation, and *deg(x)* for the static degree of x.
- $P = \mathcal{P}$
- $\mathcal{V}$ is simply the identity function (all variables are selected).
- $h_x(V)(i) = -deg(x_i)$
- h_v is the min-value heuristic
- h_s is the result of a symmetry breaking analysis. See [8].

3.4 Rules Composition & Calibration

Recommendations compose to derive an effective search procedure. CP-AS sorts the recommendations by score while breaking ties with priorities. The search template is shown in Figure 1. CP-AS iterates over the recommendations in lexicographic order of score and priority. Line 2 invokes the polymorphic labeling method *label* of the recommendation. Once all the variables are bound, the search ends in line 3. The search is complete as the set of variables it labels is $\bigcup_{r \in rec} \left(\cup_{x \in V(r)}\right) = X$.

```
1 forall(r in rec.getKeys()) by (−rec{r}.getScore(), −rec{r}.getPriority()) {
2     rec{r}.label();
3     if (solver.isBound()) break;
4 }
```

Fig. 1. A skeleton for a synthesized search procedure

Figure 2 depicts the implementation of *label* method of a *variable first recommendation*, i.e., a recommendation that first selects a variable and then chooses a value as opposed to a *value first recommendation* that selects a value and then chooses the variable to assign it to. Line 12 retrieves the variables the recommendation operates on and line 14 selects a variable according to the heuristic h_x of the recommendation. Line 16 retrieves the set of values to try for the chosen variable. Note that the *values* method encapsulates the use of value symmetry-breaking. If there are no exploitable value symmetries, the *values* method returns the complete domain for the variable *theVar*. The value selection is driven by the heuristic h_v on line 17 which embodies the value permutation adopted by the recommendation.

```
1  interface Recommendation {
2     var<CP>{int}[] getVars();
3     set{int} values();
4     set{int} unboundVars(var<CP>{int}[] x);
5     var<CP>{int} getVar(int rank);
6     int getValue(int rank);
7     int hx(var<CP>{int}[] x,int rank);
8     int hv(set{int} vals,int rank);
9  }
10 class VariableRecommendation implements Recommendation { ...
11    void label() {
12       var<CP>{int}[] x = getVars();
13       while(!bound(x)) {
14          selectMin(i in unboundVars(x))(hx(x,i)) {
15             var<CP>{int} theVar = getVar(i);
16             set{int} theVals = values(theVar);
17             tryall<solver>(t in theVals) by (hv(theVals,t))
18                solver.label(theVar,getValue(t));
19             onFailure solver.diff(theVar,getValue(t));
20          }
21       }
22    }
23 }
```

Fig. 2. A snippet of the labelVars method

4 Experimental Results

Preliminary experiments show the practicality of CP-AS on a range of problems. It compares synthesized, custom and first fail procedures on the same models. The custom procedures were taken from the state-of-the art. Table 1 reports the number of choices, the average CPU time (in ms) and its standard deviation.

Problem	#cp,**t** (FF)	#cp,**t**,σ (CP-AS)	#cp,**t**,σ (cust.)
Slab Mill	Too long time	883,**2772**,636	1656,**4755**,666
Scene Allocation	7754783, **1390186**	17615,**3233**,799	4331,**621**,88
Car Sequencing	Too long time	475,**557**,78	77,**172**,25
Perfect Square	Too long time	74,**196**,27	74,**182**,26
Progressive Party	Too long time	326,**163**,23	326,**158**,22
Warehouse	1645, **72**	106,**12**,6	29,**6**,8
SGP (8-4-7)	168, **298**	533,**514**,72	146,**146**,138
Sports Avenue	Too long time	11069,**10226**,1432	11062,**9471**,1326

Table 1. Results obtained using some benchmarks.

The selected benchmarks are well-known CSP and COP problems that use custom searches. All results were obtained from 50 runs with COMET 2.1 on 2.33GHz Intel Core Duo machine with 2GB RAM running Ubuntu 9.10. The results show that the synthesized search is often reasonably close to the custom

search procedures. For the scene allocation problem, the static-degree selected by the synthesizer is quite competitive with the custom search. Perhaps even more surprisingly, the synthesized search outperforms the custom search for the steel slab mill problem (the synthetic search uses a variable heuristic first driven by the weights of the knapsack, then by domain size). The overhead on other instances is due to the generic data structures manipulated by the templated search. In a nutshell, the synthesized searches are competitive to tailored algorithms offering a reasonable first search with no efforts beyond modeling.

5 Conclusion & Future Work

CP-AS is a framework to automatically generate search algorithms from high-level CP models. Given a COMET CP model, CP-AS recognizes and classifies its structure to synthesize an appropriate search algorithm. Preliminary empirical results indicate that the technique appears to be competitive with state-of-the-art procedures on several classic benchmarks.

While the approach demonstrates potential, work remains to augment the rule set. Ideally, rules should capture as many strategies as possible and recognize when they are applicable. Improvements to the composition mechanism, the symmetry breaking capabilities, as well as an ability to handle search strategies are needed. Finally, an in-depth empirical evaluation is absolutely essential.

References

1. R.K. Ahuja, Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.
2. I.P. Gent, C. Jefferson, and I. Miguel. Minion: A fast, scalable, constraint solver. In *ECAI 2006: 17th European Conference on Artificial Intelligence, August 29-September 1, 2006, Riva del Garda, Italy: including Prestigious Applications of Intelligent Systems (PAIS 2006): proceedings*, page 98. Ios Pr Inc, 2006.
3. C.P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of automated reasoning*, 24(1):67–100, 2000.
4. W.D. Harvey and M.L. Ginsberg. Limited discrepancy search. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 607–615, 1995.
5. SA ILOG. ILOG Concert 2.0.
6. J.N. Monette, Y. Deville, and P. Van Hentenryck. Aeon: Synthesizing scheduling algorithms from high-level models. *Operations Research and Cyber-Infrastructure*, pages 43–59, 2009.
7. G. Team. Gecode: Generic constraint development environment, 2006. *Available fro m http://www. gecode. org*.
8. P. Van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Compositional derivation of symmetries for constraint satisfaction. *Abstraction, Reformulation and Approximation*, pages 234–247, 2005.
9. P. Van Hentenryck and L. Michel. *Constraint-based local search*. The MIT Press, 2005.
10. Pascal Van Hentenryck and Laurent Michel. Synthesis of constraint-based local search algorithms from high-level models. In *AAAI'07*, pages 273–278. AAAI Press, 2007.

Maintaining Multiple Representations in DCOP Solving

Patricia Gutierrez (student) and Pedro Meseguer (supervisor)

IIIA, Institut d'Investigació en Intel.ligència Artificial
CSIC, Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Spain.
{patricia|pedro}@iiia.csic.es

Abstract. For DCOP solving, maintaining soft arc consistency during search has been very beneficial. However, including higher levels of soft arc consistency breaks usual privacy requirements. To avoid this, we propose to keep different representations of the same problem on each agent, on which soft arc consistencies respecting privacy are enforced. Deletions caused in one representation can be legally propagated to others. Experimentally, this causes significant benefits.

1 Introduction

Recently, a number of methods for solving Distributed Constraint Optimization Problems (DCOP) have appeared [8–11] possibly as consequence of the rise of multiagent technology. In this problems variables and constraints are distributed into several agents and the task of interest is to find a global optimal assignment in a distributed way.

Distributed search solves DCOP by exploring the search space using messages. Lately, the BnB-ADOPT$^+$ algorithm has been enhanced with some forms of soft arc consistency maintenance (specifically, AC* and FDAC*) [2], which have been shown very beneficial for performance, saving an important number of exploratory messages. Moving into the next soft arc consistency level, EDAC*, we have found that its enforcement breaks the privacy requirements usually assumed in the distributed setting. With the double aim of improving as much as possible distributed search performance while respecting agent privacy, we present the following approach. To enforce FDAC* agents must be ordered. At each agent, we propose to keep several orderings among agents, with different cost function representations associated from which some values can be deleted. Interestingly, these deletions can be legally propagated among representations. Experimentally, our approach causes communication savings on the benchmarks tested.

2 Preliminaries

COP. A binary *Constraint Optimization Problem* (COP) is defined by $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of variables, $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ is a set of finite domains (x_i takes values in $D(x_i)$), and $\mathcal{C}$ is a set of unary and binary cost functions; $C \in \mathcal{C}$ specifies the cost of every combination of values of $var(C)$, $C : \prod_{x_i \in var(C)} D(x_i) \mapsto \mathbb{N} \cup \{0, \infty\}$. The cost of a complete tuple is the sum of all individual cost functions evaluated on that particular tuple. An *optimal solution* is a complete tuple with minimum cost. This definition assumes the weighted model of soft constraints [7].

Soft Arc Consistency. Considering a single COP: (i, a) means the value a of variable x_i, $\top$ is the lowest unacceptable cost, C_{ij} is the binary cost function between x_i and x_j, C_i is the unary cost function on x_i values, C_ϕ is a zero-ary cost function that represents a necessary global cost of any complete assignment. As [5, 1], we consider the following local consistencies (variables connected by a cost function are ordered):

- *Node Consistency**: (i, a) is node consistent* (NC*) if $C_\phi + C_i(a) < \top$; x_i is NC* if all its values are NC* and there is $a \in D_i$ such that $C_i(a) = 0$; a COP is NC* if every variable is NC*.
- *Arc consistency**: (i, a) is arc consistency (AC) with respect to cost function C_{ij} if there is $b \in D_j$ s.t. $C_{ij}(a, b) = 0$; b is a *simple support* of a; x_i is AC if all its values are AC with respect to every binary cost function involving x_i; a COP is AC* if every variable is AC and NC*.
- *Directional arc consistency**: (i, a) is directional arc consistent (DAC) with respect to cost function C_{ij}, $j > i$, if there is $b \in D_j$ such that $C_{ij}(a, b) + C_j(b) = 0$; b is a *full support* of a; x_i is DAC if all its values are DAC with respect to every C_{ij}, $j > i$; a COP is DAC* if every variable is DAC and NC*.
- *Full DAC**: a COP is FDAC* if it is DAC* and AC*.
- *Existential arc consistency**: Variable x_i is existential arc consistent (EAC) if there is at least one value $a \in D(x_i)$ such that $C_i(a) = 0$ and it has a full support in every cost function C_{ij}; a COP is EAC* if every variable is NC* and EAC.
- *EDAC**: a COP is EDAC* if it is FDAC* and EAC*.

AC*/FDAC* can be reached forcing simple/full supports and pruning values not NC*. Simple supports can be forced on every value by projecting the minimum cost from its binary cost functions to its unary costs, and projecting the minimum unary cost into C_ϕ. Full supports can be forced in the same way, but first it is needed to extend from the unary costs of neighbors to the binary cost functions the minimum cost required to perform in the next step the projection over the value. The systematic application of these operations (projection and extension) do not change the minimum cost nor the optimal solution [5]. When we prune a not NC* value from x_i we need to recheck AC*/FDAC* on every variable that x_i is constrained with, since the deleted value could be a simple/full support.

DCOP. A *Distributed Constraint Optimization Problem* (DCOP) is defined by a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{A}, \alpha \rangle$, where $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ define a COP, $\mathcal{A}$ is a set of agents, and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent. We assume that each agent holds one variable and cost functions are unary and binary only. Agents communicate through messages, which could be delayed but never lost, and they are delivered in the order they were sent.

BnB-ADOPT / BnB-ADOPT$^+$. BnB-ADOPT [11] is a reference algorithm for DCOP. It is a depth-first version of ADOPT [8], showing a better performance. As ADOPT, it arranges agents in a DFS tree. Each agent holds a context, which is a set of assignaments involving the agent's ancestors, and will be updated with message exchange. BnB-ADOPT uses three message types: VALUE(i, j, val, th) – i informs descendant j that i takes value val with threshold th– COST$(k, j, context, lb, ub)$ –k informs parent j that its bounds are lb/ub in $context$– and TERMINATE(i, j) –i informs child j that i terminates– (for detailed definitions of these concepts see [11]). A BnB-ADOPT agent

executes the following loop: it reads and processes incoming messages and takes value. Then, it sends a VALUE to each descendant and a COST to its parent.

BnB-ADOPT$^+$ [3] is a version of BnB-ADOPT that saves most of the redundant messages, causing substantial reductions in communication costs with respect to the original algorithm. BnB-ADOPT$^+$ keeps the optimality and termination of BnB-ADOPT. It stores the last VALUE and COST messages sent to each destination, and it checks if some VALUE or COST messages to be sent at the current iteration might be redundant. If messages are found redundant, they are not sent in most of the cases (for detailed explanation see [3]).

3 Connecting Distributed Search with Soft Arc Consistency

Combining search with soft arc consistency brings substantial benefits to search performance. Taking BnB-ADOPT$^+$ as the distributed search algorithm, its combination with AC* and FDAC* soft arc consistency levels [2] have provided very good results. Soft arc consistency is maintained during distributed search, initially in a preprocess step where soft arc consistency is assured, and then it is enforced during execution every time soft arc consistency is broken. AC*/FDAC* levels are achieved implementing the projection and extension operators for the distributed case, and pruning node inconsistent values on every agent. This is done in the following way. In addition to the VALUE, COST and TERMINATE messages of BnB-ADOPT$^+$, AC*/FDAC* enforcing requires two new message types: DEL(i, j, val) $-i$ notifies j that it has deleted $val-$ and UCO$(j, i, vectorOfExtensions)$ $-j$ informs i that it extends the unary costs of $vectorOfExtensions$ into the binary C_{ij}. Also, to enforce soft arc consistencies, some ordering is needed: we take the order in which agents appear in each branch of the DFS tree used by BnB-ADOPT$^+$. The resulting algorithms maintain BnB-ADOPT$^+$ optimality and termination, improving its performance: soft arc inconsistent values are removed from their domains, reducing the search space, which causes substantial reductions in the search effort. All value deletions considered during AC*/FDAC* maintenance are unconditional (a deleted value does not have to be restored at any time).

In the distributed case, it is usually assumed that each agents knows its variable and the cost functions that it has with other agents. This second assumption implies that it knows the domain of the variables it is constrained with. To enforce soft arc consistencies higher than NC*, it is required that if agent i is constrained with agent j by C_{ij}, i has to represent locally $D(x_j)$. For privacy reasons, we assume that the unary costs of the values of an agent are held by itself, who knows them and updates them accordingly. An agent neither can know nor update unary costs of other agents.

Maintaining AC*/FDAC* during distributed search requires each agent to know the binary cost functions in which it is involved and the unary costs of its values. These requirements are in agreement with the privacy requirements not permitting an agent to know the unary costs of values of other agents. However when moving to EDAC* (the next soft arc consistency level) this privacy requirement is broken. EDAC* maintenance requires that at each variable there is a value with unary cost 0 which is fully supported in both directions (cost functions linking ancestors with this variable, cost functions linking this variable with descendants). Let us consider two agents $i, j, i < j$ that share a

cost function C_{ij}. To assure that j has a value fully supported by i, i has to extend some of its unary costs into the binary ones, which will be projected on the unary costs of j values. However, i will only extend its unary costs if it is sure that from this operation C_ϕ will increase (otherwise termination is not guaranteed). But this condition can only be assured if i knows the unary costs of j[1]. Therefore, aiming at EDAC* maintenance breaks the natural privacy requirements explained above, which represents a serious drawback in the distributed environment.

A way to partially avoid this issue, while enforcing some soft arc consistency that prunes more than FDAC*, comes from the following fact. Observe that the first variable in a FDAC* ordering satisfies the EDAC* property: for FDAC* each value has a simple/full support and there is a value with cost 0 (for NC*); since it is the first variable in the ordering, these supports have to be full supports. This suggests an alternative way for the distributed setting: instead of having a single ordering of agents, we may have several orderings. On each ordering we enforce FDAC*, and the first variable on every ordering satisfies EDAC*. Next we show that having different orderings and propagating deletions among them is legal and does not compromise the correctness of this idea. However, enforcing FDAC* n times (with a different first variable at each order) is not necessarily as strong as maintaining EDAC* on all variables.

4 Multiple Representations

It is known that with different variable orderings FDAC* maintenance prune different values depending on the ordering used [4]. This fact motivates the present approach. It is unclear how to determine the *best ordering*, in the sense of the ordering that prunes most. Instead of looking for that ordering, we consider as alternative to keep *multiple orderings* $O_1, ..., O_r$ at each agent, on which FDAC* is separately enforced. Maintaining FDAC* in O_p may cause the deletion of value a of variable i: this deletion is propagated to all other orderings $O_1, ..., O_{p-1}, O_{p+1}, O_r$, that is, value a is also removed in the domain of variable i for $O_1, ..., O_{p-1}, O_{p+1}, O_r$.

Propagating value deletions among different orderings is legal. Let us assume that enforcing FDAC* on the ordering O_1 causes to delete value (i, a), while enforcing FDAC* on the ordering O_2 causes to delete value (j, b). Then, both values can be deleted without losing any optimal solution. If enforcing FDAC* using ordering O_1 we delete value (i, a), this means that value a for variable i will not appear in any optimal solution of the problem. This fact derives directly from soft arc consistency, and it is independent of the ordering used. The same situation happens with ordering O_2 and value (j, b). Therefore, it is legal to remove both values from their domains, independently of the ordering used. Since cost functions evolve depending on the ordering used, we prefer to talk about different representations of cost functions instead of different orderings (clearly, each ordering defines a representation).

The idea of multiple representations can be included in BnB-ADOPT$^+$, producing the new BnB-ADOPT$^+$-FDAC*-MR algorithm. For single order FDAC* enforcing, we

[1] This can be clearly seen in line 1 of function FindExistentialSupport of [1]. The expression of α involves $C_i(a)$ and $C_j(b)$, unary costs of values of x_i and x_j. While this causes no difficulties in a centralized approach, it becomes a real issue in a distributed setting.

maintain a single copy of the cost functions in which we enforce FDAC*, following the order in which agents appear in the DFS tree branches. Implementing r representations requires each agent holding a set of r cost functions $\{C_1, C_2...C_r\}$. On all r cost functions agents enforce FDAC*. The direction of the arc consistency enforcement will be defined by the set of partial orders among agents $\{O_1, O_2, ..O_r\}$.

Having different orders produces different flows of costs and as result, some values may be found not NC* in some representation. These values are deleted from all representations. Every time there is a deletion, the agent will need to reinforce FDAC* over the r representations. For this agents will need to store (i) an order for each representation, (ii) a copy of the binary and unary cost functions for each representation, (iii) a C_ϕ value for every representation (since different projections and extensions are performed on each representation, different C_ϕ values may be obtained), (iv) all children $subtreeContribution$ to C_ϕ for each representation (since different projections and extensions are performed on each representation, agents will contribute to the C_ϕ differently) [3].

The following changes in messages are needed to maintain the previous structures: (i) VALUE –a vector C_ϕ[] is sent containing the C_ϕ values for every representation– (ii) COST –a vector $subtreeContribution$[] is sent containing the subtree contribution to the C_ϕ for every representation– (iii) UCO –a vector $vectorOfExtensions$[][] is sent containing the extensions for every representation–.

5 Experimental Results

We evaluate the efficiency of BnB-ADOPT$^+$-FDAC*-MR with respect to BnB-ADOPT$^+$-FDAC* (which maintains FDAC* on a single representation) on unstructured instances with binary random DCOPs, and on structured distributed meeting scheduling. We have generated binary random DCOP instances of 10 variables, domain size 10, and network connectivity $p_1 = 0.3, 0.4, 0.5, 0.6$. Costs are selected from an uniform cost distribution. Two types of binary cost functions are used, small and large. Small cost functions extract costs from the set $\{0, \ldots, 10\}$ while large ones extract costs from the set $\{0, \ldots, 1000\}$. The proportion of large cost functions is 1/4 of the total cost functions (this is done to introduce some variability among costs). On the meeting scheduling formulation, variables represent meetings, domains represent time slot assigned for each meeting, and there are constraints between meetings that share participants. We present 4 cases obtained from the DCOP repository with different hierarchical scenarios [12] .

Table 1 shows the details of the experiments maintaining 6 representations. On random DCOPs, BnB-ADOPT$^+$-FDAC*-MR showed clear benefits on communication costs with respect to BnB-ADOPT$^+$-FDAC*. Maintaining 6 representations, the number of exchanged messages is divided by a factor of at least 2. Also, the number of cycles required to reach the solution is divided by a factor from 2 to 3. For meeting scheduling instances we also observe a decrement in the number of cycles and messages exchanged, although to a smaller extent. Assuming that processing each message type requires approximately the same time, a decrement in cycles combined with a decrement in the number of messages per cycle is an improvement indicator. Since agents need to process less information coming from their neighbors on each iteration, and

(a) Random DCOPs

p_1	#Msgs	#VALUE	#COST	#DEL	#UCO	#Cycles	#NCCC	#Deletions
	6,128	2,795	3,047	**230**	**28**	1,039	**519,112**	80
	3,068	**1,335**	**1,391**	245	69	**480**	1,778,525	**86**
	110,696	48,281	62046	**288**	**53**	17,937	**9,910,897**	78
0.4	**41,147**	**19,309**	**21,357**	311	142	**5,561**	22,063,034	**85**
	510,411	225,155	284,781	**366**	**82**	85,710	**121,453,697**	78
0.5	**198,474**	**91,506**	**106,299**	397	244	**30,659**	318,565,730	**85**
	1196935	475,416	720,975	**408**	**108**	199,971	**470,462,443**	74
0.6	**524,406**	**209,150**	**314,454**	459	314	**87,357**	1,217,511,858	**84**

(b) Distributed Meeting Scheduling

	#Msgs	#VALUE	#COST	#DEL	#UCO	#Cycles	#NCCC	#Deletions
	2,524	1,056	1,240	**200**	**5**	462	**382,676**	49
A	**2,001**	**820**	**921**	216	9	**329**	1,762,278	**53**
	5,405	2,323	2,863	**184**	**7**	1,080	**659,314**	53
B	**3,556**	**1,513**	**1,7821**	210	23	**650**	2,487,359	**61**
	1,467	697	505	**225**	**6**	125	**71,439**	80
C	**1,156**	**509**	**353**	238	21	**83**	352,795	**85**
	1,251	526	448	**234**	**8**	132	**56,447**	83
D	**1,067**	**423**	**345**	241	24	**98**	327,749	**85**

Table 1. Experimental results of BnB-ADOPT$^+$-FDAC* (first row) compared to BnB-ADOPT$^+$-FDAC*-MR (second row) maintaining 6 representations.

they perform less iterations to reach the optimum, this combined reduction is beneficial. Notice that maintaining FDAC* on multiple representations has produced only few extra DEL and UCO messages.

The number of NCCCs (non concurrent constraint checks) [6] increases since more projection and extensions are needed to maintain FDAC* on all representations. However, observe that this increment is not linear with respect to the number of representations maintained, it is smoothed by the fact that less messages are generated and more deletions are performed. So there are messages on the BnB-ADOPT$^+$-FDAC* algorithm that will not be needed to process with multiple representations, and also there are values that will not be needed to assign or to check for node consistency.

References

1. S. de Givry, F. Heras, J. Larrosa, and M. Zytnicki. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. *Proc. of IJCAI-05*, pages 84–89, 2005.
2. P. Gutierrez and P. Meseguer. BnB-ADOPT$^+$ with several soft arc consistency levels. *Proc. ECAI*, 2010.
3. P. Gutierrez and P. Meseguer. Saving messages in BnB-ADOPT. *Proc. AAAI-10*, 2010.
4. F. Heras and J. Larrosa. Intelligent variables orderings and re-orderings in dac-based solvers for wcsp. *Journal of Heuristics*, pages 4–5, 2006.
5. J. Larrosa and T. Schiex. In the quest of the best form of local consistency for weighted CSP. *Proc. of IJCAI-03*, pages 239–244, 2003.
6. A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraint processing algorithms. *AAMAS Workshop on Distributed Constraint Reasoning*, pages 86–93, 2002.
7. P. Meseguer, F. Rossi, and T. Schiex. *Handbook of Constraint Programming. Chapter 9, Soft Constraints.* Elsevier, 2006.
8. P. J. Modi, W.M. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161:149–180, 2005.
9. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. *Proc. of IJCAI-05*, pages 266–271, 2005.
10. M. Silaghi and M. Yokoo. Nogood-based asynchronous distributed optimization (ADOPT-ng). *Proc. of AAMAS-06*, pages 1389–1396, 2006.
11. W. Yeoh, A. Felner, and S. Koenig. BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. *Proc. of AAMAS-08*, pages 591–598, 2008.
12. Z. Yin. USC dcop repository. Meeting scheduling and sensor net datasets, http://teamcore.usc.edu/dcop, 2008.

Arities of Symmetry Breaking Constraints in Binary CSPs

Tim Januschowski *

Cork Constraint Computation Centre and Computer Science Department
University College Cork Ireland
janus@cs.ucc.ie

Abstract. Static symmetry breaking is a well-established technique to speed up the solving process of symmetric Constraint Satisfaction Problems (CSPs). Static symmetry breaking suffers from two inherent problems: symmetry breaking constraints come in great numbers and are of high arity. Here, we consider the problem of high arity from a theoretical point of view. We show that in special cases of binary CSPs, we can completely break symmetries with binary constraints. We show that for more general binary CSPs, this does not hold.

1 Introduction

Symmetry breaking in Constraint Programming has been the subject of intense investigation for almost two decades, see e.g. [6]. Symmetry breaking has an empirically proved potential to speed up constructive search methods. The classic and practically most used technique in symmetry breaking is the addition of symmetry breaking constraints before search [2, 13].

Two of the inherent problems with the addition of symmetry breaking constraints for complete symmetry breaking are that symmetry breaking constraints may come in great numbers and that symmetry breaking constraints may come in high arity. The known general and complete methods add one symmetry breaking constraint per symmetry and the arity of each symmetry breaking constraint is the same as the number of variables. In the worst case, we can have an exponential number of symmetries. So, for complete symmetry breaking, we have to add an exponential number of symmetry breaking constraints and each symmetry breaking constraint is of highest possible arity. Adding an exponential number of constraints to a CSP is prohibitively costly due to space consumption. High arity slows down propagation as the time complexity of the known propagation algorithm depends on the arity [4]. Various remedies have been proposed in the literature: for special symmetry groups, we can find polynomial sized sets of constraints with reasonable arity that completely break an exponential number of symmetries [7], or sometimes one can use the problem structure in combination with the symmetries to reduce the number and arity of the constraints [10, 14]. Also, various reduction rules [5, 12] are known that reduce both the number and the arity of commonly

* Tim Januschowski studies under the supervision of Dr. Marc van Dongen, Computer Science Department, University College Cork, Ireland. Tim Januschowski is supported by the Embark initiative of the Irish Research Council for Science, Engineering and Technology.

used symmetry breaking constraints. Another remedy, that we do not want to consider here, is the addition of symmetry breaking constraints only for selected symmetries, so-called partial symmetry breaking.

In this paper, we consider complete symmetry breaking for binary CSPs for the complete group of constraint symmetries. Binary CSPs have a special historic importance in Constraint Programming. More important for the cause of this paper is that binary CSPs suit the study of constraint symmetries [1] particularly well because constraint symmetries are defined in terms of the *microstructure*: the microstructure of binary CSPs is a graph as opposed to a hypergraph as is the case of CSPs with arbitrary arity. Typically, it is much easier to visualise and inspect graphs compared to hypergraphs. In Section 4, we provide special cases of binary CSPs in which we can break all symmetries with binary constraints. Unfortunately, for the general case, even in binary CSPs and with complete knowledge of the CSP, we may have to rely on at least ternary constraints to completely break the symmetries, as we show in Section 5. We provide an example of a binary CSP where complete symmetry breaking with binary constraints is not possible, thereby answering a question posed in [8]. The consequence of the latter result is a lower bound for the arities of symmetry breaking constraints: our example shows that in a general setting, binary symmetry breaking constraints do not always completely break symmetries in binary CSPs.

2 Notation and Definitions

A *constraint satisfaction problem* CSP is a triple $(V,D,Cons)$, where V is the set of variables of the CSP, every variable x has a *domain* $D(x) \in D$, and $Cons$ is the set of constraint of the CSP. Every constraint has an *arity*. The k-ary constraint c is a pair $\langle s,r\rangle$, where s is a list of k variables $x_1,\ldots,x_k$ which is called the *scope* and $r \subseteq D(x_1) \times \cdots \times D(x_k)$ is called *the relation* of c consisting of the tuples that c forbids. The arity of a CSP is the maximum arity over all constraints in the CSP. A CSP is called *binary* if all constraints are of arity at most 2. A *literal* is a (variable,value)-assignment. A *partial assignment* is a set of literals in which no variable appears twice. If a partial assignment is allowed by the constraints of the CSP we call it *consistent*. A *solution* is a consistent assignment on all variables. If a CSP has a solution, the CSP is *satisfiable*, otherwise it is *unsatisfiable*.

We associate to any binary CSP a graph called the *microstructure* [3, 9]. The microstructure has as nodes the literals of the CSP. We have an edge between every pair of literals that is allowed by the constraints of the CSP. The microstructure complement (MSC) is the complement graph of the microstructure. The *constraint symmetries* [1] of a binary CSP are the automorphisms of the microstructure of the CSP. Symmetries partition the set of solutions of a CSP into a set of equivalences classes or *orbits*. Apart from constraint symmetries, other symmetries exist as well, notably solution symmetries. However, constraint programmers work with constraint symmetries mostly [1]. Here, we only consider constraint symmetries which we shall abbreviate to symmetries. We always use the entire group of constraint symmetries and never a subgroup.

Given a CSP P, a *valid reduction* P' [8, 13] is a CSP on the same variables, subsets of the domains of P and supersets of the constraints, such that for every orbit of solutions

in P, at least one solution in P' exists. A *single-representative valid reduction (*SRVR*)* is a valid reduction such that *exactly* one solution in P' exists per orbit of solutions in P. We call a solution in P' an *orbit representative solution.*

The members of a family of constraints are called *symmetry breaking constraints*, if the addition of the family to a CSP leads to a valid reduction. Lexleader constraints (LLCs) [2] are a well-known example of symmetry breaking constraints. SRVRs allow us to study symmetry breaking constraint independent of particular constraints and the results that we obtain are valid for any type of symmetry breaking constraints. SRVRs exist for any CSP: we can find a SRVR constructively using LLCs [2, 15]. With LLCs, the choice of the orbit representative solutions depends on the order of the variables that we choose to define the LLCs as well as the orders on the domains of the variables. However, also for arbitrary choices of orbit representatives, a SRVR always exists—we could simply add an n-ary constraint excluding any non-representative solution [8]. Here, we prove that for certain binary CSPs, we do not have to rely on n-ary constraints, but that also binary constraints suffice to produce a SRVR. We go on to show that this is not in general so.

3 Related Work

This paper is based on Puget's approach to systematically introduce symmetry breaking constraints via valid reductions [13]. This approach was generalised and extended in [8]. There we also posed the question, whether in a binary CSP, we can produce a SRVR with at most binary constraints. Here, we will give a negative answer to this question.

Reduction rules have been studied in [5, 7, 12]. These reduction rules reduce the arity and number of LLCs. Grayland *et al.* [7] manage to show the minimality of certain sets of constraints in terms of the reduction rules. Here, we show that in binary CSPs symmetry breaking constraints cannot always be reduced to binary constraints. This is true independent of the concrete reduction rule and independent of the constraints used for symmetry breaking. Solely based on Puget's abstract framework for symmetry breaking constraints, we show that no symmetry breaking constraints can always be reduced to binary constraints. In particular, our results hold for LLCs.

The number of symmetry breaking constraints is naturally connected to the arity. Luks and Roy [11], prove that the number of essential LLCs is exponential, from which follows than one cannot always reduce LLCs to a fixed arity.

4 Special Cases Where Binary Constraints Suffice

In this section, we sum up some special cases in which binary constraints suffice to obtain a SRVR for binary CSPs.

We define PATH as the class of CSPs whose MSC is a path, i.e., we can think of the constraints as generalised implications. We note that any CSP in path has at most binary domains. The class of CSPs is clearly tractable. For reasons of space restrictions, we omit our proofs.

Proposition 1. *Any CSP in PATH admits a SRVR obtained by adding binary constraints.*

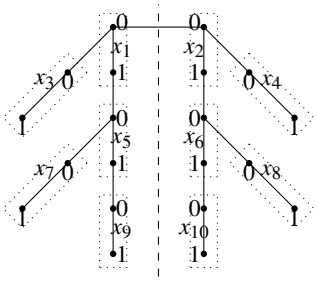


Fig. 1: The MSC of a binary CSP that does not allow a SRVR with binary constraints. The only symmetry of the CSP is a reflection about the dashed line.

Similar results hold for CSPs whose MSC consists of connectivity components that are paths and/or cycles.

We can furthermore prove that binary CSPs where the solutions are limited in a certain way admit SRVRs obtained by adding binary constraints. In the following we list some of these conditions.

Proposition 2. *Let P be a binary* CSP. *In all of the following conditions, P has a* SRVR *obtained by adding binary constraints, if P has*

1. *3 variables and arbitrary domains,*
2. *4 variables and binary domains,*
3. *no more than 3 orbits of solutions,*
4. *only one orbit of solutions of cardinality greater than 1,*
5. *solutions such that any pair of solutions from different orbits of solutions in P does not share literals.*

Case 4 of Proposition 2 subsumes the case of CSPs consisting of an all-different constraint. For this case, Puget proved that binary symmetry breaking constraints suffice [14]. Proposition 2 is unlikely to be useful in practice because it requires knowledge of the solutions. However, Proposition 2 does tell us that a binary CSP which does not have a SRVR obtained by adding binary constraints needs to have a certain number of solutions and that the solutions must interact in a non-trivial way. In the next section, we provide such a CSP.

5 Binary Constraints Do Not Always Suffice

In this section, we show that in binary CSPs and even with knowledge of the solutions of the CSP, we must rely on non-binary constraints to produce a SRVR. This provides a lower bound for the afore-mentioned reduction rules. The reasoning for this is as follows. If any reduction rule in the case of binary CSPs can always reduce the arity of the constraints to the lower bound we provide here, then the reduction rule is optimal.

We consider a binary CSP whose MSC is a tree such that we need at least ternary constraints to obtain a SRVR.

Theorem 1. *For binary* CSPs*, a* SRVR *obtained by adding binary constraints may not always exist.*

Proof (Sketch). We consider the 10 variable CSP P_{tree} whose MSC is depicted in Figure 1. Every variable has domains $\{0,1\}$, the MSC of P_{tree} is a tree. By inspection of the MSC, it is obvious that there is only one non-trivial symmetry of the MSC. The symmetry swaps literals (x_i, j) with (x_{i+1}, j) for odd i and $j \in \{0,1\}$. The symmetry is a reflection about the dotted line in Figure 1. We define a set $L := \cup_{i\in\{1,2,5,6,9,10\}}\{(x_i,1)\}$. The solutions of P_{tree} we consider in the following, have the set L in common. We consider the following four self-symmetric solutions of P_{tree}:

$$L\cup\{(x_3,1),(x_4,1),(x_7,0),(x_8,0)\}, \qquad L\cup\{(x_3,1),(x_4,1),(x_7,1),(x_8,1)\},$$
$$L\cup\{(x_3,0),(x_4,0),(x_7,1),(x_8,1)\}, \text{ and} \qquad L\cup\{(x_3,0),(x_4,0),(x_7,0),(x_8,0)\}.$$

Orbits of solutions that consist of more than one solution contain exactly two solutions because the CSP only has one non-trivial symmetry. From the orbits of solutions with two members, we consider the following eight solutions:

$$\begin{aligned}
A_1 &= L \cup \{(x_3,1),(x_4,0),(x_7,1),(x_8,1)\},\\
A_2 &= L \cup \{(x_3,0),(x_4,1),(x_7,1),(x_8,1)\},\\
B_1 &= L \cup \{(x_3,1),(x_4,1),(x_7,0),(x_8,1)\},\\
B_2 &= L \cup \{(x_3,1),(x_4,1),(x_7,1),(x_8,0)\},\\
C_1 &= L \cup \{(x_3,1),(x_4,0),(x_7,0),(x_8,1)\},\\
C_2 &= L \cup \{(x_3,0),(x_4,1),(x_7,1),(x_8,0)\},\\
D_1 &= L \cup \{(x_3,1),(x_4,0),(x_7,1),(x_8,0)\},
\end{aligned}$$

and

$$D_2 = L \cup \{(x_3,0),(x_4,1),(x_7,0),(x_8,1)\}.$$

Members of the same orbit have the same capital letter. In order to find a SRVR we need to choose an orbit representative among solutions A_1 and A_2. We note that the only pairs of literals in A_1 that is not contained in any of the four self-symmetric solutions is pair $p := \{(x_3,1),(x_4,0)\}$. If we disallow p, we remove some solutions from the other orbits and we can then easily conclude that A_2 cannot be an orbit representative solution. A symmetric argumentation holds if we choose solution A_2 as an orbit representative, which shows that P_{tree} does not have a binary realisable SRVR. □

At the time of writing, it is not clear, whether ternary constraints suffice to produce a SRVR in the CSP that proves Theorem 1. With four-ary constraints, it is straight-forward to prove that a SRVR exists in the case of the afore-mentioned CSP.

6 Conclusion and Future Work

In this paper, we considered arities of symmetry breaking constraints. Using Puget's abstract framework of valid reductions for symmetry breaking constraints, we presented

the following result: in a number of binary CSPs, binary constraints suffice to completely break all symmetries. However, we also proved that for general binary CSPs, we may have to rely on non-binary constraints.

Our results on binary symmetry breaking constraints are existence results and do not come with an algorithm. Future work could try to identify ordering heuristics that allow us to reduce the arities of LLCs.

Acknowledgements

The author would like to thank Barbara M. Smith for helpful discussions and Marc van Dongen for the same as well as proof reading.

References

1. D. Cohen, P. Jeavons, C. Jefferson, K. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints 11*, 2006.
2. J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Principles of Knowledge Representation and Reasoning (KR '96)*. Morgan Kaufmann, 1996.
3. E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings AAAI'91*, pages 227–233, 1991.
4. A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographic orderings. In *Principles and Practice of Constraint Programming (CP 02)*, 2002.
5. A. M. Frisch and W. Harvey. Constraints for breaking all row and column symmetries in a three-by-two matrix. In *In Proceedings of SymCon'03*, 2003.
6. I. P. Gent, K. E. Petrie, and J.-F. Puget. Symmetry in constraint programming. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, pages 329–376. Elsevier, 2006.
7. A. Grayland, C. Jefferson, I. Miguel, and C. Roney-Dougal. Minimal ordering constraints for some families of variable symmetries. *Annals of Mathematics and AI*, 2009.
8. T. Januschowski, B. M. Smith, and M. R. C. van Dongen. Foundations of symmetry breaking revisited. In *Proceedings of SymCon'09*, 2009.
9. P. Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *Proceedings AAAI*, pages 731–736, 1993.
10. V. Kaibel and M. E. Pfetsch. Packing and partitioning orbitopes. *Mathematical Programming*, 114, Number 1 / July, 2008:1–36, 2008.
11. E. M. Luks and A. Roy. The complexity of symmetry-breaking formulas. *Annals of Mathematics and Artificial Intelligence*, 41:2004, 2002.
12. H. Öhrmann. Breaking symmetries in matrix models. Master's thesis, Dept. Information Technology, Uppsala University, 2005.
13. J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Methodologies for Intelligent Systems*, pages 350–361, London, UK, 1993. Springer-Verlag.
14. J.-F. Puget. Breaking symmetries in all-different problems. In *International Joint Conferences on Artificial Intelligence*, pages 272–277, 2005.
15. T. Walsh. General symmetry breaking constraints. In *Principles and Practice of Constraint Programming – CP 2006*, volume 4204/2006 of *Lecture Notes in Computer Science 4204, 2006*, pages 650–664, 2006.

Conflict and Solution Driven Constraint Learning in QBF

Student: Paolo Marin
Advisors: Enrico Giunchiglia and Massimo Narizzano

DIST - Università di Genova
Viale Causa 13, 16145 Genova, Italy
name.lastname@unige.it

Abstract. In this paper we describe the Conflict and Solution Driven Constraint Learning (CSDCL) developed in *QuBE*7, a modern search-based QBF solver which incorporates the latest SAT techniques in terms of data structures, search heuristics and learning mechanisms, extending them to the QBF case. The new learning mechanism implemented in *QuBE*7 improves the effectiveness of non-chronological backtracking by producing asserting clauses and terms that lead to longer back-jumps and greater reduction in the search space to be explored. We also report the positive impact given by modifying the heuristic function, so that it assigns a variable with the polarity it had before the last backtrack, also called progress saving in RSat. In the experimental analysis we compare *QuBE* in the current and previous versions, showing that the new algorithm is far more effective in reducing the search space to explore, especially when combined with progress saving, resulting in greater efficiency and more solved problems. The comparison with other state-of-the-art solvers shows that *QuBE*7 is the fastest search based QBF solver.

1 Introduction

Propositional Satisfiability (SAT) is a success story in Computer Science: Current state-of-the-art solvers can efficiently solve problems with millions of variables and are routinely used to solve many different problems in formal verification. Essential features in nowadays' SAT solvers are (*i*) learning mechanisms for non chronological backtracking and learning; and (*ii*) search heuristics used to explore the search space, which is again based on the learning mechanism of the solver. Because of the importance of the learning mechanism, these solvers are often called "Conflict Driven Clause Learning" (CDCL) solvers.

Quantified Boolean Formulas are a powerful extension of the SAT problem, allowing the variables to be existentially as well as universally quantified. The added expressiveness allows many different problems to be efficiently encoded into QBF instances, such as in Verification [1, 2] and Planning [3, 4]. Despite the fact that the encoding of the problem is more compact, in practice solving a QBF problem in many cases is still harder than solving its SAT encoded version. One of the reasons is that current *state-of-the-art* QBF solvers implement only

few of the modern techniques developed for SAT. Indeed, (*i*) in many cases it is not trivial how to generalise SAT techniques to the QBF case; (*ii*) it is a priori not clear whether they are going to pay-off; and (*iii*) the effort required by the implementation is far more substantial: To get an idea of the different complexity, MiniSat consists of roughly 900 lines of code, and *QuBE* of almost 8000.

In this paper we describe the CSDCL procedure developed into the QBF solver *QuBE*7, which also incorporates the latest SAT techniques in terms of data structures, search heuristics and learning mechanisms, extending them to the QBF case. This paper extends [5] as we concentrate on the learning mechanism implemented in the core solver, which improves that of *QuBE*6, and whose implementation in the QBF setting is far more complicated than in the SAT case. Indeed, assuming the current assignment (defined as a set of consistent literals) is μ and that we are backtracking from a conflict, in the QBF case it might be the case that the resolution involves clauses with more than one open (i.e., not assigned by μ) literals or with a subsumed literal (i.e., with a literal which is satisfied by μ). This cannot happen in the SAT case, and avoids the complications one has to deal in the QBF case, such as the problem of dealing with tautological clauses (i.e., containing both a variable and its negation) while backtracking. Further, in the QBF case, the conflict analysis procedure has to allow for non-chronologically backtracking and learning not only when a conflict is discovered, but also when a solution is found.

In the experimental analysis we compare *QuBE* in the current and previous versions, showing that the new algorithm is far more effective in reducing the search space to explore, in particular when combined with progress saving [6], resulting in greater efficiency and more solved problems. The comparison with other state-of-the-art solvers shows that *QuBE*7 is the fastest search based QBF solver.

2 UIP Learning in *QuBE*

*QuBE*7 engine is a modern, CSDCL implementation of QDLL [7], and features many of the latest techniques developed for improving the effectiveness and the efficiency of SAT solvers, like separate data structures for binary and n-ary constraints [1] or progress saving [6, 8], but also techniques which are specific to QBF solvers, like pure literal detection and propagation and prime implicant construction for solution learning [9]. Progress saving imposes which value to assign a splitting variable by caching the assignment values the variables have before a backtrack takes place. It was extended to QBF in *depqbf* [10]. See [5] for a description of *QuBE*7 from the user perspective, and refer to [9] for an extensive introduction to QBF logic and the notation we will use in the following.

The concept of Unique Implication Point was introduced for SAT in [11], and in the form of 1-UIP is now the most widely used stop criterion in the conflict

[1] With the term *constraint* we mean either a clause or a term (or cube) without distinction.

analysis procedure of CDCL-based SAT solvers. 1-UIP schema lets the clause to learn (usually called *asserting clause*) X to be the first one obtained during the conflict analysis procedure through resolution steps having exactly one literal assigned at the maximum decision level among all the literals in X. Learning was extended to QBF in [12–14]. Differently from SAT, in QBF 3 special cases may happen during the clause (resp. term) resolution process when backtracking from a conflict (resp. solution): (*i*) an unassigned universal (resp. existential) literal, or (*ii*) a satisfying universal (resp. existential) literal, or (*iii*) a pair of universal (resp. existential) literals which are occurrences of the same variable but with opposite polarities. The latter produces a tautological clause/term, which can be either learned and used anyway if the procedures involved in the forward phase of the search, i.e. propagation, allow for that, or turned to an *asserting constraint* by resolving out the literals that do not allow the conflicting literals being removed by minimisation [9]. *QuBE* exploits the second option.

Our procedure to calculate the asserting constraint is an extension to QBF of that of Chaff [15] in which tautological constraints are not produced, nor allowed in the search. A different approach for extending Chaff's procedure to QBF where tautological clauses are allowed was proposed in Quaffle [13]. The idea is that, starting from an empty clause (resp. term), that is a disjunctive (resp. conjunctive) set of literals which are all assigned to false, only the existential (resp. universal) literals can be resolved by replacing each of them with the set of literals that implied their assignment, and the Unique Implication Point must be an existential (resp. universal) literal as well. In practice, we keep the highest decision level d of the existential (resp. universal) variables we met while backtracking, marking as unassigned the universal (resp. existential) variables and resolving the existential (resp. universal) variables till we have only one existential (resp. universal) literal l in the current "asserting" clause (resp. term) X having assignment level d: In the problematic case in which the asserting clause (resp. term) X contains also an unassigned universal (existential) literal occurring to the left of l in the prefix (this is possible only if l was assigned as a unit, because of the restrictions imposed on the branching order by the prefix dependencies), we cannot use X as asserting clause (resp. term) for assigning l as unit, and thus we have to persist in backtracking resolving out l. This allows to directly remove the conflicting literals, e.g. universal (resp. existential) literals $l', \overline{l'}$ occurring with both polarities, when they occur to the left of l in the prefix (also this case is possible only if l was assigned as a unit). Roughly speaking, we can always keep on doing resolution following the assignment stack order, meaning that we follow the graph of implications, unless we find an asserting clause (resp. term) X, having l as the only existential (resp. universal) literal assigned at the highest level d, and some conflicting literals that occur to the right of l. In this case, l cannot be assigned as unit, and thus we cannot generalise the previous case as well. Instead, we have to resolve all the existential (resp. universal) literals in X that follow l in the prefix, until all the conflicting literals can be removed by minimisation. This can be safely done, since they are guaranteed to be assigned as unit, but can introduce variables coming from

parts of the implication graph which are not strictly related to the current conflict/solution. In *QuBE*6 this kind of "out-of-order resolution" is performed more often, as it switches from implication graph order to prefix order as soon as a pair of conflicting literals is found.

With respect to *QuBE*6 (see [16]) the new algorithm can allow for longer-range back-jumps: Indeed, the back-jump level we consider is not anymore that of the node at the maximum decision level among the literals in the UIP constraint excluding l, but the maximum decision level among *(i)* the literals in the constraint that precede l in the prefix, and *(ii)* the existential (resp. universal) literals in the constraint that follow l in the prefix. Consider for example the asserting clause $X = x \vee y \vee z$ derived through Q-Resolution steps while backtracking from a conflict (the prefix is $\exists x \ldots \forall y \ldots \exists z$). Let's say that the decision levels for $\{x, y, z\}$ are, respectively, $\{10, 8, 7\}$; then x is the UIP literal. We can safely back-jump to decision level 7 and assign x as unit since at that level y does not belong to the clause because of minimisation (or universal reduction). Instead, if $\{7, 8, 10\}$ are the decision levels for $\{x, y, z\}$, z will be the unit literal at decision level 8.

3 Experimental Analysis

As environment, we use a cluster made of 4 IBM HS21 computing nodes, each with 2 Quad Core Xeon 2.5 GHz, 16 GB RAM, running Linux CentOS 5; the time limit was set to 1200 s and the memory limit to 2 GB; for each node we ran 4 processes at the time.

We first compare *QuBE*7.0 against *QuBE*6.6, and the other two best solvers according to the latest QBF Evaluation. These are *AQME-10* [17], a self-adaptive QBF solver based on QBFEVAL'06 state-of-the-art systems, and *depqbf*, a search based solver that exploits dependency schemes in the decision and backtrack strategies [10]. [2] We used the same pool of (568) fixed-structure QBF instances selected for the main track of QBF Evaluation 2010 [18]. Results are shown in Table 1, where for each solver (in Column 1) we report, respectively, the time needed to solve the testset[3], the total number of instances solved within the given time, and the number of SAT and UNSAT solved instances. Thank to its multi-engine nature, *AQME* is still the fastest solver, able to solve the highest number of problems: This is to be expected because it is well known that in the QBF case some of the problems which cannot be solved by search based solvers are easy for solvers based on variable elimination, and vice-versa. Significant is the fact that *QuBE*7.0 is a great improvement of *QuBE*6.6, being able to solve up to 53 more benchmarks. Furthermore, *QuBE*7 can solve 4 more SAT problems than *AQME*. The row *QuBE*7.0-ps refers to a slightly different

[2] A preliminary version of *QuBE*7 took part to the last QBF evaluation and solved more problems than *depqbf*. Unfortunately, the submitted version contained a bug which caused the solver to return incorrect results on 7 instances.

[3] Time is expressed in seconds; a penalty of 1200 s is given for each time-out or memory-out.

version of *QuBE*7.0 where progress saving was disabled. It is noticeable how this technique improves the performance of the solver, in particular being effective for solving satisfiable problems.

Solver	Total (s)	Total (#)	SAT (#)	UNSAT (#)
AQME-10	171,231.30	447	196	251
QuBE7.0	198,995.40	430	200	230
QuBE7.0-ps	234,435.00	408	185	223
QuBE6.6	267,608.09	377	165	212
depqbf0.1	256,801.85	369	165	204

Table 1. Performance on QBFEVAL'10 Testset.

Solver	Time (s)	#Sol.	avg(term)	#Confl.	avg(clause)
QuBE7.0	13515.72	12422	99.75	8094	74.48
QuBE7.0-ps	18657.68	18060	94.91	27454	39.95
QuBE6.6	31373.58	13823	99.77	28256	40.47

Table 2. Comparison of the search space explored by *QuBE* in 3 versions.

In Table 2 we compare the size of the search trees explored by *QuBE*7 (with and without progress saving) and *QuBE*6 on the 335 benchmarks solved by all the 3 systems. For each solver (in Column 1) we report, respectively, the cumulative solving time, the averages of the number of solutions, learned term size, number of conflicts and learned clause size. The first immediate observation is that the new search engine is more efficient, needing far less time to solve the same problems. This is due to the significant reduction of the search space explored, as witnessed by the much lower number especially of conflicts, but of solutions as well, discovered. It is noticeable that progress saving has a valuable role in that. On the other hand, we see that the size of the learned constraints is much higher, which seems somehow counter-intuitive given the fact that in *QuBE*7 binary constraints are propagated first and this in SAT reduces the size of the learned clauses [19]: This will be matter of future investigation. *QuBE*7 binary is available at `www.star-lab.it/~qube`.

4 Conclusion

In this paper we presented the new learning algorithm implemented in *QuBE*7, a search-based QBF solver. In particular, the new learning algorithm allows for more effective pruning of the search space, especially when combined with progress saving. This algorithms together with the new data structure make *QuBE*7 much faster than its previous version. Experimental results confirm that

QuBE is a *state-of-the-art* solver. In the future we want to investigate the individual effects given by the new data structures and algorithms, to try different unlearning strategies and to take advantage of dependency schemes.

References

1. Ayari, A., Basin, D.A.: Bounded model construction for monadic second-order logics. In: Proc. CAV '00. (2000)
2. Scholl, C., Becker, B.: Checking equivalence for partial implementations. In: Proc. DAC. (2001) 238–243
3. Castellini, C., Giunchiglia, E., Tacchella, A.: Improvements to SAT-based conformant planning. In: Proc. ECP. (2001)
4. Rintanen, J.: Constructing conditional plans by a theorem prover. Journal of Artificial Intelligence Research **10** (1999) 323–352
5. Giunchiglia, E., Marin, P., Narizzano, M.: QuBE7.0, System Description. In: SAT'10 workshop "Pragmatics of SAT" (POS 2010). To appear in Journal on Satisfiability, Boolean Modeling and Computation (system description category). (2010)
6. Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In: Proc. SAT. (2007)
7. Cadoli, M., Giovanardi, A., Schaerf, M.: An algorithm to evaluate quantified Boolean formulae. In: Proc. AAAI. (1998)
8. Pipatsrisawat, K., Darwiche, A.: Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA (2007)
9. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of quantified Boolean formulas. Journal of Artificial Intelligence Research (JAIR) **26** (2006) 371–416
10. Lonsing, F., Biere, A.: Integrating dependency schemes in search-based qbf solvers. In: Proc. SAT. (2010 (To appear))
11. Marques-Silva, J.P., Sakallah, K.A.: GRASP - A New Search Algorithm for Satisfiability. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design. (November 1996) 220–227
12. E. Giunchiglia and M. Narizzano and A. Tacchella: Backjumping for quantified Boolean logic satisfiability. In: Proc. IJCAI. (2001)
13. Zhang, L., Malik, S.: Conflict driven learning in a quantified Boolean satisfiability solver. In: Proc. ICCAD. (2002)
14. Letz, R.: Lemma and model caching in decision procedures for quantified Boolean formulas. In: Proc. of Tableaux, Springer (2002)
15. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: Proc. DAC. (2001)
16. Giunchiglia, E., Narizzano, M., Tacchella, A.: QuBE++: An efficient QBF solver. In: Proc. FMCAD. (2004)
17. Pulina, L., Tacchella, A.: A self-adaptive multi-engine solver for quantified Boolean formulas. Constraints **14** (2009) 80–116
18. Peschiera, C., Pulina, L., Tacchella, A., Bubeck, U., Kullmann, O., Lynce, I.: The seventh qbf solvers evaluation (qbfeval'10). In: Proc. SAT. (2010)
19. Ryan, L.: Efficient Algorithms for Clause-Learning SAT Solvers. PhD thesis, SFU University (2003)

Constraints in the Cloud

Student: Jacopo Mauro
Supervisors: Maurizo Gabbrielli and Zeynep Kiziltan

Department of Computer Science, University of Bologna, Italy.
{jmauro,gabbri,zeynep,}@cs.unibo.it

1 Introduction

Parallelising constraint solving based on tree search has become a popular research topic (see [2] for a brief overview). One approach is search-space splitting in which different parts of the search space are explored in parallel (e.g. [6]). Another approach is the use of algorithm portfolios. This technique exploits the significant variety in performance observed between different algorithms and combines them in a portfolio [3]. In constraint solving, an algorithm can be a solver or a tuning of a solver. Portfolios have often been run in an interleaving fashion (e.g. [8]). Their use in a parallel context is relatively new [5, 2].

Recent years have witnessed considerable interest in large-scale parallelism as can be observed by the ever growing number of volunteer/grid/cloud computing projects that use wast amount of processors. Considering the complexity of the constraint problems and thus the computational power needed to tackle them, it is appealing to benefit from large-scale parallelism and push for a massive number (hundreds if not thousands) of CPUs. Bordeaux et. al have started to investigate large-scale parallelism in constraint solving [2] by using the portfolio and search-space splitting strategies. They have conducted experiments on constraint problems using a parallel computer with the number of processors up to 128. They reported that the parallel portfolio approach scales very well in SAT, in the sense that utilizing more processors consistently helps solving more instances in a fixed amount of time.

As done also in [2], most of the prior work in parallel constraint solving assumes a parallel computer with multiple CPUs. This architecture is fairly reliable and has low communication overhead. However, such a computer is costly and is not always at our disposal, especially if we want to push for massive parallelism. Jaffar et al addressed this problem in [6] by using a bunch of computers in a network. They employed 61 computers in a search-space splitting approach and showed that such a method is effectively scalable.

Our aim is to combine the parallelization benefits of [2] and [6] when solving constraint satisfaction problems (CSPs). We present an architecture in which massive number of volunteer computers can run several (tunings of) constraint solvers in parallel in a portfolio approach. The goal is to maximise the probability of solving many CSPs in a fixed amount of time and/or minimise the power consumption required to solve the CSP instances. To reach our goals, we

would like to exploit certain information such as expected solving time, probability of solving an instance in a fixed amount of time, and expected resource consumption, by using machine learning techniques.

We are developing our architecture using the service-oriented computing paradigm (SoC) which is an emerging paradigm where services are autonomous computational entities that can be composed to obtain more complex services for developing massively distributed applications (see e.g [4] or the Sensoria Project at `http://www.sensoria-ist.eu/`). The architecture is designed and implemented in Jolie [7] which is the first full-fledged programming language based on SoC paradigm. The reasons behind the choice of SoC, and thus Jolie, can be summarised as follows. First, it is scalable; massive number of communications with different computers can easily be handled. Second, it is modular; new services can easily be integrated and organised in a hierarchy. This is particularly important in an architecture like ours which has several sub services. Third, it allows us to deploy the framework in a number of different ways. Jolie indeed provides interaction between heterogeneous services, like in the case of web services (e.g integrating a google map application in a hotel-search application). We can therefore easily interact with other services (even graphical ones) in the future, make our architecture be part of a more complex system or deploy the architecture on modern cloud computing networks such as Amazon EC2 or Microsoft Azure.

We report experiments up until 100 computers using a first naive prototype. As the results confirm, the architecture is effectively scalable.

2 Architecture

Fig. 1 depicts our architecture using a notation similar to UML communication diagrams. When services are used, we can have two kinds of messages: one way message denoted by the string ⟨message name⟩(⟨data sent⟩) and a request response message denoted by ⟨message name⟩(⟨data sent⟩)(⟨data received⟩).

The figure is read as follows. The user utilises the *redirecting service* to get the location of the *preprocessing service* and then sends to the preprocessing service a problem instance i_k to be solved. Once i_k is sent, the preprocessing service contacts the *CBR service* which runs a case-based reasoning system to provide the expected solving time t_k of i_k. The preprocessing server then sends t_k and i_k to the *instance distributor service*. This service is responsible for scheduling the instances for different (tunings of) solvers and assigning the related jobs to the volunteer computers. This can be done in a more intelligent way thanks to t_k provided by the CBR service. This value can be used for instance to minimize the average solving time. Finally, the *volunteer service* asks the redirecting service the location of the instance distributor service and then requests a job from it using a request response message. Note that the use of the redirecting service makes it possible to have multiple preprocessing and instance distributor services in the future. It is even possible to consider the concurrent use of different services exploiting machine learning algorithms to predict the solving times.

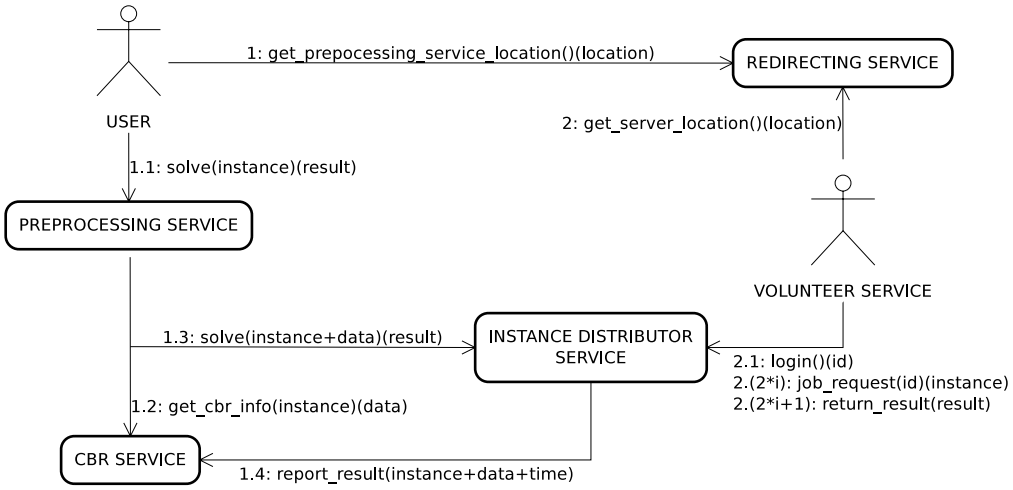


Fig. 1. Architecture.

An input instance of the architecture is specified in XCSP which is a relatively new format to represent constraint networks using XML (`http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf`). The reasons of this choice are that XCSP format has been used in the last constraint solver competitions (and therefore many solvers support it) and that such a low level representation is useful to extract the feature vectors needed by a CBR algorithm.

3 Preliminary Experimental Results

We have developed a first prototype to serve as a proof of concept. Since our initial concern is scalability, we have discarded the CBR service from the first prototype. In the experiments, up to 100 computers of our labs computers are employed for running the volunteer services and only one for the remaining services. We consider the instances of the 2009 CSP Solver Competition (`http://www.cril.univ-artois.fr/CPAI09/`), six of its participating solvers (Abscon 112v4 AC, Abscon 112v4 ESAC, Choco2.1.1 2009-06-10, Choco2.1.1b 2009-07-16, Mistral 1.545, SAT4J CSP 2.1.1) and one solver (bpsolver 2008-06-27) from the 2008 competition (`http://www.cril.univ-artois.fr/CPAI08/`). These solvers are provided as black-box, their tunings is not possible. As an experiment is affected by the current work load of the computers, we perform and report three runs. We implemented an instance distribution service that for every instance received tries to use in parallel every solver in the portfolio without interrupting a job whenever it has started. The experiments focus on the following instances:

– Easy SAT: 1607 satisfiable instances solved in less than 1 minute

– Easy UNSAT: 1048 unsatisfiable instances solved in less than 1 minute
– Hard SAT: 207 satisfiable instances solved in between 1 and 30 minutes
– Hard UNSAT: 106 unsatisfiable instances solved in between 1 and 30 minutes

Such times refer to the best solving times of the competition.

n°	Easy SAT (30 min)			Easy UNSAT (30 min)			Hard SAT (1h)			Hard UNSAT (1h)		
20	15	14	15	17	18	18	3	3	6	7	7	9
40	132	128	135	150	150	150	8	8	7	16	17	13
60	141	140	140	320	318	322	19	15	14	23	23	22
80	144	145	151	335	323	328	25	21	25	29	30	30
100	179	179	192	336	345	334	25	25	25	44	33	36

Table 1. Experimental results.

In Table 1, we present the number of instances solved in 30 minutes for the easy instances and in 1 hour for the hard instances.

The results are promising. Even without the CBR service and the different tunings of solvers, the number of the instances solved in a fixed amount of time increases as the number of computers increases, no matter how busy the laboratory computers are. Note that only one computer is used to run the preprocessing and the instance distributor services, and yet the system can handle 100 computers without any problems. Performing this experiments we have found that this first prototype does not reach good (i.e.. linear) speed ups because some solvers cannot solve even the easy instances in less then 30 minutes. Hence, many computers are spending more than 30 minutes for solving an already solved instance. These observations suggest we shall allow the interruption of a computation if the related instance is already solved.

4 Related Work

There is considerable amount of prior work on parallel constraint solving. We here discuss only those that use massive parallelism. Our work is similar to the one described in [2] in the sense that we too use the portfolio approach. However, there are a number of differences. First, we consider CSP instances and several different constraint solvers (including SAT and CP solvers), as opposed to SAT instances and one SAT solver. Second, we create portfolios by running each instance on several computers and several (tunings of) solvers at the same time. The solver-independent approach of [2] instead uses only different tunings of the same solver. Third, whilst we assume a cloud of independent computers, [2] assumes a dedicated cluster of computers.

Jaffar et al [6] as well propose an architecture based on volunteer computing. Unlike ours, this architecture uses the search-space splitting strategy and the experiments confirm scalability on ILP instances using 61 computers. There are however other substantial differences. In many environments like laboratories and home networks, computers stay behind a firewall or network address

translation (NAT) which limit their access from outside. We are able to access such computers by using only the request response messages instead of using direct messages as done in [6]. This choice brings further advantages over [6] like smaller number of messages sent, the applicability to the majority of networks, and having only the server as a possible bottleneck. The price to pay is the impossibility of using certain protocols that allow, for instance, the interruption of tasks that are searching for a solution that has been already found.

Our architecture owes a lot to CPhydra [8] , the winner of 2008 CSP Solver Competition. It combines many CP solvers in a portfolio. CPhydra determines via CBR the subset of the solvers to use in an interleaved fashion and the time to allocate for each solver, given a CSP instance. Our work can thus be seen as the parallel version of CPhydra which eliminates the need of interleaving, giving the possibility of running several (tunings of) solvers at the same time. This brings the chance of minimising the expected solving time as there is no order among the solvers. Moreover, parallelism gives the opportunity of updating the base case of CBR even in a competition environment.

5 Future Work

We have presented an architecture in which massive number of computers can run several (tunings of) constraint solvers in parallel in a portfolio approach. The architecture is implemented in SoC which is becoming the choice of paradigm for the development of scalable and massively distributed systems. The initial experimental results confirm the scalability. Our future plans are to make the architecture more efficient, useful and reliable.

As for efficiency, we are developing the CBR service to predict the run time of the solvers. We are currently adapting the CPhydra framework for the development of this service. As the preliminary tests suggest, we are facing a scheduling problem that is too big to be solved optimally in the CPhydra fashion. For this reason, we are developing different kind of scheduling algorithms using different heuristics which aim at minimising the average solving time and/or minimising the power consumption of the system. We are also interested in considering other machine learning approaches such as support vector machines and artificial neural networks. Moreover, to the best of our knowledge, we are not aware of works that have studied in detail what features of a CP instance should be considered to enhance the prediction reliability of machine learning algorithms. We are therefore also interested in studying good feature vectors that can be used to predict the solving time or the resource consumption of a CSP solver.

We are as well considering the use of randomized solvers to exploit the computational power of idle nodes of the system. When some nodes of the system are not used, we can exploit them to increase the probability of solving an instance using the randomized solvers for a short amount of time.

As for usability, we aim at tackling two limitations. First, our architecture gets XCSP instance format which is too low level for a CP user. The good news is that the architecture can easily be integrated to a high level modelling and solv-

ing platform such as Numberjack (`http://4c110.ucc.ie/numberjack`) which will soon output to XCSP. In this way, we can obtain a system in which the user states her problem easily at a high-level of abstraction, then the problem gets converted to XCSP and then our cloud solver is invoked. Second, our architecture is focused on the portfolio approach. We intend to investigate how to exploit massive number of computers in search-space splitting when solving CSPs or constraint optimization problems. In particular, we would like to adapt heuristics like the ones described in [9, 1].

Finally, should the computers go off or malfunction, we might want to replicate the jobs assigned to the computers or redirect them to other computers. This is something very important since in cloud computing nodes can fail but when this happens usually a new node is quickly given. So we will study methods to make the architecture more reliable from this perspective.

Acknowledgements

We would like to thank Emmanuel Hebrard, Eoin O'Mahony, and Barry O'Sullivan for useful discussions and their input for the implementation.

References

1. Boivin, S., Gendron, B., Pesant, G.: A load balancing procedure for parallel constraint programming, `https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2008-32.pdf`
2. Bordeaux, L., Hamadi, Y., Samulowitz, H.: Experiments with massively parallel constraint solving. In: IJCAI. pp. 443–448 (2009)
3. Gomes, C.P., Selman, B.: Algorithm portfolios. Artif. Intell. 126(1-2), 43–62 (2001)
4. Guidi, C., Montesi, F.: Reasoning about a service-oriented programming paradigm. In: YR-SOC. pp. 67–81 (2009)
5. Hamadi, Y., Jabbour, S., Sais, L.: Manysat: Solver description. SAT-Race 2008 (2008)
6. Jaffar, J., Santosa, A.E., Yap, R.H.C., Zhu, K.Q.: Scalable distributed depth-first search with greedy work stealing. In: ICTAI. pp. 98–103 (2004)
7. Montesi, F., Guidi, C., Lucchi, R., Zavattaro, G.: Jolie: a java orchestration language interpreter engine. Electr. Notes Theor. Comput. Sci. 181, 19–33 (2007)
8. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. Proceedings of the 19th Irish Conference on Artificial Intelligence (AICS'08) (2009)
9. Perron, L.: Search procedures and parallelism in constraint programming. In: CP. pp. 346–360 (1999)

Soft Constraints and Partially Ordered Preferences in a Multi Criteria Optimisation Environment

Conor O'Mahony (PhD student)
and Nic Wilson (supervisor)

Cork Constraint Computation Centre,
University College Cork, Ireland.
c.omahony@4c.ucc.ie; n.wilson@4c.ucc.ie

Abstract. This paper introduces some concepts of multi-criteria decision making and multi-criteria optimisation problems, and relates them to soft constraints formalisms, specifically when there exists some incomparability between the preference levels of the problem solutions. It also outlines some future work that will be carried out in this area to investigate the strengthening the partial orderings used to compare the preference levels of problem solutions.

Keywords: preferences, multi-criteria decision making, multi-criteria optimisation, soft constraints

1 Introduction

Combinatorial optimisation involves the consideration of a discrete set of items and finding the best combination of these items according to a particular criterion. In a *multi-criteria optimisation* problem, there is more than one criterion on which each alternative or solution to the problem must be evaluated. An alternative or solution to a problem can be expressed as an *evaluation vector*, where each term in the vector represents an evaluation of the alterative with respect to a particular criterion. One such type of valuation that can be assigned to an alternative for a particular criterion is the score or *preference level* of an agent for that alternative. The set of evaluation vectors can then be ordered in some way to derive information about the corresponding alternatives, for example, which vector corresponds to the optimal alternative.

There are various different approaches for comparing alternatives, the study of which is covered in the theory of *multi-criteria decision making*. This relates also to *soft constraints* frameworks, where the evaluation of each soft constraint corresponds to a criterion, and the value of the soft constraint when applied to an alternative corresponds to an evaluation of the alternative. However there are a number of scenarios where there is some incomparability between alternatives, which results in the set of alternatives being partially ordered.

This paper details some of the work that will be carried out to investigate situations in multi-criteria optimisation where preference levels of problem solutions are partially ordered, and ways in which to strengthen the ordering of preference levels in solving these problems in a soft constraints setting. Section 2 deals with some background material in relation to multi-criteria decision making and optimisation, and also how this relates to soft constraints formalisms and Section 3 details the proposed work in this area.

2 Preliminaries

2.1 Alternatives and Evaluation Vectors

In this section we introduce the concepts of an alternative and the corresponding evaluation vector which provides an evaluation of the alternative over one or more criteria.

An alternative is defined as a solution to a combinatorial optimisation problem, which usually involves a complete assignment to the set of variables of the problem. For an alternative a in the set A of alternatives and for some evaluation function $e := A \rightarrow S$, let $e(a)$ represent an evaluation of alternative a over a particular criterion (where S is some set of possible evaluations). Let $\mathcal{A}$ be a set of m-dimensional evaluation vectors of the alternatives over m criteria, where $e_i(a)$, represents the valuation for alternative a over criterion i, i.e., for alternative $a \in A$, let $e(a) \in \mathcal{A} = (e_1(a), e_2(a), \ldots, e_m(a))$. In the remainder of this paper, we will use a shorter notation to represent an evaluation vector, i.e., an evaluation vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m) \in \mathcal{A}$, where each $\alpha_i = e_i(a)$, for some alternative $a \in A$.

In a situation where there is more than one criterion on which alternatives are evaluated, there needs to be a way of performing a comparison between alternatives, taking into account all the criteria. It might also occur that the evaluation vectors of alternatives α and β are *incomparable*, where neither alternative is preferred over the other.

2.2 Multi-criteria Decision Making and Preferences

In this section we introduce some theory behind multi-criteria decision making, specifically how alternatives are compared with one another using their evaluation vectors. This will complement some of the later material on how alternatives and evaluation vectors are compared in a soft constraint setting.

There are a number of different approaches for comparing evaluation vectors, and by extension, alternatives, which are based on *multi-criteria aggregation procedures* (MCAP) (see [6]). The first class of MCAP approaches involve aggregating the m preference levels of each evaluation $\alpha \in \mathcal{A}$ to produce an overall evaluation, and assigning each alternative to a position on a scale. Approaches based on this technique are called approaches based on a *synthesizing criterion*, since it leads to a definition of a single criterion which synthesize the m criteria. For instance, when each evaluation of an alternative over each criterion is a

numerical value, one way of computing an overall evaluation of the evaluation vector is the sum of the individual evaluations of the alternative, e.g., where $h(\alpha)$ represents an overall evaluation of α, i.e, $h(\alpha) = h(\alpha_1, \alpha_2, \ldots, \alpha_m) = \sum_{i=1}^{m} \alpha_i$.

Another class of MCAP approaches determine how each evaluation vector $\alpha \in \mathcal{A}$ compares to each other evaluation vector $\beta \in \mathcal{A}$ from a preference point of view. This will result in a *preference relation* $\succeq$ on the set $\mathcal{A}$, which is a binary relation that is usually reflexive, i.e., $\forall \alpha \in \mathcal{A}, \alpha \succeq \alpha$, and transitive, i.e., $\forall \alpha, \beta, \gamma \in \mathcal{A}$, if $\alpha \succeq \beta$ and $\beta \succeq \gamma$, then $\alpha \succeq \gamma$. Methods which use this approach are called *outranking methods*. The preference relation allows for incomparability between evaluation vectors, where if $\alpha \not\succeq \beta$ and $\beta \not\succeq \alpha$, then α and β are incomparable.

2.3 Soft Constraints and Multiple Criteria Optimization

In this section we introduce some soft constraint formalisms, we relate them to multi-criteria optimisation and discuss how previous research has implemented some multi-criteria decision making approaches and multi-criteria optimisation with soft constraints.

A soft constraint formalism involves an ordered set, $\mathcal{R}$, of preferences degrees, with some methods for combining these degrees. For a set of variables $\mathcal{W}$, a soft constraint is a function from the set of assignments to $\mathcal{W}$, i.e., the domain of $\mathcal{W}$, to $\mathcal{R}$, associating a preference degree to each assignment to a set of variables. The preference degrees in $\mathcal{R}$ may be numerical, e.g., a real number, or they might be *ordinal*, e.g., taking a discrete set of values such as: 'unsatisfied', 'poor', 'OK', 'good', and 'very good'.

Soft constraints connect with multiple criteria decision making methods and preferences, where the evaluation of each soft constraint corresponds to a criterion, and the value of the soft constraint when applied to a solution corresponds to an evaluation of an alternative. Therefore, a solution to a multi-criteria optimisation problem gives rise to a vector of preference values, one for each soft constraint. Applying the previous notation for alternatives and evaluation vectors as described in Section 2.1 to soft constraints, an evaluation $\alpha \in \mathcal{A}$ of alternative $a \in \mathcal{A}$ is defined as $\alpha = (k_1(a), k_2(a), \ldots, k_m(a))$, where each $k_i(a)$ is the evaluation of a with respect to soft constraint k_i, i.e. k_i is a function from the set of assignments to $\mathcal{W}$, to $\mathcal{R}_i$, where $\mathcal{R}_i$ is the set of preference degrees associated with each soft constraint k_i from $i = 1, \ldots, m$.

Chapter 9 of [10] contains an introduction to various specific and general formalisms of soft constraints. Many specific soft constraint formalisms can be generalised into a framework called *semiring CSPs* [2]. A semiring $\mathcal{K}_{CSP} = (\mathcal{R}, \otimes_S, \oplus_S)$ is an algebraic structure consisting of a set $\mathcal{R}$ of preference degrees, with two binary operators, $\otimes_S$ specifying how to combine preferences, and $\oplus_S$ is used to induce an ordering on the set. In a multi-criteria context, [1] has modelled multi-criteria optimisation problems as multi-criteria semiring CSPs, where each of the m criteria k_i is modelled over a semiring CSP $\mathcal{K}_{CSP_i}$ and the problem as a whole is modelled using the product of the semirings.

Other research where multi-criteria decision making approaches that have been integrated into a soft constraints setting for performing multi-criteria optimisation include [8], [7] and [11]. [4] concentrated on algebraic representations of bi-polar optimisation problems (where there are both positive and negative evaluations), with a view to using soft constraint mechanisms to solve these problems, and [12] looked at a general logic for soft constraints for partially ordered preferences.

3 Extensions of Pareto

This section details some of the proposed and ongoing work in relation to multi-criteria optimisation and soft constraints formalisms.

As described in Section 2.3, an alternative can be represented as a vector of evaluations, for example, preference values, where each preference value in the vector is given by a soft constraint associating a preference level to that solution. When comparing alternatives via their evaluation vectors, there are some situations where it is not appropriate to use a method based on a synthesizing criterion, i.e., a method in which the valuations of the m criteria for each alternative are aggregated into one valuation which is placed on a scale that is totally ordered. For example, where the preference degrees are ordinal, it is not possible to use a summation of individual valuations for each criteria into one overall valuation. Also, it could be the case that there are a number of sources of the constraints, e.g., from different experts, and comparing these vectors using a summation or some other simple aggregation operator, is not always desirable as the different experts may have different ways of using their scales. This can be modelled as having a criterion corresponding to each expert, and a way of comparing each vector or alternative over the criteria.

A standard ordering applied to compare the evaluation vectors is based on *Pareto dominance* $\succeq_P$ (or *Pareto optimality*), where one evaluation vector is preferred to another if it is preferred in all individual criteria, i.e., for some $\alpha, \beta \in \mathcal{A}, \alpha \succeq_P \beta$ if and only if $\alpha_1 \geq \beta_1, \alpha_2 \geq \beta_2, \ldots, \alpha_m \geq \beta_m$. This results in a partial order, as some comparisons of evaluation vectors will not result in one vector dominating another according to Pareto optimality, and due to this the Pareto ordering is very weak. Other approaches such as a fuzzy comparison [10] of alternatives attribute great, sometimes excessive, importance to the worst valuation in a vector. The proposed work will include the investigation of approaches and methods that will result in ordering of preferences that are stronger than the ordering induced by Pareto dominance and will take into account situations where aggregation into one valuation is not desired.

Sorted Pareto

An example of an ordering stronger than the Pareto ordering is the *sorted Pareto* ordering, which involves sorting each vector of valuations and then applying the Pareto ordering to the sorted vector. The *sorted-permutation* of a vector

$\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathcal{A}$ is $\alpha_S = (\alpha_{(1)}, \alpha_{(2)}, \ldots, \alpha_{(m)})$ such that $\alpha_{(1)} \geq \alpha_{(2)} \geq \ldots \geq \alpha_{(m)}$. For all $\alpha, \beta \in \mathcal{A}$, vector α *sorted-Pareto dominates* vector β (written as $\alpha \succsim_S \beta$), if and only if $\alpha_S \succeq_P \beta_S$, i.e., the sorted-permutation α_S of α Pareto dominates the sorted-permutation β_S of β. Some properties of this sorted Pareto ordering are proposed as follows:

Proposition 1. *$\succsim_S$ is transitive, i.e., $\forall \alpha, \beta, \gamma \in \mathcal{A}$, if $\alpha \succsim_S \beta$ and $\beta \succsim_S \gamma$, then $\alpha \succsim_S \gamma$.*

Proposition 2. *$\succsim_S$ extends the Pareto ordering, i.e., $\forall \alpha, \beta \in \mathcal{A}$, if $\alpha \succeq_P \beta$, then $\alpha \succsim_S \beta$.*

The sorted Pareto ordering can be given a semantics in terms of utility functions. Define a compatible utility function U to be a real-valued monotonic function on the ordered set of preference degrees. For alternative α we define $U(\alpha)$ to be $\sum_{i=1}^{n} U(\alpha_i)$. It can be shown that $\alpha \succsim_S \beta$ holds if and only if for all compatible utility functions, $U(\alpha) \geq U(\beta)$. Stronger ordering relations can then be generated by placing extra constraints on the compatible utility functions.

Other orderings that will be considered and investigated include the maximin approach [3], discrimin ordering [5], and various lexicographical orderings [9].

Properties of Soft Constraint Systems

We will analyse various natural properties of the soft constraint systems. One such property, monotonicity, concerns the relationship between the preference ordering and a combination operator. For an order relation $\succeq$ on set $\mathcal{A}$, the combination operator $\otimes$ is monotonic over $\succeq$ if the combination preserves the given order, i.e., if and only if $\forall a, b, c, \in A, a \succeq b \Rightarrow a \otimes c \succeq b \otimes c$. The proposed work will include the investigation of monotonicity of different soft constraint combination operators —such as multiset union and pointwise $+, \times, max, min$ —over different partial order relations with a view to evaluating alternatives.

Optimisation

A further consideration for multi-criteria optimisation in soft constraints is the generation of optimal solutions in an efficient manner. To find an optimal solution in an optimisation problem usually involves a branch-and-bound tree search, where at each node of the tree, some form of propagation is used to generate upper bound information which is used to prune the tree during the search. Both [13] and [7] looked at a branch-and-bound search for soft constraints problems where the preferences are partially ordered. The proposed work includes further investigation on the techniques described in [13] for branch-and-bound search, specifically using a mini-buckets approach to establish suitable bounds which can then be used at search nodes in the tree to provide efficient pruning of the search space.

4 Conclusion

In this paper we introduced some concepts from multi-criteria decision making, and discussed relationships with soft constraints formalisms. We introduced some proposed and ongoing work in relation to strengthening partial orders such as the Pareto order, and developing these in a soft constraints setting.

Acknowledgements. This material is based upon works supported by the Science Foundation Ireland under Grant No. 08/PI/I1912.

References

1. Bistarelli, S., Gadducci, F., Larrosa, J., Rollon, E.: A soft approach to multi-objective optimization. In: de la Banda, M.G., Pontelli, E. (eds.) ICLP. Lecture Notes in Computer Science, vol. 5366, pp. 764–768. Springer (2008)
2. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint satisfaction and optimization. Journal of the ACM 44, 201–236 (1997)
3. Dubois, D., Fargier, H., Prade, H.: Refinements of the maximin approach to decision-making in a fuzzy environment. Fuzzy Sets and Systems 81(1), 103 – 122 (1996)
4. Fargier, H., Wilson, N.: Algebraic structures for bipolar constraint-based reasoning. In: ECSQARU '07: Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty. pp. 623–634. Springer-Verlag, Berlin, Heidelberg (2007)
5. Fargier, H., Wilson, N.: Local computation schemes with partially ordered preferences. In: ECSQARU '09: Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty. pp. 34–45. Springer-Verlag, Berlin, Heidelberg (2009)
6. Figueira, J., Greco, S., Ehrgott, M.: Multiple Criteria Decision Analysis: State of the Art Surveys. Springer Verlag, Boston, Dordrecht, London (2005)
7. Gavanelli, M.: An implementation of Pareto optimality in CLP(FD). In: Jussien, N., Laburthe, F. (eds.) CP-AI-OR - International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems. pp. 49–64 (2002)
8. Huédé, F., Grabisch, M., Labreuche, C., Savéant, P.: Integration and propagation of a multi-criteria decision making model in constraint programming. Journal of Heuristics 12(4-5), 329–346 (2006)
9. Junker, U.: Preference-based search and multi-criteria optimization. Annals of Operations Research 130, 75–115 (2004)
10. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA (2006)
11. Torrens, M., Faltings, B.: Using soft CSP's for approximating pareto-optimal solution sets. In: AAAI Workshop Proceedings Preferences in AI and CP: Symbolic Approaches. AAAI Press (2002)
12. Wilson, N.: A logic of soft constraints based on partially ordered preferences. Journal of Heuristics 12(4-5), 241–262 (2006)
13. Wilson, N., Fargier, H.: Branch-and-bound for soft constraints based on partially ordered degrees of preference. In: Proc. ECAI-08 Workshop on Inference methods based on graphical structures of knowledge (WIGSK08) (2008)

Efficient Load Balancing in Distributed Branch and Bound for Weighted CSPs

Student: Lars Otten and Advisor: Rina Dechter

Bren School of Information and Computer Sciences
University of California, Irvine
{lotten,dechter}@ics.uci.edu

Abstract. We have developed a strategy for grid parallelization of a state-of-the-art Branch and Bound algorithm for weighted CSPs and related tasks: independent worker nodes concurrently solve subproblems, managed by a Branch and Bound master process. Efficient load distribution is key to good parallel performance and we propose a scheme that uses the problem cost functions and learns from past results to predict and balance subproblem complexity. Experimental results on up to 20 nodes yield very promising results and suggest the effectiveness of the scheme. The system runs on commodity hardware, simplifying deployment on a larger scale in the future.

1 Introduction

Our work explores parallelization of combinatorial optimization tasks over graphical models (e.g., weighted CSPs, Belief Networks). Specifically, we consider one of the best exact search algorithms for solving these problems, AND/OR Branch and Bound (AOBB), which exploits independencies and unifiable subproblems and has demonstrated superior performance [1].

To parallelize AOBB we adapt the established concept of parallel tree search [2]: the search space of partial instantiations is explored centrally up to a certain depth and the remaining conditioned subproblems are solved in parallel. Our approach assumes the most general master-worker scenario with only minimal inter-node communication and can hence be deployed on a multitude of parallel setups spanning large numbers of heterogenous computers. This concept has already proven successful for likelihood computations in the Superlink-Online system [3], which performs distributed genetic linkage analysis on thousands of computers worldwide.

The novelty in our work is in focusing on optimization and in exploiting problem decomposition for additional parallelism through the AND/OR paradigm. We use the power of Branch and Bound in a central search space that manages (and prunes) the set of conditioned subproblems. The central difference compared to likelihood computation, however, lies in two key aspects: *(1)* Avoid *redundancies* due to compromised caching of unifiable subproblems across the independently solved subproblems. *(2)* Maintain *load balancing* among the grid resources, dividing the total work equally and without major idling periods. Redundancies are critical in theory, but in empirical evaluations their practical impact has proven to be relatively small [4].

Balancing the subproblem loads, on the other hand, is a far greater challenge in the context of optimization problems, because the cost functions are used in pruning the search space. Capturing this aspect is thus the current focus of our work; we propose an estimation scheme that predicts the size of subproblems based on the problem cost functions and learns from previous subproblems to predict the extent of BaB pruning. Initial experiments yield very good results on several very hard practical problem instances. And while our current empirical work is limited to 20 machines, we demonstrate some promising scaling potential.

Related work. Parallel search schemes commonly assume a shared-memory architecture or extensive inter-process communication [2, 5, 6], in contrast to our minimal communication assumptions. Early results on estimating search complexity go back to [7] and more recently [8], which predict the size of general backtrack trees through random probing. Similar schemes were devised for Branch and Bound algorithms [9], where BaB is ran for a limited time and the partially explored tree is extrapolated. Our method on the other hand, is not sampling-based but only uses parameters available a priori and information learned from past subproblems.

2 Background

We assume the usual definitions of a graphical model as a set of cost functions over discrete variables; in a weighted constraint problem, for instance, we aim to find an assignment of minimal overall cost. The concept of *AND/OR search spaces* provides a unifying framework for algorithmic schemes to capture the underlying problem structure. In particular, AND nodes are introduced into the search space to capture conditional independencies between variables. Furthermore, caching is applied to unifiable subproblems, thereby avoiding redundant computations at the expense of using additional memory. The complexity of AND/OR graph search has been shown to be time and space exponential in the problem's induced width [10]. We can also extend the established Branch and Bound algorithm to the AND/OR paradigm; the resulting *AND/OR Branch and Bound* (AOBB) enjoys the same improved complexities [1].

2.1 Parallel Setup

In our parallelization approach we assume a very general parallel framework in which autonomous hosts are loosely connected over some network – in our case we use ten dual-core desktop computers, with CPU speeds between 2.33 and 3.0 GHz, on a local Ethernet, thus allowing experiments with up to 20 parallel nodes. We impose a *master-worker* hierarchy on the computers in the network, where a designated *master* node runs a central process to coordinate the *workers*, which cannot communicate with each other. This general model is chosen to accommodate a wide range of parallel resources, where direct node communication is often either prohibitively slow or entirely impossible; it also facilitates flexible deployment on geographically dispersed, heterogenous resources in the future, similar to Superlink-Online mentioned above [3].

2.2 Outline of Parallel Scheme

The master process implements the basic exploration and propagation of a Bround and Bound procedure. Before expanding a node n, however, the complexity of the subproblem below n is estimated. If this estimate is below a certain threshold T, the central

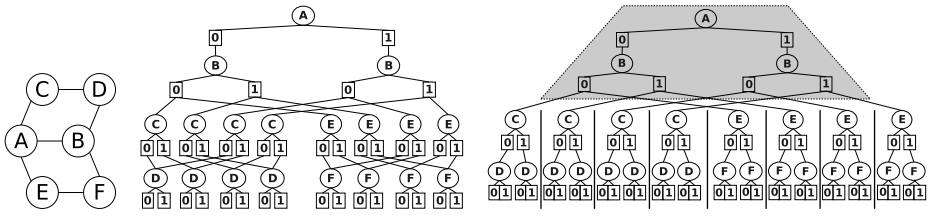


Fig. 1: (a) Example primal graph with 6 variables, (b) a corresponding AND/OR search graph, (c) the parallel search space with parallelization frontier at constant depth 2.

search is "cut off" and the subproblem is sent to a worker node for solving; otherwise n is expanded within the master and its children will be considered recursively. Worker nodes solve subproblems using sequential AOBB and send the solution back to the master, where it is processed: upon receipt of a solved subproblem, its solution is assigned to the respective node in the master search space and recursively propagated upwards as in sequential AOBB. With a fixed number of workers p, the master initially generates only the first p subproblems; each time a worker finishes the central exploration is resumed to generate the next subproblem. For more details and pseudo code see [4].

To illustrate this principle, consider Figure 1: (b) shows an AND/OR search graph for the primal graph in (a) – note the decomposition below nodes C and E, as well as the caching for D and F. (c) depicts the parallel search space when the cutoff is at constant depth 2, with the master search space in grey and 8 independent subproblems (cf. [4] for an in-depth discussion of the introduced redundancies).

In practice we need an efficient scheme to make the cutoff decisions and set the *parallelization frontier*; the master should dynamically decide at which point a given subproblem is "simple enough" for parallelization (to avoid excessively hard tasks) and also avoid very simple subproblems, where solution time will be outweighed by the distributed system overhead. We present our current solution next.

3 Subproblem Estimation

In the following we outline a scheme for estimating the size of the explored search space of a conditioned subproblem, using parameters associated with the problem's cost functions. Namely, given a subproblem P_n rooted at node n, we aim to estimate its complexity $N(n)$ as a function of the lower bound $L(n)$ on the optimal subproblem solution (obtained from a suitable heuristic function) as well as the upper bound $U(n)$, which can be derived from earlier parts of the search space.

If the search space below n was a balanced tree of height D with branching factor b, the total number of nodes is clearly $N = (b^{D+1} - 1)/(b - 1) \approx b^D$. But even in that case AOBB will only explore parts of this due to pruning. Yet it gives rise to the notion of the *effective branching factor* $b(n)$ (similar to [11]) and *average leaf node depth* $D(n)$: in post-solution analysis, $N(n)$ is known and $D(n)$ can simply be extracted from the solved subproblem search space. We can can then define $b(n)$ as the solution to $N(n) = b(n)^{D(n)}$. Consequentally, we could estimate $N(n)$ ahead of time if we knew

$b(n)$ and $D(n)$. Our prediction scheme will therefore first calculate estimates for $b(n)$ and $D(n)$, based on subproblem parameters and by learning from past subproblems, and use those to predict $N(n)$.

Estimating $b(n)$. Since all subproblems are conditioned within the same graphical models, we assume an underlying, "true" effective branching factor b. This suggests modeling $b(n)$ as a random variable; imposing a normal distribution we take its mean as the constant b. An obvious way to learn b is then to average over the effective branching factors of previous subproblems P_{n_i}, where $b(n_i)$ can be computed in post-solution analysis as before.

Estimating $D(n)$. We will not estimate the average leaf depth directly, but instead reason about its dependence on the subproblem bounds. Recall that a subproblem rooted at node n has a (heuristic) lower bound $L(n)$ and an upper bound $U(n)$ derived from the best solution so far. We know $L(n) < U(n)$, since otherwise n would be pruned. We denote with $lb(n')$ and $ub(n')$ the lower and upper bounds of nodes n' within the subproblem at the time of their expansion and similarly assert $lb(n') < ub(n')$.

Consider now a single path $\pi_k = (n'_0, \ldots, n'_k)$ within the subproblem, from $n'_0 = n$ down to leaf node $n'_k = l_k$ at depth $d_n(l_k) = k$ (relative to n). Writing $lb_i := lb(n'_i)$ and $ub_i := ub(n'_i)$ we can state $lb_i \leq lb_{i-1}$ (assuming a monotonic heuristic) and $ub_i \leq ub_{i-1}$ (because the lower bound can only improve) for all i. A node n'_i is pruned iff $lb_i \geq ub_i$ or equivalently $lb_i - ub_i \geq 0$. Hence we consider the non-increasing sequence $(ub_i - lb_i)$ along π_k, in particular its average change:

$$inc(\pi_k) := \frac{1}{d_n(l_k)} \sum_{i=1}^{d_n(l_k)} ((ub_i - lb_i) - (ub_{i-1} - lb_{i-1})) \quad (1)$$

Since l_k is a leaf we take $ub(l_k) - lb(l_k) = 0$ and the sum reduces to $U(n) - L(n)$. Rewriting Equation 1 for $d_n(l_k)$ and averaging over all subproblem paths $\pi_k, 1 \leq k \leq j$ yields an expression for the average leaf node depth as $D(n) = (U(n) - L(n)) \cdot \frac{1}{j} \sum_{k=1}^{j} \frac{1}{inc(\pi_k)}$. Define $inc(n)^{-1} := \frac{1}{j} \sum_{k=1}^{j} \frac{1}{inc(\pi_k)}$ to obtain $D(n) = (U(n) - L(n)) \cdot inc(n)^{-1}$. Now $inc(n)$, the *subproblem average increment*, can be computed directly in post-solution analysis when $D(n)$ is known.

As before we assume an underlying, "true" increment inc coinciding with the mean of the random variable $inc(n)$. For a new subproblem n we thus average over $inc(n_i)$ of previously solved subproblems P_{n_i} and estimate $D(n)$ as $(U(n) - L(n))$ divided by this average. Together with an estimate for $br(n)$, equally derived from earlier subproblems as outlined above, we can then predict $N(n)$.

Initialization. To find initial estimates for br and inc we run 15 sec of sequential search and extract the values $br(n_0)$ and $inc(n_0)$ from the largest solved subproblem n_0. To obtain an initial upper bound we perform one iteration of stochastic local search [12].

4 Empirical Evaluation

Table 1 presents results of experiments on pedigree networks and mastermind game instances,[1] using the estimation scheme with a threshold of $T = 12 \cdot 10^8$ nodes (≈ 20

[1] Available as part of the the UAI'08 evaluation, http://graphmod.ics.uci.edu/uai08/

Table 1: Results of the automated parallel scheme (ped: $p=15$, mm: $p=10$ workers).

instance	S	S/sls	P/fix^*	P/aut	instance	S	S/sls	P/fix^*	P/aut
ped7	19,114	19,309	3,352	2,783	ped31	77,580	37,844	15,230	3,910
ped13	2,752	2,796	379	359	ped41	14,643	13,999	2,173	2,251
ped19	*time*	*time*	27,372	10,611	ped51	*time*	*time*	65,818	59,915
mm_03_08_05-0011	9,715	2,943	1,443	1,085	mm_10_08_03-0000	26,102	9,876	3,866	7,604
mm_03_08_05-0012	7,568	2,030	1,430	1,584	mm_10_08_03-0011	84,920	39,761	10,044	6,846
mm_04_08_04-0000	10,620	7,807	1,306	3,076	mm_10_08_03-0012	5,630	2,489	1,357	754
mm_06_08_03-0000	12,595	259	1,797	228	mm_10_08_03-0013	10,385	5,337	2,413	2,128

minutes of processing time), a good compromise between subproblem granularity and parallelization overhead. S is the time of sequential AOBB and S/sls the time of AOBB with SLS initialization; P/fix^* is the best parallel result of various fixed-depth cutoff experiments from [4]. Finally, P/aut is the parallel time using the estimation scheme.

For pedigrees the automated scheme does at least as good, if not better, than the best fixed cutoff (it is important to realize that the latter is the best result over trying several fixed depths, while P/aut requires no such "trial and error"). We also see that the SLS initialization can have a big impact on the sequential solutions times already, which the automated scheme will equally benefit from. But even for problems where $S \approx S/sls$ (ped7, e.g.), the automated scheme improves upon the best fixed-depth scheme. In case of mastermind problems, the SLS preprocessing generally has a larger impact. But again, in many cases the automated scheme performs at least as well as the best fixed cutoff. When it does worse (mm_04_08_04 for instance) our analysis showed that the initial parameters for the subproblem prediction were too far off – we are confident an improved initialization would alleviate this.

Load Balancing. Figure 2 gives detailed subproblem statistics for the first 75 subproblem generated for ped51 (other instances exhibit similar characteristics and had to be omitted here for space reasons). Plotted is the predicted and actual number of nodes as well as the (constant) threshold T. The cutoff depth of each subproblem's root is depicted against a separate scale to the right. We see that the prediction does not give perfect estimates (which was expected), but it reliably captures the trend. Consequently the actual subproblem complexities are all within roughly one order of magnitude of each other – significantly more balanced than the fixed-depth results [4].

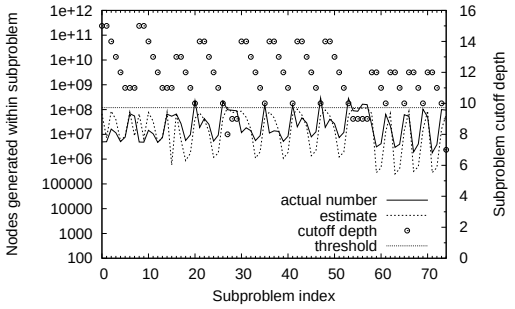


Fig. 2: Subproblem statistics for ped51.

Scaling. At this time we only have limited resources at our disposal and experimentation is limited to at most 20 worker nodes, yet we wanted to get a feeling of how the scheme scales with the number of workers. Figure 3 plots the relative speedup in relation to $p = 5$ workers. We can confirm that mostly the results are as expected, often improving linearly with p. It is evident that relatively complex problem instances (ped51 in particular) profit more from increasing p; for simpler instances we think the threshold T is too close to the overall problem complexity and inhibits better scaling.

5 Conclusion & Future Work

We presented a new scheme for parallelization of AND/OR Branch and Bound, a state-of-the-art optimization algorithms for graphical models. Based on a very general framework, our approach is viable on many kinds of parallel resources, in particular a loosely coupled group of commodity hardware.

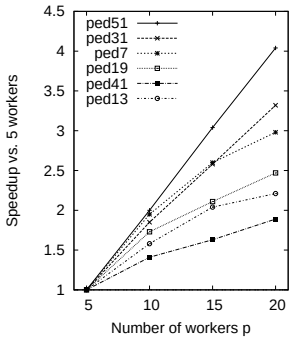


Fig. 3: Performance relative to $p = 5$ workers.

Our experiments have shown that effective load balancing is crucial for good parallel performance. We thus derived a scheme that learns from previously solved subproblems to predict subproblem complexity using an exponential expression of several parameters, including the problem cost functions. We demonstrated empirically the effectiveness of the resulting parallelization strategy, leading to consistent workload balancing and improved solutions times on very hard problems.

This is clearly work in progress and the initial scheme, while effective, still includes some ad hoc aspects. We aim to advance it by taking into account additional parameters and by providing firm theoretical grounds. Future work will also more thoroughly investigate the issue of scaling, using more resources. Finally, we need to conduct additional experiments on hard problems from various domains.

References

1. Marinescu, R., Dechter, R.: AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. Artif. Intell. **173**(16-17) (2009) 1457–1491
2. Grama, A., Kumar, V.: State of the art in parallel search techniques for discrete optimization problems. IEEE Trans. Knowl. Data Eng. **11**(1) (1999) 28–35
3. Silberstein, M., Tzemach, A., Dovgolevsky, N., Fishelson, M., Schuster, A., Geiger, D.: Online system for faster multipoint linkage analysis via parallel execution on thousands of personal computers. American Journal of Human Genetics **78**(6) (2006) 922–935
4. Otten, L., Dechter, R.: Towards parallel search for optimization in graphical models. In: ISAIM. (2010)
5. Anstreicher, K., Brixius, N., Goux, J.P., Linderoth, J.: Solving large quadratic assignment problems on computational grids. Mathematical Programming **91**(3) (2002) 563–588
6. Chu, G., Schulte, C., Stuckey, P.J.: Confidence-based work stealing in parallel constraint programming. In Gent, I., ed.: CP. Volume 5732 of Lecture Notes in Computer Science., Lisbon, Portugal, Springer-Verlag (September 2009) 226–241
7. Knuth, D.E.: Estimating the efficiency of backtrack programs. Mathematics of Computation **29**(129) (1975) 121–136
8. Kilby, P., Slaney, J., Thiébaux, S., Walsh, T.: Estimating search tree size. In: AAAI, AAAI Press (2006) 1014–1019
9. Cornuéjols, G., Karamanov, M., Li, Y.: Early estimates of the size of branch-and-bound trees. INFORMS Journal on Computing **18**(1) (2006) 86–96
10. Dechter, R., Mateescu, R.: AND/OR search spaces for graphical models. Artif. Intell. **171**(2-3) (2007) 73–106
11. Nilsson, N.J.: Artificial Intelligence: A New Synthesis. Morgan Kaufmann (1998)
12. Hutter, F., Hoos, H.H., Stützle, T.: Efficient stochastic local search for MPE solving. In: IJCAI. (2005) 169–174

Using Abstract Domains in Constraint Programming

Marie Pelleau† (student), Frédéric Benhamou†, Pascal Van Hentenryck‡, and Charlotte Truchet†

† Université de Nantes
2 rue de la Houssinère
44322 Nantes cedex 3
France
Firstname.Name@univ-nantes.fr

‡ Brown University
115 Waterman Street
Providence, RI 02912
USA
pvh@cs.brown.edu

Abstract. In semantics, Abstract Interpretation is a domain where a lot of research has been done on abstract domains, which allow the representation and manipulation in a common framework of variables of different nature (floating points, integers in particular). We investigate the possible use of abstract domains in Constraint Programming (CP), where we believe it could help and improve constraint techniques on two points: firstly, abstract domains are naturally able to mix variables of different nature (discrete, continuous), while in CP the techniques used on discrete and continuous problems differ significantly. Secondly, abstract domains can be of any shape, while CP domains are restricted to Cartesian products. We test the abstract domains for constraints by defining a consistency on octagons for continuous CSPs.

1 Introduction

The goal of this PhD is to design techniques in Constraint Programming (CP) to solve problems with discrete and continuous variables. In CP, methods already exist to solve problems over discrete or continuous variables. In particular, most of the techniques proposed to solve problems with real variables are based on interval arithmetic. The intervals' lack of precision may generate overestimations of the solutions (e.g. when there are several occurrences of the same variable, when the constraints are not linear, ...). Several methods have been proposed to reduce this overestimation using estimators for nonlinear constraints [4], using safe relaxation of quadratic constraints [9], or more recently using the monotonicity of the constraints [1].

Instead of trying to improve the interval-based methods we decided to investigate the possible use in CP of *abstract domains* of Abstract Interpretation (AI). This short paper presents some ideas to integrate the abstract domains in CP. The next section shortly introduces AI and useful notions. Section 3 presents some of the ideas to integrate abstract domains in CP. Section 4 presents the tests and the results obtained so far.

2 Abstract Interpretation

It is a well known fact that the correctness of programs cannot be generically proven. To automatically analyze a code and prevent the bugs at compiling time, AI approximates the program's possible traces. If the intersection between the approximated semantic and some dangerous zones is empty, we know that the program has no bugs. Of course, false positives can occur, the process can raise alarms while there are no bugs. We present here very briefly and intuitively some basic notions of AI. For a more complete presentation, see [5].

Useful notions for CP. Consider for example a program, in an imperative language, checked for division-by-zero and overflow errors. Clearly, these two errors occur when some variables take their values in particular zones. The goal is thus to define an over-approximation of the program's semantics, and check that this over-approximation is out of the dangerous zones.

The main principle is to focus on the values taken by the variables. These values change along the program, and form a trace that is called *concrete semantic*. They are replaced by an over-approximation of the possible values, called *abstract domains*. The theoretical tool behind this is the notion of *lattice*: a relation $\sqsubseteq$ on a non-empty set E is a *partial order* (po) iff it is reflexive, antisymmetric and transitive. A po $(E, \sqsubseteq)$ is a *lattice* iff for $a, b \in E$, the pair $\{a, b\}$ has both a least upper bound and a greatest lower bound. It is *complete* iff any set has a least upper bound.

A complete lattice has a least element and a greatest element. Abstract domains are imposed to be lattices or complete partial orders (*cpo*), so that the iterative chains defined by the loop construction converge. Another important feature of AI is that abstract domains are linked by Galois connections: given two abstract domains D_1 and D_2, a Galois connection is defined by two morphisms $\alpha : D_1 \to D_2$ and $\gamma : D_2 \to D_1$ such that α and γ are monotonic, and composing α and γ does not lose solutions. Formal definitions are given for instance in Miné's PhD thesis [10].

The links with CP are obvious: in CP, Cartesian products of the domains form a complete (even finite) poset and the consistent domains can be seen as the least fixpoint of this cpo, restricted to the elements containing the solutions. This fact has already been identified by [2] for instance. Yet, there is an important difference: CP computes *the* least fixpoint while AI computes, most of the time, *a* fixpoint in the abstract domains, and this is done in one shot by applying ad hoc operators. Hence, the fixpoint is overestimated. It is worth mentioning that in particular cases, the loop process can be iterated, performing so-called local iterations [8]. This is closer to the consistency computation in CP.

Numerical abstract domains. There exist many numerical abstract domains which differ in expressiveness and computational complexity. They can be classified in three groups: the non-relational ones (e.g. Intervals), the relational ones (e.g. Polyhedra, Ellipsoids) and the "in between" called the weakly relational ones (e.g. Zones). All of them (except Ellipsoids) come with Galois connections.

In practice, AI researchers show that non-relational domains (as the Cartesian products used in CP) are very fast but can generate overestimation due to their lack of precision. In contrast, the relational ones are very precise but are very costly in term of computational time. In between there are the weakly relational ones which give a compromise in terms of precision and computational time. For a state of the art on numerical abstract domains, with all the comparisons, see Antoine Miné's PhD thesis [10].

3 Abstract domains for CP

We explain here possible ways of integrating abstract domains in the CP framework. As said before, although the applications differ significantly, the theoretical tools are very similar. In particular, we are interested in two important properties of the AI tools: they naturally mix different abstract domains, and any kind of abstract domain can be defined (provided they satisfy the theory). In particular, abstract domains are not restricted to Cartesian products of intervals or finite sets, as it is the case in CP.

Abstract domains and consistency. The link between consistency and fixpoints on cpo is not a new idea in CP. For instance, Benhamou [2] gives a generic definition of consistency as the least fixpoint of a particular set (any set of subsets of the domains including the solutions, provided it is closed by intersection), and proposes a unified framework for heterogeneous constraint solving.

Consider a CSP on domains $D_1 ... D_n$ (integers or reals). Consider E a set of subsets of $D_1 ... D_n$, closed by intersection (always true for the usual CP domains). The elements of E that include the solutions form a cpo for inclusion, and consistency for a constraint C can be defined as the least fixpoint of this cpo. Existence and unicity are easily checked thanks to the cpo structure of $(E, \subset)$. It is worth noticing that this definition is not restricted to Cartesian products.

In fact, some of the consistencies defined on continuous and discrete domains have the same definition when abstracting the underlying cpo. We give the example of discrete *Bound consistency* and continuous *Hull consistency*.

Let $\bigtimes$ be the Cartesian product, $\mathbb{F}$ be the set of floating point numbers according to the norm IEEE 754 [7], $\mathcal{B}_{\mathbb{N}} = \bigtimes_{a,b \in \mathbb{N}} [\![a, b]\!]$ be the set of Cartesian products of integer intervals with integer bounds, and $\mathcal{B}_{\mathbb{R}} = \bigtimes_{a,b \in \mathbb{F}} [a, b]$ be the set of Cartesian products of real intervals with floating point bounds. For simplicity, in the following we will call any element of $\mathcal{B}_{\mathbb{N}}$ or $\mathcal{B}_{\mathbb{R}}$ a *box*. Consider a constraint C that can hold either on discrete or continuous variables, and define $\mathcal{B}_{\mathbb{N}|C}$ as the elements of $\mathcal{B}_{\mathbb{N}}$ that include the solutions of C (resp. $\mathcal{B}_{\mathbb{R}|C}$, $\mathcal{B}_{\mathbb{R}}$).

$\mathcal{B}_{\mathbb{N}|C}$ and $\mathcal{B}_{\mathbb{R}|C}$ being both cpos, they have least elements for inclusion. This box is the Cartesian product of the consistent domains, with respect to Hull consistency for $\mathcal{B}_{\mathbb{R}}$ and Bound consistency for $\mathcal{B}_{\mathbb{N}}$. Between Hull and Bound, only the underlying intervals are changed (on $\mathbb{N}$ or $\mathbb{F}$).

We are currently investigating the idea of using this unified framework to solve problems with discrete and continuous variables. We could for instance de-

fine and compute different consistencies depending on the variable type, and mix them in the same way as in AI with Galois Connections. We saw previously that AI uses different sort of numerical abstract domains for which they have transfer functions, an idea will be to change the representation each time a certain condition is satisfied. For example when a fixpoint is reached during the contraction, changing for another representation may improve the approximation.

In fact, we can even explore new consistencies, for instance on more relational domains. We chose to first study a new consistency for continuous CSPs (also suitable for discrete CSPs, although probably very inefficient), with octagons as the underlying domains. This is a test-case for a deeper exploration of the links between AI and CP.

Improving the precision: weakly relational domains. Instead of trying to improve the interval-based methods we decided to change for a more relational representation like in AI. To begin with, we opted for a weakly relational one, the octagon, and adapted it to continuous CSPs solving.

The idea is to use a more precise representation, hoping that there will be a gain in time as the variable domains are better reduced. In other words, more contraction and less splitting. Compared to other Cartesian abstract domains, the intuition is that octagons are more precise than boxes (Hull consistency) and less precise, but less costly, than unions of boxes (Box consistency).

This idea raises a lot of practical questions: what does it mean to contract an abstract domain like an octagon? Can consistency techniques be used? How to define the splitting part? Will it really reduce the computational time?

To try to answer these questions, we are first going to shortly introduce what an octagon is. Then we will give an idea on how using to use in CP.

The Octagon abstract domain has been introduced by Antoine Miné in his PhD thesis [10]. It is represented by conjunctions of constraints of the form $\pm X \pm Y \leq c$ where X and Y are variables and c is a constant in $\mathbb{Z}, \mathbb{Q}$ or $\mathbb{R}$.

Let $V = \{v_1, \ldots, v_n\}$ be the set of variables taking their value in the set $\mathbb{I}$ that can be $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. We call *octagonal constraint* any constraint of the form $\pm v_i \pm v_j \leq c$ with $c \in \mathbb{I}$. An octagon is the conjunction of octagonal constraints that can be stored in a matrix for instance. Note that in case there are redundant octagonal constraints, we only keep the restrictive one by comparing the c coefficients. Intersecting octagons induces a neglectable computational cost.

Adapting Octagons to continuous CP is quite straightforward at the theoretical level. We just need to replace octagons defined on $\mathbb{R}$ (real elements, real bounds) by octagons defined on the floating-point intervals (real elements, floating-point bounds), which does not change the fundamental properties.

In practice, we need to define a consistency on octagons. In two dimensions, an octagon can be seen as the intersection of two boxes, one in the original basis and the other one in the basis obtained by rotating the first one by $\frac{\pi}{4}$ rad. Note that the idea of rotating the basis has already be introduced by Alexandre

Goldsztejn et al. in [6] for other purposes. This rotation is cheap in terms of time, we simply have to replace x by $\cos(\frac{\pi}{4})x' - \sin(\frac{\pi}{4})y'$ and y by $\cos(\frac{\pi}{4})y' + \sin(\frac{\pi}{4})x'$, where (x', y') are the coordinates in the new basis.

Suppose that we want to contract an octagon w.r.t. Hull consistency, say, by applying HC4 (see [3] for details). We now have two sets of constraints $\mathcal{C} = \{C_1 ... C_p\}$ in the canonical basis and $\mathcal{C}' = \{C'_1 ... C'_p\}$ in the new basis. Each constraint has its own hull-consistent box, that can be computed using HC4, as shown on figure 1. By the cpo structure, the consistent octagon is unique and can be computed by applying intersection and HC4 on these boxes until the fixpoint is reached (no matter the order). This allows us to easily adapt the HC4 algorithm to octagons. Notice that the intersection can be done in linear time, here the most expensive part in terms of complexity, is the HC4 part. In dimension 2, we use it twice as much as for the Hull consistency on boxes.

However, for a problem in dimension n we can either apply the same strategy and apply n rotations, one for each axis and keep the octagonal structure; or apply only fewer rotations. In the second case the rotation can be guided by the gradients to choose on which axis to rotate. The rotations still are of $\frac{\pi}{4}$. The output will no longer be an octagon but an octahedron.

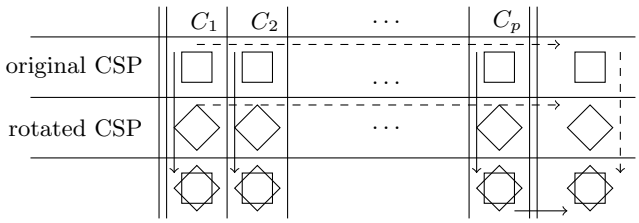


Fig. 1. Scheme of the possible heuristics for octagonal propagation.

We can do the same to split an octagon, we can see it like two independent problems and split either one or the other. As we do not want to do twice the work, the reduction made on one CSP is reflected on the other by adding the corresponding octagonal constraints.

4 Tests

We have started by implementing an HC4 version for octagons with Ibex[1] a library for interval-based solver/paver. We tested this first implementation on three simple CSPs in dimension 2.

The figure 2 shows the results obtained. On the first two examples (2(a), 2(b)), we can see that the octagonal version of HC4 improves the result. For the third example, the two versions found that the problem is unfeasible. The

[1] http://www.ibex-lib.org/

octagonal version just needs one HC4 on the rotated CSP to prove it, while the normal version needs a little more iterations. Even if the octagonal version of HC4 has been tested on few simple examples it seems promising.

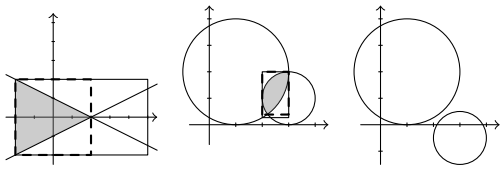


Fig. 2. Simple examples of octagon consistency. In solid line, the box obtained with HC4, and in dashed line, the box obtained with the octagonal version of HC4.

5 Conclusion

This paper has presented two new ideas to solve problems with continuous variables using domains from Abstract Interpretation. We made the hypothesis that different domains can be mixed depending on the variable's type using a general framework described previously by Benhamou. We also introduced a new propagation technique which is an octagonal version of HC4. This work is still in progress but the few tests made show encouraging results.

References

1. Ignacio Araya, Bertrand Neveu, and Gilles Trombettoni. An interval constraint propagation algorithm exploiting monotonicity. In *Proceedings of IntCP'09*.
2. Frédéric Benhamou. Heterogeneous constraint solvings. In *Proceedings of ALP'76*.
3. Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revisiting hull and box consistency. In *Proceedings of ICLP'99*.
4. Glencora Borradaile and Pascal Van Hentenryck. Safe and tight linear estimators for global optimization. *Mathematical Programming*, 102:495 – 517, 2005.
5. Patrick Cousot and Radia Cousot. Static determination of dynamic properties of programs. In *Proceedings of ISOP'76*, pages 106–130, 1976.
6. Alexandre Goldsztejn and Laurent Granvilliers. A new framework for sharp and efficient resolution of NCSP with manifolds of solutions. In *Proceedings of CP'08*.
7. David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, 1991.
8. Philippe Granger. Improving the results of static analyses of programs by local decreasing iterations. In *Proceedings of FSTTCS'92*.
9. Yahia Lebbah, Claude Michel, Michel Rueher, David Daney, and Jean pierre Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2004.
10. Antoine Miné. *Weakly Relational Numerical Abstract Domains*. PhD thesis, École Normale Supérieure, December 2004.

Overview of thesis: *Transformations of representation in constraint satisfaction*

András Z. Salamon[1,2]

[1] Computing Laboratory, University of Oxford
[2] Oxford-Man Institute of Quantitative Finance

I am currently completing my DPhil thesis at the University of Oxford. This is a brief overview of the document.

1 Complexity of constraint satisfaction

Constraint programming shifts the challenge of solving problems away from constructing a special-purpose computer program, to clearly expressing the constraints that solutions should satisfy.

Constraint satisfaction is the mathematical study of the combinatorial decision problems that underlie constraint programming. The task in combinatorial decision problems is to determine whether or not there is a solution that satisfies all the requirements. This is called ***deciding*** the instance.

The complexity of constraint satisfaction is the study of *infinite* classes of *constraint satisfaction problem instances*. These infinite classes are called constraint satisfaction *problems* or *CSPs*. Studying such infinite classes of CSP instances provides insight into why individual instances may be easy or difficult to solve.

In general, constraint satisfaction problems are believed to be difficult to decide. Many of them are NP-complete. An important question in the complexity of constraint satisfaction is:

for which CSPs is there a method to decide every instance efficiently?

*Example 1 (*Planar Graph k-Colouring*).* Any map can be coloured with four colours so that no two adjacent regions have the same colour. This colouring can be found efficiently. In contrast, we know of no efficient method to tell whether three colours are enough for a given map, and which works for every kind of map.

When the constraints are provided by listing allowed combinations of values, constraint satisfaction is NP-complete. Exhaustive search is the only general algorithm known to decide such problems. For more compact representations of constraints, such as by means of propagators, the decision problem can be even more difficult.

On the other hand, some CSPs are straightforward to decide. Sometimes all instances have properties that can be used to construct an efficient decision procedure. The following two example CSPs are straightforward.

*Example 2 (*Directed Graph Acyclicity*).* An instance is a system of strict inequalities with n variables. Each inequality involves two variables, so is of the form $X < Y$. Is it possible to assign positive integers $1, 2, \ldots, n$ to the n variables so that all inequalities are satisfied? This problem has no solution when there is some cycle of inequalities involving at least two variables, of the form $X_1 < X_2$, $X_2 < X_3$, $\ldots$, $X_{k-1} < X_k$, $X_k < X_1$. Every other instance has at least one solution. If every pair of variables appears in some inequality, the instance has at most one solution.

*Example 3 (*Victorian Letters*).* Given is an alphabet with k letters, in ascending order (for English, $k = 26$). Call a letter *Victorian* if the way it is usually depicted contains a straight line, and two straight lines only ever meet at right angles. (Letters BDEFGHIJLPRTU are Victorian in the English alphabet.) Interleave the two alphabetic sequences of Victorian and non-Victorian letters, so that Victorian letters are never neighbours.

There are two solutions for the English alphabet:
BADCEKFMGNHOIQJSLVPWRXTYUZ and
ABCDKEMFNGOHQISJVLWPXRYTZU.

If the number of Victorian letters in the alphabet exceeds $(k + 1)/2$ then there are no solutions; exactly $(k+1)/2$ Victorian letters guarantees a unique solution (this can only happen when k is odd); and at most $k/2$ Victorian letters gives rise to at least two solutions.

If the instances of the constraint satisfaction problem are restricted in some way, the problem often becomes easier. Two main kinds of restriction have been studied. The first is to restrict the *structure*, the way that the constraint variables interact. If the interactions can be rearranged so that they are close to being hierarchical, then solutions can be computed efficiently using a divide-and-conquer approach. Example 3 has this kind of structure, as there are only interactions between adjacent pairs of letters. The second kind of restriction is to limit the *language*, the types of constraints that can be used to express a problem instance. When the language allowed for expressing constraints has a highly regular algebraic structure, it becomes possible to exploit this structure to compute solutions efficiently. Example 2 has a restricted language, as it can be expressed using a single kind of constraint, inequality between integers.

Essentially all structural and language restrictions have been found which give rise to problems that are of practical interest and that can be solved efficiently. It is believed that the other cases cannot be solved efficiently [4,8]. Attempts to unify the language restrictions that lead to classes that can be solved efficiently is an area of active investigation in universal algebra. This involves links to open questions about algebras with cyclic terms [2]. The frontier of structural restrictions that guarantee efficient solutions also involves links to open questions, this time in computational complexity [12].

Yet many constraint satisfaction problems cannot be expressed using purely structural or purely language restrictions. One of the most common constraints requires the solution to consist of values that are all different from each other.

Such ***all-different*** constraints are the basic building blocks of scheduling problems, of puzzles such as Futoshiki, and for expressing many kinds of mathematical objects. The all-different constraint cannot be expressed purely in terms of a language restriction, as its constraint language can express any other constraint with the right interaction of constraint variables. The all-different constraint also cannot be expressed purely in terms of a structural restriction, as it allows potentially every variable in the instance to interact with every other. The special interaction between structure and language in the all-different constraint is what makes this constraint amenable to practical algorithms, and therefore requires a restriction combining both structure and language at the same time.

In my thesis I initiate a systematic study of CSPs which are defined in terms of some combination of structure and language. I focus on those CSPs where restricting just the structure cannot describe the class of instances, and nor can restricting just the language. Such CSPs require *hybrid descriptions*, combining restrictions on structure as well as language. I consider CSPs with constraints that may involve arbitrarily many variables, with no restriction on the number of constraints allowed in a problem instance, and where the structure is not necessarily hierarchical.

In this setting, existing results on when a CSP can be decided efficiently do not apply directly. As the techniques for purely structural and purely language restrictions have taken over a decade to be resolved, I am proposing some initial ideas on how to study CSPs with hybrid descriptions. An important thread is transforming the representation of a CSP, while ensuring that the new representation falls into a class that is already well understood. In this way, many previously understood results can be unified, and some new results can be derived, to obtain useful information about classes of CSPs with hybrid descriptions.

2 Examples of constraint satisfaction problems

A constraint satisfaction problem instance informally consists of a set of ***variables*** for which we wish to assign ***values***, so that a given set of ***constraints*** is satisfied. The constraints specify which combinations of values may be assigned to specific groups of variables, and which combinations may not be assigned.

*Example 4 (*ALL-DIFFERENT*).* Suppose $u_1, u_2, \ldots, u_n$ are n variables and that for each $i = 1, 2, \ldots, n$, the variable u_i has a value from set D_i. An ALL-DIFFERENT constraint on $u_1, u_2, \ldots, u_n$ is satisfied if it is possible to assign values $a_i \in D_i$ to u_i (for each $i = 1, 2, \ldots, n$), so that the values are pairwise different.

*Example 5 (*FUTOSHIKI*).* Futoshiki is the common name for a type of logic puzzle. Given is a grid of n by n squares (typically $n = 5$ or $n = 7$). Each square takes a value from $\{1, 2, \ldots, n\}$, and there is an ALL-DIFFERENT constraint on each row and column. Further, $<$ constraints are imposed between some adjacent squares in the grid, and some of the squares may be filled in.

An instance of FUTOSHIKI is illustrated in Figure 1. (This one is reasonably difficult to solve for many humans.)

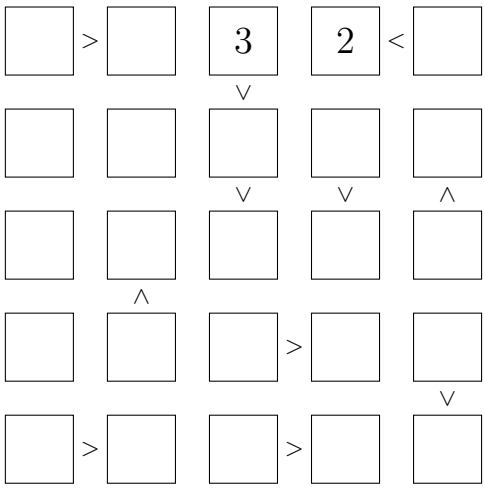


Fig. 1. An instance of FUTOSHIKI.

Although I have illustrated a particular instance of Futoshiki, note that the decision problem FUTOSHIKI allows arbitrarily large grids. This provides an infinite class of instances.

3 Contributions

Having explained some of the background, I now briefly describe the main contributions of my thesis. The descriptions assume familiarity with the theory of constraint satisfaction, and some technical terms are left undefined here.

The thesis makes extensive use of the ***microstructure*** of a CSP instance [7,11]. The microstructure contains in a single structure all allowed combinations of assignments of values to variables as specified in the instance. The microstructure complement contains all partial assignments explicitly disallowed by the constraints specified in the instance. Because the microstructure representation considers the structure and the language of an instance in a unified structure, it turns out to be helpful in the analysis of problems that have efficient algorithms for hybrid reasons.

Perfect microstructure unifies results I show that several classes of CSPs for which efficient algorithms were known, can be unified as having microstructures that are perfect graphs. This provides a surprising way to unify several quite disparate classes: constraint networks that form a tree, CSPs which have chordal

microstructures or microstructure complements, and the ALL-DIFFERENT constraint together with domain pruning. Each such CSP instance can be solved by applying the polynomial-time algorithm to find maximum independent sets in their perfect microstructure complements [9]. The ALL-DIFFERENT constraint is truly hybrid: the existence of efficient algorithms cannot be explained by either purely structural or purely language results [13]. I published this work with my supervisor Peter Jeavons at CP 2008 [14].

Broken-triangle new hybrid class With collaborators, I introduced a new class that generalizes tree structured CSPs. This class is defined by means of an excluded configuration, a *broken-triangle*, relative to the existence of a suitable variable ordering. For this class, each instance can be preprocessed to efficiently find a variable ordering which guarantees backtrack-free search. This yields the first known class that is not tractable for either structural or language reasons but for which such an ordering exists. This work led to a paper at ECAI 2008, where the paper won the best paper award [5]. This has been developed further into a paper in the journal *Artificial Intelligence* [6].

Complete representation, for analysis of width measures The edge relation in graphs is often represented as a 2-dimensional matrix. It is also possible to represent the hyperedges in a hypergraph using a k-dimensional structure, if every hyperedge contains precisely k vertices. The complete representation can then be seen as a higher-dimensional adjacency matrix, specifying for every set of k variables which values they may take. When the complete representation is at most polynomially larger than the usual representations, then it provides a useful and regular representation for tractability results. In particular, it provides worst-case examples for structural width measures.

Microstructure as product The microstructure and the microstructure complement can be seen as two kinds of products of relational structures. A product form of the microstructure was previosly mentioned in passing, for the special case of graph homomorphism [10]. I extend this product representation to general relational structures, and show how to characterize the microstructure and microstructure complement of CSPs as products. As an application, the product form yields an analysis of the few cases when the microstructure or the microstructure complement are close to tree-like, and why these cases are so uncommon.

Direct encoding as SAT By interpreting the vertices of the microstructure complement as literals, each edge in the microstructure complement can be seen as a clause in the direct encoding of a CSP instance to SAT. There is therefore a close correspondence between results about the microstructure complement and the behaviour of the direct encoding of a CSP instance as SAT. Structural results about the microstructure complement are therefore of relevance in understanding the structure of many SAT instances.

Hereditary microstructure vs. propagation Propagation is the central mechanism of constraint programming [1,3]. I discuss how domain pruning during propagation motivates the notion of hereditary CSPs. These are CSPs where any induced substructure of the microstructure of an instance is the microstructure of some other instance in the class. These CSPs correspond naturally to the partial progress of a constraint solver during search. Moreover, hereditary CSPs can be defined by a set of excluded substructures in the microstructure.

References

1. K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
2. L. Barto and M. Kozik. New conditions for Taylor varieties and CSP. In *LICS 2010: 25th Annual IEEE Symposium on Logic in Computer Science*, 2010.
3. C. Bessiere. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3, pp. 29–83. Elsevier, 2006.
4. A. Bulatov, P. Jeavons, and A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, **34**(3), pp. 720–742, 2005. doi:10.1137/S0097539700376676.
5. M. C. Cooper, P. G. Jeavons, and A. Z. Salamon. Hybrid tractable CSPs which generalize tree structure. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *ECAI 2008, Proceedings of the 18th European Conference on Artificial Intelligence, July 21–25, Patras, Greece*, Frontiers in Artificial Intelligence and Applications **178**, pp. 530–534. IOS Press, 2008. doi:10.3233/978-1-58603-891-5-530.
6. M. C. Cooper, P. G. Jeavons, and A. Z. Salamon. Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination. *Artificial Intelligence*, **174**(9–10), pp. 570–584, June 2010. doi:10.1016/j.artint.2010.03.002.
7. E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proc. AAAI-91, Anaheim, CA*, pp. 227–233, 1991.
8. M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, **54**(1), pp. 1–24, 2007. doi:10.1145/1206035.1206036.
9. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, **1**(2), pp. 169–197, 1981. doi:10.1007/BF02579273.
10. P. Hell and J. Nešetřil. *Graphs and Homomorphisms*, volume 28 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2004.
11. P. Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *Proceedings AAAI'93*, pp. 731–736, 1993.
12. D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pp. 735–744. ACM, 2010. doi:10.1145/1806689.1806790.
13. J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI'94*, pp. 362–367, 1994.
14. A. Z. Salamon and P. G. Jeavons. Perfect constraints are tractable. In *CP 2008*, volume 5202 of *Lecture Notes in Computer Science*, pp. 524–528. Springer-Verlag, 2008. doi:10.1007/978-3-540-85958-1_35.

Capturing Configuration Complexity

Evgenij Thorstensen

Oxford University Computing Laboratory, Oxford, UK
evgenij.thorstensen@comlab.ox.ac.uk

Abstract. In this paper we describe three formalisms for configuration: A constraint formalism called conditional CSPs, another such formalism known as composite CSPs, and a fragment of first-order logic called $\exists\text{FO}_{\rightarrow,\wedge,+}$. We show that composite CSPs are polynomial-time equivalent to conditional CSPs, and that since $\exists\text{FO}_{\rightarrow,\wedge,+}$ is known to be polynomial-time equivalent to conditional CSPs, all three formalisms are so equivalent. Furthermore, we show that all three formalisms are polynomial-time equivalent to classic CSPs by reducing conditional CSPs to classic CSPs.

1 Introduction

The problem of configuration [1] is to find, given a set of available components and a specification, a configuration that satisfies the specification — the configuration being a subset of components and a description of how they are to be connected. As the problem is dynamic, the CSP framework (see [2] for an introduction), while powerful, is generally seen as suboptimal for this task. Therefore, several formal systems [3,4,5] that extend the CSP framework have been proposed. These formalisms highlight several interesting aspects of the configuration task and serve to clarify the structure of practical configuration problems. In this paper, we prove them all polynomial-time equivalent.

To illustrate the formalisms we define and discuss, the following somewhat simplified and informal example will be used throughout the paper:

Example 1 (Car part configuration). A car needs, among other things, an engine, an electrical system, and an exhaust system. The engine can either be a petrol or diesel engine. Both engines need pumps and cylinders, while the petrol engine also needs spark plugs. The spark plugs, if present, must be compatible with the electrical system, while the pumps must match the cylinders.

To make this slightly more formal, assume we have a variable En with domain $\{p, d\}$, as well as variables El, Ex, Pu, Cy, Sp with constraints $C(El, Sp)$ and $D(Pu, Cy)$. However, note that a solution need not assign all the variables or satisfy both constraints.

In the following, we assume the standard definition of classic CSPs and constraint databases. Whenever it is clear from context, we denote the variables of an object α by $\mathcal{V}(\alpha)$. In particular, we will talk about the variables of a constraint (CSP, formula) and an assignment. In all the formalisms discussed, when

an assignment θ satisfies a constraint (formula) C with respect to a constraint database (usually implicit), we will write $\theta \models C$. The value of a variable v under θ is written as $\theta(v)$.

2 Three Proposed Formalisms for Configuration

2.1 Conditional CSPs

This framework was developed by Mittal and Falkenhainer [4].

Definition 1 (Conditional CSP syntax). *A conditional CSP $\langle V, V_I, C_C, C_A\rangle$ is a tuple where V is a set of variables with associated domains (not shown), $V_I \subseteq V$ a set of* initial variables*, C_C a set of constraints over V, called* compatibility constraints*, and C_A a set of* activation constraints*. Activation constraints have two forms, $A \overset{RV}{\Longrightarrow} v$ and $A \overset{RN}{\Longrightarrow} v$, with A a compatibility constraint and v a variable.*

Definition 2 (Conditional CSP semantics). *Let $\Delta = \langle V, V_I, C_C, C_A\rangle$ be a conditional CSP, and θ an assignment to a subset of V. The assignment θ satisfies Δ iff*

- $V_I \subseteq \mathcal{V}(\theta)$,
- *for every constraint $C \in C_C$ with $\mathcal{V}(C) \subseteq \mathcal{V}(\theta)$, we have that $\theta \models C$,*
- *for every constraint $(A \overset{RV}{\Longrightarrow} v) \in C_A$ with $\mathcal{V}(A) \subseteq \mathcal{V}(\theta)$ and $\theta \models A$, we have that $v \in \mathcal{V}(\theta)$, and*
- *for every constraint $(A \overset{RN}{\Longrightarrow} v) \in C_A$ with $\mathcal{V}(A) \subseteq \mathcal{V}(\theta)$ and $\theta \models A$, we have that $v \notin \mathcal{V}(\theta)$.*

In [4], this formalism is called dynamic CSP, but it has since been renamed due to name collisions.

Example 2. We can represent the problem in Example 1 as a conditional CSP $\langle V, V_I, C_C, C_A\rangle$ by letting $V = \{En, El, Ex, Pu, Cy, Sp\}$, $V_I = \{En, El, Ex\}$, $C_C = \{C(El, Sp), D(Pu, Cy)\}$, and

$$C_A = \left\{ \begin{array}{c} (En = p) \overset{RV}{\Longrightarrow} Sp, (En = p) \overset{RV}{\Longrightarrow} Pu, (En = p) \overset{RV}{\Longrightarrow} Cy, \\ (En = d) \overset{RV}{\Longrightarrow} Pu, (En = d) \overset{RV}{\Longrightarrow} Cy \end{array} \right\}$$

.

2.2 Composite CSPs

This formalism was introduced informally by Sabin and Freuder [5]. A composite CSP is a CSP where variables can have subproblems (other CSPs) as values. If such a variable v is assigned a subproblem T as the value, any constraint containing v is removed and the constraints and variables in T are added to the CSP. An attempt at formalizing this is below.

Definition 3 (Composite CSP syntax). *A composite CSP is a pair $\langle S, < \rangle$ with $S = \{T_1, \ldots, T_n\}$ a set of classic CSPs, and $<$ a strict partial order on S with a single minimal element T_r. If $T_i < T_j$, then any variable $v \in \mathcal{V}(T_i)$ may contain T_j as a domain element.*

We will abuse notation and identify a composite CSP $\langle S, < \rangle$ with the set S whenever convenient.

Definition 4 (Variables and constraints). *Let S be a composite CSP. The set of variables in S is the set $\mathcal{V}(S) = \bigcup\{\mathcal{V}(T) \mid T \in S\}$. Likewise, the set of constraints in S is the set $\mathcal{C}(S) = \bigcup\{C \mid \langle V, C \rangle \in S\}$.*

Definition 5 (Subproblem variables). *Let S be a composite CSP and θ an assignment to a subset of $\mathcal{V}(S)$. The set of subproblem variables in θ, denoted $SV(\theta)$, is the set of variables $v \in \mathcal{V}(\theta)$ such that $\theta(v) = T$ for some $T \in S$.*

In other words, $SV(\theta)$ is the set of variables assigned subproblems as values, and we have to treat them in a special way.

Definition 6 (Composite CSP semantics). *Let S be a composite CSP with minimal element T_r, and θ an assignment to a subset of $\mathcal{V}(S)$. The assignment θ satisfies S iff $\theta \models T_r$, and we define $\theta \models T$ for any $T \in S$ recursively as follows:*

- *$\mathcal{V}(T) \subseteq \mathcal{V}(\theta)$,*
- *for every constraint C in T with $\mathcal{V}(C) \cap VS(\theta) = \emptyset$, we have that $\theta \models C$, and*
- *for every $v \in \mathcal{V}(T) \cap VS(\theta)$, we have that $\theta \models \theta(v)$.*

In other words, we have to deal with any subproblem that we have assigned to a variable in our initial problem (recursively), but we do not need to check constraints containing such variables, as they are replaced by constraints from the subproblem.

Example 3. We can represent the problem in Example 1 as a composite CSP $S = \{S_1 = \langle V_1, C_1 \rangle, S_2 = \langle V_2, C_2 \rangle, S_3 = \langle V_3, C_3 \rangle\}$, where

- $V_1 = \{En, El, Ex\}$, $C_1 = \emptyset$, and the domain of En is $\{S_2, S_3\}$,
- $V_2 = \{El, Cy, Pu, Sp\}$ and $C_2 = \{C(El, Sp), D(Pu, Cy)\}$, and
- $V_3 = \{Cy, Pu\}$ and $C_3 = \{D(Pu, Cy)\}$.

The minimal element is S_1, with $S_1 < S_2$ and $S_1 < S_3$.

2.3 $\exists\mathbf{FO}_{\rightarrow,\wedge,+}$

This fragment of first-order logic defined by Gottlob, Greco, and Mancini [3] gives a fully logical, as opposed to constraint-based characterization of configuration problems.

Definition 7 ($\exists$FO$_{\to,\wedge,+}$ syntax). *An $\exists FO_{\to,\wedge,+}$ sentence is a formula of the form*

$$\exists \vec{x}. \bigwedge_{1 \le i \le n} (\phi_i(\vec{x}) \to \exists \vec{y}. \psi_i(\vec{x}, \vec{y}))$$

where $\vec{x}$, $\vec{y}$ are lists of variables, $n \ge 0$, and ϕ_i, ψ_i are conjunctions of atoms, possibly empty.

Definition 8 ($\exists$FO$_{\to,\wedge,+}$ semantics). *Let ϕ be an $\exists FO_{\to,\wedge,+}$ sentence, and θ an assignment to a subset of $\mathcal{V}(\phi)$. Denote by ϕ' the formula obtained from ϕ by replacing any atom R such that $\mathcal{V}(R) \not\subseteq \mathcal{V}(\theta)$ by $\bot$ (meaning false). The formula ϕ is satisfied by θ iff $\theta \models \phi'$.*

Example 4. We can represent the problem in Example 1 as an $\exists$FO$_{\to,\wedge,+}$ sentence (outermost conjunction omitted)

$$\exists En, El, Ex. \begin{pmatrix} T(El) \wedge T(Ex) \wedge T(En) \\ (En = d) \to \exists Pu, Cy. D(Pu, Cy) \\ (En = p) \to \exists Pu, Cy, Sp. (C(El, Sp) \wedge D(Pu, Cy)) \end{pmatrix}$$

where the predicate T is a new predicate that allows all values in the domain of a variable. Since there are no constraints on some of the required variables, we need it to make sure that they are present in any satisfying assignment.

3 Reductions

In this section we show that there exist polynomial-time many-one reductions between composite CSPs and conditional CSPs, and also between conditional CSPs and classic CSPs. Since the existence of such reductions between conditional CSPs and $\exists$FO$_{\to,\wedge,+}$ was shown in [3], all four are polynomial-time equivalent.

Theorem 1. *Conditional CSPs and composite CSPs reduce to each other in polynomial time.*

The proof is split into two independent parts.

Proof (Composite CSPs to conditional CSPs). Let S be a composite CSP with minimal element $T_r \in S$. First, create for every variable $v \in \mathcal{V}(S)$ a new variable v' with the same domain. Denote these new variables by $\mathcal{V}'(S)$, and let this extend to the subproblems: Given $T \in S$, we write $\mathcal{V}'(T)$ for the new variables of T.

We construct a conditional CSP $\langle V, V_I, C_C, C_A \rangle$ by letting $V = \mathcal{V}(S) \cup \mathcal{V}'(S)$ and $V_I = \mathcal{V}'(T_r)$. For every $T \in S$, let every constraint C in T be in C_C. To handle the activation and deactivation of variables that are assigned subproblems, we will for every variable $v \in \mathcal{V}(S)$ with corresponding new variable $v' \in V'(S)$ create an activation constraint

$$\left(\bigwedge_{T \in S} v' \neq T \right) \overset{RV}{\Longrightarrow} v$$

and a compatibility constraint $v = v'$. The former states that the variable v and the constraints that include it need only be considered if the corresponding new variable v' is not assigned to a subproblem, while the latter makes the proof easier. Furthermore, we will for every $T \in S$ and every $w' \in \mathcal{V}'(T)$ create activation constraints $(v' = T) \Rightarrow w'$. These constraints state that whenever a new variable is assigned to a subproblem, the new variables of that subproblem become active.

Let S be a composite CSP. The conditional CSP $\langle V, V_I, C_C, C_A \rangle$ we create is bounded by $|V| = 2|\mathcal{V}(S)|$, $|C_C| = |\mathcal{C}(S)| + |\mathcal{V}(S)|$, and $|C_A| = |\mathcal{V}(S)| + |\mathcal{V}(S)|^2 \times |S|$. If we have a satisfying assignment θ to the conditional CSP, we can obtain a solution to the composite CSP by deleting all the new variables. In the other direction, given a satisfying assignment for the composite CSP, we can obtain a satisfying assignment to the conditional CSP if we let $\theta(v') = \theta(v)$ for every $v \in \theta$. □

Proof (Conditional CSPs to composite CSPs). Let $\langle V, V_I, C_C, C_A \rangle$ be a conditional CSP. We can encode it as a composite CSP by creating for every $v \in V$ a CSP $T_v = \langle \{v\}, \emptyset \rangle$ and an intermediate variable w_v with domain $\{\bot, T_v\}$ that we add to the minimal element T_r.

Furthermore, for every compatibility constraint $C \in C_C$ we will create a CSP $T_{\mathcal{V}(C)} = \langle \mathcal{V}(C), C \rangle$ and a variable $w_{\mathcal{V}(C)}$ with domain $\{\bot, T_{\mathcal{V}(C)}\}$. We also add a new constraint $\left(\bigwedge_{v \in \mathcal{V}(C)} w_v = T_v \right) \rightarrow w_{\mathcal{V}(C)} = T_{\mathcal{V}(C)}$ to T_r saying that the constraint C needs to be satisfied only if the variables in it are present.

Likewise, for every activation constraint $(A \overset{RV}{\Longrightarrow} v) \in C_A$ we create a CSP $T_{\mathcal{V}(A)}$ (if it doesn't already exist) and a variable $w_{\mathcal{V}(A)}$ with domain $\{\bot, T_{\mathcal{V}(A)}\}$. We add $\mathcal{V}(A)$ to the variables of $T_{\mathcal{V}(A)}$ and also create a new constraint $A \rightarrow w_v = T_v$ that we add to the constraints of $T_{\mathcal{V}(A)}$. To connect this new problem to the other problems, we add $w_{\mathcal{V}(A)}$ to T_r and post a constraint $\left(\bigwedge_{v \in \mathcal{V}(A)} w_v = T_v \right) \rightarrow w_{\mathcal{V}(A)} = T_{\mathcal{V}(A)}$ on this variable. For each activation constraint $A \overset{RN}{\Longrightarrow} v \in C_A$ we do the same as above, but instead of $A \rightarrow w_v = T_v$ we create the constraint $A \rightarrow w_v = \bot$. Finally, for every $v \in V_I$ we add a constraint $w_v = T_v$ to T_r, expressing the fact that the initial variables must be present in any satisfying assignment.

The composite CSP S that we create is bounded by $|\mathcal{V}(S)| = 2|V| + |C_C| + |C_A|$ and $|\mathcal{C}(S)| = 2|C_C| + 2|C_A| + |V_I|$. Let $\Delta = \langle V, V_I, C_C, C_A \rangle$ be a conditional CSP and S be the composite CSP constructed from it. Assume we have a satisfying assignment θ for S. First, $V_I \subseteq \mathcal{V}(\theta)$ as T_r contains for every $v \in V_I$ a constraint $w_v = T_v$, and since $\theta \models w_v = T_v$, we must have that $\theta \models T_v$, and thus that $v \in \mathcal{V}(\theta)$.

Secondly, we need to consider any compatibility constraint $C \in C_C$ with $\mathcal{V}(C) \subseteq \mathcal{V}(\theta)$ and $\theta \not\models C$. Such a constraint is contained in a subproblem $T_{\mathcal{V}(C)}$,

and $\theta \not\models C$ implies that for some $v \in \mathcal{V}(C)$ we have $\theta(w_v) = \bot$, due to the constraint on $w_{\mathcal{V}(C)}$. However, this means that we can remove v from $\mathcal{V}(\theta)$ to obtain a smaller satisfying assignment for S that does not need to satisfy C.

Finally, we need to consider any activation constraint $(A \overset{RV}{\Longrightarrow} v) \in C_A$ with $\mathcal{V}(A) \subseteq \mathcal{V}(\theta)$ such that $\theta \models A$ and $v \notin \mathcal{V}(\theta)$. This turns out to be impossible, since we have in T_r a constraint $A \to w_v = T_v$. This gives us that $\theta(w_v) = T_v$ and therefore $v \in \mathcal{V}(\theta)$. For a constraint $(A \overset{RN}{\Longrightarrow} v) \in C_A$ we can do a proof similar to the above, except that $\theta(w_v) = \bot$, which means that we can remove v from $\mathcal{V}(\theta)$ to obtain a smaller satisfying assignment for S that does satisfy $A \overset{RN}{\Longrightarrow} v$. □

Theorem 2. *Conditional CSPs and classic CSPs reduce to each other in polynomial time.*

Proof. As any CSP $\langle V, C\rangle$ is a conditional CSP $\langle V, V, C, \emptyset\rangle$, we need only prove the other direction. Therefore, let $\langle V, V_I, C_C, C_A\rangle$ be a conditional CSP. Create a boolean *activation variable* a_v for every $v \in V$ to keep track of variable activation. Then, create for every $C \in C_C$ a constraint $(\bigwedge_{v\in\mathcal{V}(C)} a_v) \to C$ saying that C needs be considered only if the variables in it are active.

For any activation constraint $A \overset{RV}{\Longrightarrow} v$, create the constraint $A \to a_v$, and for $A \overset{RN}{\Longrightarrow} v$ create the constraint $A \to \neg a_v$. Finally, create a constraint $\bigwedge_{v\in V_I} a_v$ stating that the initial variables are active. The CSP $\langle C', V'\rangle$ that we have constructed is bounded by $|V'| = 2|V|$ and $|C'| = |C_C| + |C_A| + |V_I|$.

Let θ be a satisfying assignment for this CSP. We can obtain a satisfying assignment for the conditional CSP by removing from $\mathcal{V}(\theta)$ any v such that $\theta(a_v) = \bot$ and all the activation variables.

In the other direction, if θ is a satisfying assignment to the conditional CSP, we can obtain a satisfying assignment θ' to the CSP constructed as follows: For every $v \in \mathcal{V}(\theta)$ we set $\theta'(v) = \theta(v)$ and $\theta'(a_v) = \top$. For every other variable w in the CSP, we set $\theta'(w)$ to a random element in the domain of w and $\theta'(a_w) = \bot$. As all constraints are conditional on the activation variables, the assignment to these variables does not affect satisfiability. □

Acknowledgements. We would like to thank Markus Aschinger, Peter Jeavons, and András Z. Salamon for helpful comments on drafts of this paper.

References

1. Mittal, S., Frayman, F.: Towards a generic model of configuraton tasks. In: Proc. IJCAI. (1989) 1395–1401
2. Dechter, R.: Constraint Processing. Morgan Kaufmann Publishers (2003)
3. Gottlob, G., Greco, G., Mancini, T.: Conditional constraint satisfaction: Logical foundations and complexity. In: Proc. IJCAI. (2007) 88–93
4. Mittal, S., Falkenhainer, B.: Dynamic constraint satisfaction problems. In: Proc. AAAI. (1990) 25–32
5. Sabin, D., Freuder, E.C.: Configuration as composite constraint satisfaction. In: Proc. AI and Manufacturing Research Planning Workshop. (1996) 153–161

Two Preferences Based Conversational Recommender Systems

Walid Trabelsi[1], Nic Wilson[1] (supervisor) and Derek Bridge[2] (supervisor)

[1]Cork Constraint Computation Centre, [2]Department of Computer Science
University College Cork, Ireland
w.trabelsi@4c.ucc.ie, n.wilson@4c.ucc.ie, d.bridge@cs.ucc.ie

1 Introduction

In an era of overwhelming choice, *recommender systems* are a new source of assistance, helping their users to decide which goods, services or information to purchase or consume [1]. These systems infer user preferences from data gathered either explicitly, e.g., product ratings, or implicitly by observing user behaviour.

Conversational recommender systems typically involve iteratively showing the user a small set of options (e.g., products) for them to choose between. In order to choose an appropriate set to display at each stage, from a much larger collection of options, it is useful to have information regarding which options are likely to be preferred to others by the user, based on previous responses they have given in the dialogue; if one assumes that the user has some kind of preference relation over products, this amounts to determining if certain products are dominated according to this preference relation.

In 2007, Bridge & Ricci introduced a new kind of *conversational recommendation*, which they call *Information Recommendation* [2]. We describe Information Recommendation in Section 5. We use it as the setting in which we explore different models of user preferences.

Two instances of such framework are developed. The first (see Section 3) is based on a simple quantitative preferences formalism, involving a sum of weights, with a language of linear inequalities. The second (see Section 4) is a qualitative preference formalism, where models are a kind of generalised lexicographic order, and the inputs are comparative preference statements. Experimentations are described in Section 6.

2 A Framework for Dominance of Preferences

We assume a set Ω of configurations, representing possible user's choices.

We would like to generate some kind of (partially ordered) preference relation on Ω, based on previous information we have received during the interaction with the user. Using this preference relation we can say that the set of products satisfying a configuration are preferred to other products that are satisfying another configuration.

Configurations generated from features. We assume a collection of features $V = \{F_1, \ldots, F_n\}$. Define a configuration α to be a mapping from $\{1, \ldots, n\}$ to $\{1, 0\}$ (α can be represented by a bit set, in case of very big number of features other efficient representation are described in [3]). Configuration α can also be thought of a set of features, i.e., all features F_i such that $\alpha(i) = 1$. Products are judged on the features they possess, and so

can be mapped to configurations. We assume that having a feature is always at least as good as failing to have it. The set of outcomes is then the set of products that satisfy the configuration and are available to the user (e.g.,in a database). For convenience the configuration is denoted by $(1,0,1)$ as $f_1\bar{f}_2f_3$.

3 Sum of Weights Model of the User

This model assumes that a user assigns a weight to each feature, and configurations are compared on the sum of weights of the associated set of features.

The set of models is the set of all vectors of weights $w = (w_1, \ldots, w_n)$, where w_i is a non-negative real number. w_i can be considered as a weighting assigned to feature F_i. Given a weights vector w, the overall value $w(\alpha)$ of a configuration α is the sum of weights of the features included, i.e., $w(\alpha) = \sum_{i:\alpha(i)=1} w_i$. This is used to define the ordering on configurations. We define the preference relation $\succcurlyeq_w$ for model w by $\alpha \succcurlyeq_w \beta$ if and only if $w(\alpha) \geq w(\beta)$, i.e., iff $\sum_i w_i(\alpha(i) - \beta(i)) \geq 0$.

Constraint language: this consists of statements of the form $\alpha \geq \beta$, where α and β are configurations. Weight vector w is defined to satisfy $\alpha \geq \beta$ if $\alpha \succcurlyeq_w \beta$. Let Φ be a set of input constraint statements. The induced preference relation given Φ is such that : for outcomes α and β, $\alpha \succcurlyeq_\Phi \beta$ if and only if $\alpha \succcurlyeq_w \beta$ for all weight vectors w satisfying Φ. Dominance relation $\succ_\Phi$ is then the strict part of $\succcurlyeq_\Phi$. When comparing with the second semantics, we also use the notation $\succcurlyeq_\Phi^{sw}$ for $\succcurlyeq_\Phi$.

Example 1. Let Φ be two constraints: $f_1\bar{f}_2f_3 \geq \bar{f}_1f_2f_3$, and $f_1f_2\bar{f}_3 \geq \bar{f}_1f_2f_3$. Let α be the configuration $f_1\bar{f}_2\bar{f}_3$, and let β be the configuration $\bar{f}_1f_2f_3$. Weights vector w satisfies the constraint $f_1\bar{f}_2f_3 \geq \bar{f}_1f_2f_3$ if and only if $w(f_1\bar{f}_2f_3) \geq w(\bar{f}_1f_2f_3)$, i.e., $w_1 + w_3 \geq w_2 + w_3$, which holds if and only $w_1 \geq w_2$. By similar reasoning, w satisfies Φ if and only if $w_1 \geq w_2$ and $w_1 \geq w_3$. Also, w satisfies $\alpha \geq \beta$ if and only if $w_1 \geq w_2 + w_3$. Thus Φ does not entail $\alpha \geq \beta$, so we do not have $\alpha \succcurlyeq_\Phi^{sw} \beta$, since, for example, weights vector w with $w_1 = 4$, $w_2 = 2$ and $w_3 = 3$ satisfies Φ but does not satisfy $\alpha \geq \beta$.

Computational aspects: We wish to determine if $\alpha \succcurlyeq_\Phi \beta$, for given configurations α and β. Let *Pos* be the set of constraints $w_i \geq 0$, for $i = 1, \ldots, n$, representing the non-negativity of the weights (and corresponding to the assumption that including a feature is always at least as good as not including it). The definition implies that $\alpha \succcurlyeq_\Phi \beta$ if and only if the linear constraints $\Phi \cup Pos$ (over real-valued variables w_i) entail the constraint $\sum_i(\alpha(i) - \beta(i))w_i \geq 0$.

For implementation of this using a Linear Programming solver, it can be convenient to express it as a linear optimization problem. Define A_{min} to be the minimum value of $\sum(\alpha(i) - \beta(i))w_i$ subject to constraints $\Phi \cup Pos$. It can be easily shown that $\alpha \succcurlyeq_\Phi \beta$ if and only $\Phi \cup Pos$ entails $\sum_i(\alpha(i) - \beta(i))w_i \geq 0$ if and only if $A_{min} \geq 0$.

4 Comparative Preferences Model of the User

In this scenario, user models are a kind of generalised lexicographic order, called *cp-trees* [4] (see Figure 1), which are similar to search trees used for solving constraint satisfaction problems. Two configurations α and β are compared first on the most important variable. If they do not agree on that variable then the comparison is settled: If α contains the feature and β does not, then α is better than β.

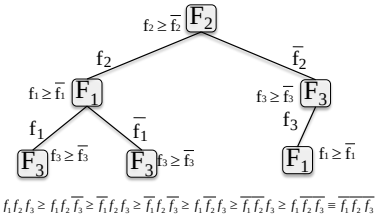


Fig. 1. A cp-tree σ, along with its associated ordering on outcomes $\succcurlyeq_\sigma$.

The constraint Language: The preference constraint language will include statements that compactly express comparative (and sometimes conditional) preferences among configurations. Each statement φ in the language has an interpretation φ^*, which is a relation between configurations, giving the direct implications of φ regarding preferences between configurations. We say that total pre-order $\succcurlyeq$ satisfies φ if and only if $\succcurlyeq$ extends φ^*, i.e., $(\alpha, \beta) \in \varphi^*$ implies $\alpha \geq \beta$. The language will include only comparative preference statements φ of the form $p \geq q \parallel T$, where P, Q and T are sets of features, and p is an assignment to P (i.e., a function from P to $\{0,1\}$), and q is an assignment to Q. Informally, the statement $p \geq q \parallel T$ represents the following: p is preferred to q if T is held constant.

Computation of Preference: Given set of input constraint statements Φ, and configurations α and β we can determine in polynomial time whether or not $\alpha \succcurlyeq_\Phi \beta$ holds, using the algorithm given in [4], as shown by Theorem 1 in [4].

5 Application to Conversational Recommender Systems

We compare the Sum of Weights Model and the Comparative Preferences Model in Information Recommendation described in [2]. In this setting, a user repeatedly edits and resubmits a query (i.e., configuration) until she finds a product (outcome) that she wants. The recommender system: observes the user's actions; infers user-related constraints on the preferred configurations; uses these inferences to deduce which queries a user is likely to try next. In [2], users are represented by the Sum Of Weights Model.

The interaction between the user and the recommender system proceeds as follows:

1. The recommender system analyzes q, with particular regard to differences between current query q and the previous query the user submitted. (In the case where q is the very first query, the previous query is the empty set.) The system induces some additional preference constraint statements to add to inputs Φ.
2. The recommender system generates a set of candidate next possible queries and prunes this set to those that are satisfiable and undominated (see below).
3. The user chooses and submits her next query. This becomes the new current query q. In the experiments reported in the next section, we arrange that the user always chooses one of the queries that the system advises.

Steps 1–3 are repeated until the user is satisfied with q or the set of undominated, satisfiable candidates is empty, in which case q cannot be bettered. At this point, the user can request to see the products that satisfy q.

The goal of the recommender system is to give the advice that has the greatest value. We consider this to be that which minimizes the total quantity of advice given and the dialogue length, while guiding the user to the best product.

During step 2 above, the system computes the following three sets of queries:

- *Candidates:* Candidate queries are ones which are close, in a particular sense, to the current query q. For example, if $f_i \notin q$, the set of candidates will include the query that results from adding just feature f_i to q.
- *Satisfiables:* The system eliminates from *Candidates* those queries which are unsatisfiable. A query is satisfiable if and only if there exists a product which has all the features in the query.
- *Undominated*: The system eliminates from *Satisfiables* each query which is dominated by (i.e., worse than) some other *Satisfiable* query; the remaining set of queries is called *Undominated*. The dominance relation is based on what is induced in step 1 above. The rationale is to exclude from the system's advice queries that, on the basis of the user's preferences, it thinks the user would regard as inferior.

Generating induced preference statements for sum of weights model If the user has added feature f_i to query q, then statements $q^i \geq q^j$ are induced for all $f_j \notin q$, $i \neq j$ unless $Add(q, f_j) = q^j$ is unsatisfiable. This implies that the weight vector satisfies $w_i \geq w_j$.

Generating induced preference statements for comparative preferences model Consider the situation where the user has chosen to add feature f_i instead of feature f_j. There is more than one way one might induce a comparative preference statement from this decision by the user. We consider two, each being a kind of counterpart for the constraint $w_i \geq w_j$ induced for the sum of weights approach.

- **Basic:** Let q be the current query, let q^i be the current query q with the feature f_i added, and let q^j be q with the feature f_j added. A basic, somewhat conservative, approach is to just model the preference of feature i over feature j by the preference statement: $q^i \geq q^j || \emptyset$, i.e., $q^i \geq q^j$, which just expresses a preference for q^i over q^j.
- **Importance:** Less conservatively, we can induce $f_i \geq \overline{f_i} || V \setminus \{F_i, F_j\}$, which says that the presence or not of the feature F_i is more important than the choice of F_j. Thus, whatever the state of the feature F_j in the query the user will prefer F_i to be present in the query so that this feature is included in the best product.

6 Experiments

In this section, we report experiments with simulated users that demonstrate the feasibility of using both the Sum of Weights Model and the Comparative Preferences Model within the conversational recommender system that we described above. We use one product database describing hotels by their amenities expressed as Boolean features such as *airport shuttle*, *pets type*, etc. The Trentino-10 database records 10 features about 4056 hotels, of which 133 are distinct.

The simulated users behave in the following somewhat idealized way: within a dialogue, they do not try queries that they have tried earlier in the dialogue; they are aware of their own preferences and never choose a next query that would be inferior to the current one. The user's true preferences are represented either in the Sum of Weights Model by randomly generating weight vectors over features or in the Comparative Preferences Model by randomly generating cp-trees over features.

In the experiments, we pair the users, which select the next query to execute either using a Sum of Weights Model or a Comparative Preferences Model, with each of several recommender systems. One recommender system uses the Sum of Weights Model. Six use the Comparative Preferences Model, differing first on which of the two alternative preference statements they infer (Basic or Importance), and on their value for γ (1, 2 or 3). γ is one parameter of the procedure in [4] and represents the maximum number of variables one node of the cp-tree can contain. For each pairing of a user with a recommender system, we ran 500 simulated dialogues.

In the experiments we compare the pruning rate achieved,it is defined as follows:

$$pruning\ rate = \frac{|Satisfiables \setminus Undominated|}{|Satisfiables|} \times 100$$

It shows the extent to which an approach eliminates what it takes to be inferior satisfiable candidate queries from its advice. Other things being equal, the shorter the advice the better, as this reduces the choice the user has to make. The results in the case where the users' true preferences are represented in the Sum of Weights Model are shown in the table below.

Table 1. The pruning rates (users represented in Sum of Weights Model)

	γ=1	γ=2	γ=3
Trentino-10			
Comp. Prefs. Basic	87.49	16.51	13.98
Comp. Prefs. Importance	87.42	87.57	86.72
Sum of Weights	85.72		

It shows that, in nearly all settings, the Comparative Preferences approach is pruning non-optimal queries a little more than the Sum of Weights approach.

What is also of concern from a practical point of view is the average length of the advice that the system gives; this is inversely related to the pruning rate. Except in the cases where pruning is very low (Basic with $\gamma = 2$ or 3), advice from the Sum of Weights recommenders is very slightly longer than it is in the case of the Comparative Preferences recommenders. Dialogue lengths are very similar in the case of all recommenders: around 6 steps on average.

Where the user's true preferences are represented in the Sum of Weights Model, we have also measured the amount by which the utility of the product that the user ultimately chooses falls short of the utility of the best product that she could have reached, normalized by the difference between the products of highest and lowest utility. Unsurprisingly, these follow a similar pattern to the percentage agreements reported in the previous paragraph. The values are very close to zero, ranging from 0 to 0.008.

7 Preference-based constrained optimization

For the case of configurable products, the set Ω of possible products could be expressed implicitly as the (potentially extremely large) set of solutions of a Constraint Satisfaction Problem (CSP). We suppose that the user preferences are expressed as a set Γ of comparative preferences statements. We are interested in the problem of generating optimal elements of Ω, i.e., solutions of the CSP which are undominated with respect to the comparative preferences. For this task a Branch and Bound algorithm can be used.

Using the methods of [5], we can restrict the variable and value orderings to ensure that a generated search tree is compatible with the preferences Γ. The first solution generated by such a search will be optimal. For generating further optimal solutions it is important—especially for the situation when Ω is extremely large—to be able to find prune subtrees consisting of dominated solutions. We are currently developing sufficient conditions for such pruning, based on the dominance procedure described in [4].

8 Discussion

There has been a lot of excellent theoretical work produced on comparative preference formalisms in recent years, for example, the award winning papers [6, 7]; however, development towards applications has been lagging somewhat.We have shown how a comparative preferences approach can be adapted for a conversational recommender system. We are developing sufficient conditions for selecting optimal solutions when they are expressed implicitly as the solutions of a XSP. In future work, we will extend these approaches for other kinds of recommender systems, including for non boolean features. We will integrate the preference-based constrained optimization framework into a configurator in order to select the most preferred products.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering **17**(6) (2005) 734–749
2. Bridge, D., Ricci, F.: Supporting product selection with query editing recommendations. In: RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems, New York, NY, USA, ACM (2007) 65–72
3. Haenni, R., Lehmann, N.: Implementing belief function computations. Technical Report 01-28, Department of Informatics, University of Fribourg (2001)
4. Wilson, N.: An efficient deduction mechanism for expressive comparative preference languages. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-09). (2009) 961–966
5. Wilson, N.: Consistency and constrained optimisation for conditional preferences. In: Proceedings of ECAI-04. (2004) 888–892
6. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H., Poole, D.: CP-nets: A tool for reasoning with conditional *ceteris paribus* preference statements. Journal of Artificial Intelligence Research **21** (2004) 135–191
7. Koriche, F., Zanuttini, B.: Learning conditional preference networks with queries. In: Proc. IJCAI 2009. (2009) 1930–1935