Achieving High Quality Knowledge Acquisition using Controlled Natural Language

Tiantian Gao*

Department of Computer Science,
Stony Brook University, Stony Brook, NY, USA
`tiagao@cs.stonybrook.edu`

Abstract. Controlled Natural Languages (CNLs) are efficient languages for knowledge acquisition and reasoning. They are designed as a subset of natural languages with restricted grammar while being highly expressive. CNLs are designed to be automatically translated into logical representations, which can be fed into rule engines for query and reasoning. In this work, we build a knowledge acquisition machine, called KAM, that extends Attempto Controlled English (ACE) and achieves three goals. First, KAM can identify CNL sentences that correspond to the same logical representation but expressed in various syntactical forms. Second, KAM provides a graphical user interface (GUI) that allows users to disambiguate the knowledge acquired from text and incorporates user feedback to improve knowledge acquisition quality. Third, KAM uses a paraconsistent logical framework to encode CNL sentences in order to achieve reasoning in the presence of inconsistent knowledge.

Keywords: Knowledge Acquisition, Controlled Natural Language, Logic Programming

1 Introduction

Much of human knowledge can be represented as rules and facts, which can be used by rule engines (e.g., Prolog [23], Clingo [10], IDP [6]) to conduct formal logical reasoning in order to derive new conclusions, answer questions, or explain the validity of true statements. However, rules and facts extracted from human knowledge can be very complex in the real world. This will demand domain experts to spend a lot of time on understanding the rule systems in order to write logical rules. CNLs emerge as better knowledge acquisition systems over rule systems in that they can acquire knowledge from text and represent the text in logical forms for reasoning. CNLs are designed based on natural languages, but with restricted grammar to avoid ambiguities while being highly expressive. Representative languages include ACE [8], Processable English (PENG) [25], BioQuery-CNL [7]. In general, CNL systems provide a GUI for user to enter CNL text. The language parser checks the grammar of the text and sends back

* The author is advised by Michael Kifer and Paul Fodor from Stony Brook University.

suggestions for correction to the user. CNL text is then mapped into the corresponding logic programs based on the syntax and semantics of the underlying rule engine in order to perform question answering tasks.

Though the aforementioned systems have good intent of design, we found that there are several limitation in current CNL systems. First, they have limited ability to identify sentences that express the same meaning but in various syntactical forms. For instance, ACE translates sentences `Mary owns a car` and `Mary is the owner of a car` into two different logical representations. As a result, if the first sentence is entered into the knowledge base, the reasoner will fail to answer the question `who is the owner of a car`. However, in the real world, it is very common that the user writes questions in a different way from the author who composes the knowledge base. Second, current CNL systems do not accept inconsistent knowledge to occur. In other words, once inconsistent information is found, the underlying rule engine will break and not be able to conduct any inference tasks. In our view, inconsistency is very likely to happen when the knowledge base is formed by merging multiple resources together. Hence, it is useful to design a paraconsistent logical framework that can reason in the presence of inconsistent knowledge. Third, there is no way for the user to edit or audit the acquired knowledge. CNL systems are not guaranteed to always return the user-expected results. As a result, it is necessary to provide a mechanism for the user to edit the acquired knowledge as opposed to re-write sentences many times in order to meet the requirement.

In this work, we design a knowledge acquisition system, KAM, that achieves three goals. First, KAM performs deep semantic analysis of English sentences and maps sentences that express the same meaning via different syntactic forms to the same standard logical representations. Second, KAM performs valid logical inferences based on the facts and rules extracted from English sentences and achieves inconsistency-tolerance for query answering. Third, KAM builds an environment to assist users with entering and disambiguating English texts. In the following parts, Section 2 shows the background knowledge of the natural language processing tools KAM uses in the knowledge acquisition process, Section 3 describes the architecture of the system, Section 4 shows the current state of research and discusses some open issues, and Section 5 concludes the paper.

2 Background

In this section, we provide the background of linguistic databases, semantic relation extraction, and word similarity measures in the field of natural language processing in order to help readers better understand KAM.

2.1 Linguistic Databases

KAM uses a lexical database, BabelNet, and a frame-relation database, FrameNet, in the process of knowledge acquisition. A lexical database is a database of words. It contains information of part-of-speech, word sense, synset and word relation.

WordNet [20] is one of the famous ones, where each word is defined with a list of word senses. Words that share similar meanings are grouped as a synset. Synsets are connected by semantic relations. For instance, the hypernym relation says that one synset is a more general concept of the other, i.e., human is the hypernym of homo sapiens. WordNet is rich in word knowledge, but it lacks sufficient information of named entities. DBPedia [1], WikiData [28], and YAGO [26] are databases of entities, where each is defined with a set of properties and the relations with other entities or with some pre-defined ontological classes. However, there is no link between an entity in an entity database like DBPedia and a concept in WordNet. As a result, there is no way to find their semantic relations which is useful in many cases. BabelNet solves this problem by integrating multiple knowledge bases, including WordNet, DBPedia, Wikidata, etc. Besides, it automatically finds the mapping across different knowledge bases and therefore bridges the gap between concepts and entities.

FrameNet is a database representing entity relations using *frames*. A frame consists of a set of *frame elements* and *lexical units*. A frame element denotes an entity that serves a particular semantic role in a frame relation. Frame elements are frame-specific. Therefore, they are not shared among frames. A lexical unit indicates a target word in a sentence that triggers a frame relation. For example, the sentence `Mary works for IBM as an engineer` semantically entails the `Being_Employed` frame relation, where `work` is the lexical unit and `Mary`, `IBM`, and `engineer` represent the `Employee`, `Employer`, and `Position` frame elements respectively. In FrameNet, each lexical unit is associated with a set of *valence patterns* and *exemplar sentences*. Valence patterns show the *grammatical functions* [14] of each frame element with respect to the lexical unit. For the above sentence, `Mary` is the `external` of `work`, and `IBM` and `engineer` are the `dependent` of the prepositional modifiers of `work`. Exemplar sentences are the sample English sentences that realize the valence patterns.

In addition to FrameNet, VerbNet [24] and PropBank [15] are also databases of entity relations. VerbNet and PropBank are purely verb-oriented. Therefore, they cannot recognize noun-, adjective-, or adverb-triggered relations. Besides, since VerbNet and PropBank group verbs based on the syntactic patterns of verbs with respect to the entities, verbs that belong to the same class may not represent the same meaning. The advantage of VerbNet over FrameNet is that VerbNet assigns a WordNet synset ID to each verb. Additionally, it defines an ontology that defines the semantic restrictions for entities that can serve particular semantic roles in an entity relation. In KAM, we use FrameNet augmented with BabelNet synset IDs for each frame element and lexical unit.

2.2 Semantic Relation Extraction

Semantic relation extraction tools analyze the semantics of English sentences and extract their entailed relations. Representative tools include Ollie [19], Stanford Relation Extractor [27], LCC [17], SEMAFOR [5], and LTH [13]. Ollie is a relation extractor that extracts triples representing binary relations based on open domains. Stanford Relation Extractor and LCC, on the other hand, can only extract from a fixed set of relations. Although Ollie is flexible at extracting relations, it cannot standardize triples that represent the same semantic relation. Stanford Relation Extractor and LCC are better at relation standardization, but can work with a limited number of relations.

Compared with the aforementioned tools, SEMAFOR and LTH are FrameNet-based semantic parsers that aim to identify a large number of relations and achieve standardization. Basically, they use machine learning algorithms to train the model based on the exemplar sentences in FrameNet. Based our empirical study, SEMAFOR and LTH do not perform well enough for knowledge acquisition. Recall the sentence `Mary works for IBM as an engineer` from the previous section. SEMAFOR extracts two frames: one is `usefulness` frame triggered by `work`, where `Mary` and `for IBM` represent the `entity` and `purpose` frame elements respectively; the other one is `People_by_vocation` frame triggered by `engineer`, with no frame elements attached. The first one is wrong because `for` in this context does not express the purpose meaning. Although the second frame is correct, it does not find who holds this vocation.

In our analysis of FrameNet 1.6 data, 65592 out of 93413 valence patterns have only one exemplar sentence and 11984 valence patterns have two exemplar sentences. This accounts for 83% of the total valence patterns. However, there are also valence patterns with more than 100 exemplar sentences. An uneven distribution of the exemplar sentences per valence pattern will result in an imprecise estimation of model parameters. In addition, frame elements do not have semantic restrictions, which are useful in practical cases. For instance, comparing sentences `Mary has a full-time job` and `Mary has a well-paid job`, both `full-time` and `well-paid` are adjective modifiers of `job`. But, they are classified as two different frame elements: `Contract-basis` and `Compensation` respectively. Without any semantic constraints, we cannot distinguish these two frame elements based on their syntactical context.

2.3 Semantic Similarity

Semantic similarity measures the semantic closeness of a pair of synsets. Traditional methods include lesk [2], wup [29], lch [16], jcn [12], lin [18], res [22], hso [11]. For instance, lesk measures the semantic similarity of two synsets based on the overlapping information between their glosses. Others are based on the graph relation between two synsets in WordNet. Recent work includes the NASARI approach [3], where each word is represented as a vector. Semantic similarity is computed based on the Weighted Overlap measure [21]. In KAM, we use NASARI approach. First, NASARI dataset is based on BabelNet, which is more up-to-date than WordNet. Second, NASARI approach shows better performance than the aforementioned traditional approaches based on [3].

3 KAM Framework

KAM consists of two parts: supervised knowledge annotation and knowledge acquisition. Supervised knowledge annotation is designed to create a Prolog knowledge base that represents an augmented version of FrameNet data. Basically, the knowledge base includes the logical representations of frames, frame elements, lexical units, and valence patterns. Besides, each frame element is assigned with a list of BabelNet synsets that capture its definition. Users can also add new frames to the knowledge base. The Prolog knowledge base is used in knowledge acquisition, where we provide a tool that achieves the following:

1. run deep semantic analysis of controlled English text in order to ensure that different sentences that express the same meaning are mapped to the same logical representations.
2. perform valid logical inferences based on the facts and rules extracted from English sentences and achieve inconsistency-tolerance in the process of knowledge acquisition.
3. allow the user to enter controlled English text, disambiguate acquired knowledge, and perform question answering tasks

First, we give an brief overview of KAM's language parser, Attempto Parsing Engine (APE), which is based on ACE grammar[1]. APE translates CNL sentences into a Discourse Representation Structure (DRS)[2], which captures the semantic meaning of the sentences. A DRS uses six pre-defined predicates to represent the semantics of a word in a sentence, including `object`, `property`, `relation`, `modifier_adv`, `modifier_pp`, `has_part`, `query`, and `predicate` predicates. For instance, the sentence `A man enters a door with a card` is represented as

```
object(A,man,countable,na,eq,1)
object(B,door,countable,na,eq,1)
object(C,card,countable,na,eq,1)
predicate(D,enter,A,B)
modifier_pp(D,with,C)
```

where the `object`-predicate denotes the head word of a noun phrase, the `predicate`-predicate represents an action, and the `modifier_pp` signifies a prepositional modifier to the action.

We define the semantic relation between two predicates as a *dependency path* that connects these two predicates via a list of variables and intermediate predicates. For the above example, `man` is the subject of the `enter` action. The semantic relation is represented as

```
predicate(D,enter,A,B) -> A -> object(A,man,countable,na,eq,1)
```

There can be more than one dependency paths that connect two predicates. For the rest of this section, we will only consider the shortest dependency path.

[1] `http://attempto.ifi.uzh.ch/site/docs/syntax_report.html`
[2] `http://attempto.ifi.uzh.ch/site/pubs/papers/drs_report_66.pdf`

3.1 Supervised Knowledge Annotation

Figure 1 shows the architecture of supervised knowledge annotation. The GUI provides an environment for the user to annotate FrameNet frames and query BabelNet. Given a frame, the user is required to disambiguate each frame element name by assigning a BabelNet synset to it. For instance, in `Being_Employed` frame, `Position` is assigned with the synset `bn:00010073n` (a job in an organization) and `Employee` is assigned with the synset `bn:00030618n` (a person who is hired to perform a job). The annotated frame and frame elements are mapped into Prolog representation by LFrame Generator as

```
frame_def(Frame_Name,[
    frame_element(Frame_Element_Name, BabelNet_SID)|...])
```

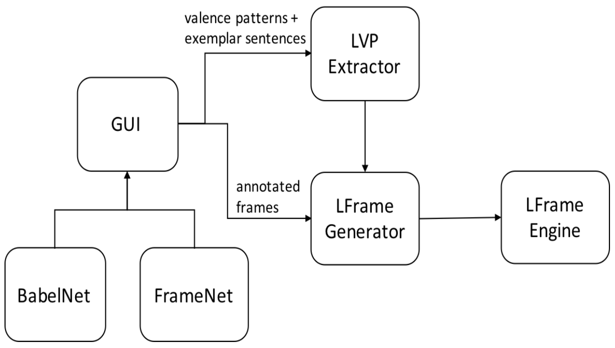


Fig. 1. Supervised Knowledge Annotation

Next, the user annotates each lexical unit and its exemplar sentences. Given that FrameNet exemplar sentences are written in normal English, some may not be parsed by APE. Therefore, the user needs to manually rephrase each exemplar sentence according to ACE grammar. Besides, the user marks the lexical unit and frame elements of a sentence. The annotated lexical units and exemplar sentences are mapped into Prolog representation by LFE Extractor as

```
lvp(Lexical_Unit, Frame_Name, [
    lgf(Frame_Element_1,Dependency_Path_1)|...])
```

where it extracts dependency paths that represent the semantic relations between a lexical unit and the frame elements.

LFrame Engine uses the `frame_def` and `lvp` predicates to extract frame relations and identify frame elements from new sentences. Specifically, LFrame Engine applies the `lvps` to each word of a sentence to extract potential frames and frame elements, denoted as

```
frame(Frame_Name,[
    frame_element(Frame_Element_Name, Val)|...])
```

3.2 Knowledge Acquisition

Figure 2 shows the process of translating a CNL sentence into its logical form. First, APE parses the input sentence and generates the DRS and part-of-speech of each word. Second, KAM queries BabelNet and gets the synsets each word belongs to. In parallel, LFrame Engine extracts the candidate frames and frame elements from the DRS. Next, for each candidate frame relation, KAM disam-

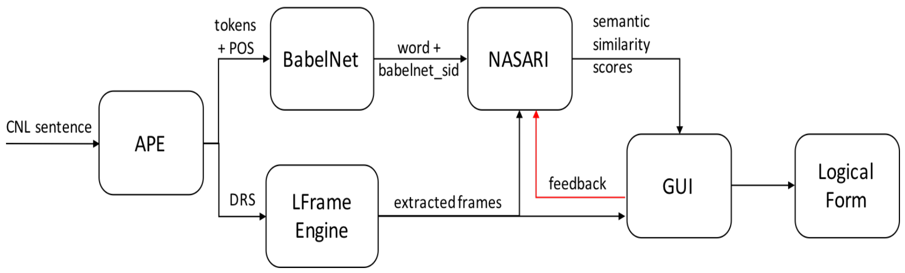


Fig. 2. Knowledge Acquisition

biguates the word sense of each frame element based on the frame element name. Recall from the previous subsection, each frame element name is assigned with a BabelNet synset ID that captures its definition. Here, KAM uses NASARI database to measure the semantic similarity between each synset the frame element belongs to and the frame element name. KAM chooses the synset with the highest semantic similarity score as the word sense of the frame element. The sum of the semantic scores of each frame element is defined as the score of the extracted frame relation. Finally, KAM ranks the candidate frames based on their scores. For example, given the sentence `There is a person who works in London`, LFrame Engine finds three candidate frame relations:

```
frame(Being_Employed,[frame_element(Employee, person),
    frame_element(Employer, London)])
frame(Being_Employed,[frame_element(Employee, person),
    frame_element(Position, London)])
frame(Being_Employed,[frame_element(Employee, person),
    frame_element(Place, London)])
```

KAM computes the semantic similarity scores between `person` and `Employee` (resp. `London` and `Employer`, `London` and `Position`, and `London` and `Place`) in order to disambiguate the word sense of `person` and `London` in each frame. In this case, the third frame has the highest score where person is assigned with BabelNet synset `bn:00046516n` (a human being) and London is assigned with `bn:00013179n` (the capital and largest city of England). KAM shows the ranked results to the user and asks the user to choose the one which is consistent with his/her understanding. Given that NASARI uses a statical approach to measure the semantic similarities, there could be errors in the computation. KAM allows the user to audit the result. The feedback will be recorded in order to improve the quality of semantic similarity measures in the next run.

KAM represents the semantics of the frame relations in a paraconsistent logical framework, Annotated Predicate Calculus (APC) [9]. The advantage APC provides over Answer Set Programming (ASP) systems and first-order logic is that APC allows inference in the presence of inconsistent knowledge. Besides, it captures a lot of complex features in natural language, e.g., negation, numerical constraints, reasoning by cases. Further details of APC and its applications in natural language understanding can be found in [9]. For the previous sentence `There is a person who works in London`, its encoding is

```
frame(being_employed, #1) : t.
frame_element(#1, employee, #2) : t.
frame_element(#1, place, #3) : t.
object(#2, person, bn:00046516n) : t.
object(#3, london, bn:00013179n) : t.
```

where `t` is a truth annotation in APC, `#1`, `#2`, and `#3` are skolemized constants, `bn:00046516n` and `bn:00013179n` refer to BabelNet synsets.

4 Current State of Research and Open Issues

Currently, we are working on building the prototype of the system that achieves knowledge annotation and knowledge acquisition. In the first stage, we focus on extracting logical facts from CNL sentences. We have encoded a subset of the frames in FrameNet that suffices to capture the frame relations in a certain domain. We also run experiments to show the power of KAM in standardizing CNL sentences to logical representations in comparison with other relation extraction tools. As the next step, we will work on extracting rules from CNL text and apply the rules and facts in question answering. Besides, we will expand the LFrame Engine to include additional frames and apply to broader domains.

5 Conclusion

In this paper, we show a novel knowledge acquisition system, KAM. First, it is a new approach in information extraction that can identify English sentences expressing the same meaning in different syntactic forms and standardize them to the same semantic representation. Second, it applies APC, a paraconsistent logical framework to encode English sentences in a logical manner to support inference in the presence of inconsistent knowledge. Third, KAM provides the users an environment to enter and disambiguate the English text and perform question answering tasks.

References

1. Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007.
2. Satanjeev Banerjee and Ted Pedersen. An adapted lesk algorithm for word sense disambiguation using wordnet. In Alexander F. Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing, Third International Conference, CICLing 2002, Mexico City, Mexico, February 17-23, 2002, Proceedings*, volume 2276 of *Lecture Notes in Computer Science*, pages 136–145. Springer, 2002.
3. José Camacho-Collados, Mohammad Taher Pilehvar, and Roberto Navigli. Nasari: Integrating explicit knowledge and corpus statistics for a multilingual representation of concepts and entities. *Artif. Intell.*, 240:36–64, 2016.
4. Danqi Chen and Christopher D. Manning. A fast and accurate dependency parser using neural networks. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 740–750. ACL, 2014.
5. Dipanjan Das, Desai Chen, André F. T. Martins, Nathan Schneider, and Noah A. Smith. Frame-semantic parsing. *Computational Linguistics*, 40(1):9–56, 2014.
6. Broes de Cat, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Predicate logic as a modelling language: The IDP system. *CoRR*, abs/1401.6312, 2014. URL: `http://arxiv.org/abs/1401.6312`.
7. Esra Erdem, Halit Erdogan, and Umut Öztok. BIOQUERY-ASP: querying biomedical ontologies using answer set programming. In Stefano Bragaglia, Carlos Viegas Damásio, Marco Montali, Alun D. Preece, Charles J. Petrie, Mark Proctor, and Umberto Straccia, editors, *Proceedings of the 5th International RuleML2011@BRF Challenge, co-located with the 5th International Rule Symposium, Fort Lauderdale, Florida, USA, November 3-5, 2011*, volume 799 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
8. Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto controlled english for knowledge representation. In Cristina Baroglio, Piero A. Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web, 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*, volume 5224 of *Lecture Notes in Computer Science*, pages 104–124. Springer, 2008.
9. Tiantian Gao, Paul Fodor, and Michael Kifer. Paraconsistency and word puzzles. *TPLP*, 16(5-6):703–720, 2016.
10. Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.
11. Graeme Hirst, David St-Onge, et al. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database*, 305:305–332, 1998.

12. Jay J. Jiang and David W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *CoRR*, cmp-lg/9709008, 1997.
13. Richard Johansson and Pierre Nugues. Lth: Semantic structure extraction using nonprojective dependency trees. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, SemEval '07, pages 227–230, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.
14. Chrstopher R. Johnson, Charles J. Fillmore, Miriam R.L. Petruck, Collin F. Baker, Michael J. Ellsworth, Josef Ruppenhofer, and Esther J. Wood. FrameNet: Theory and Practice, 2002.
15. Paul Kingsbury and Martha Palmer. Propbank: the next level of treebank. In *Proceedings of Treebanks and lexical Theories*, volume 3. Citeseer, 2003.
16. Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283, 1998.
17. John Lehmann, Sean Monahan, Luke Nezda, Arnold Jung, and Ying Shi. LCC approaches to knowledge base population at TAC 2010. In *Proceedings of the Third Text Analysis Conference, TAC 2010, Gaithersburg, Maryland, USA, November 15-16, 2010*. NIST, 2010.
18. Dekang Lin. An information-theoretic definition of similarity. In Jude W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pages 296–304. Morgan Kaufmann, 1998.
19. Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. Open language learning for information extraction. In Jun'ichi Tsujii, James Henderson, and Marius Pasca, editors, *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea*, pages 523–534. ACL, 2012.
20. George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.
21. Mohammad Taher Pilehvar, David Jurgens, and Roberto Navigli. Align, disambiguate and walk: A unified approach for measuring semantic similarity. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 1341–1351. The Association for Computer Linguistics, 2013.
22. Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *arXiv preprint cmp-lg/9511007*, 1995.
23. Konstantinos Sagonas, Terrance Swift, and David S. Warren. Xsb as an efficient deductive database engine. In *In Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 442–453. ACM Press, 1994.
24. Karin Kipper Schuler. *Verbnet: A Broad-coverage, Comprehensive Verb Lexicon.* PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 2005. AAI3179808.
25. Rolf Schwitter. English as a formal specification language. In *13th International Workshop on Database and Expert Systems Applications (DEXA 2002), 2-6 September 2002, Aix-en-Provence, France*, pages 228–232. IEEE Computer Society, 2002.
26. Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

27. Mihai Surdeanu, David McClosky, Mason R. Smith, Andrey Gusev, and Christopher D. Manning. Customizing an information extraction system to a new domain. In *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics*, RELMS '11, pages 2–10, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.
28. Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.
29. Zhibiao Wu and Martha Stone Palmer. Verb semantics and lexical selection. In James Pustejovsky, editor, *32nd Annual Meeting of the Association for Computational Linguistics, 27-30 June 1994, New Mexico State University, Las Cruces, New Mexico, USA, Proceedings.*, pages 133–138. Morgan Kaufmann Publishers / ACL, 1994.

Improved Filtering for the Bin-Packing with Cardinality Constraint

Guillaume Derval[1](main author, PhD student), Jean-Charles Régin[2] (co-author) and Pierre Schaus[1] (advisor)

[1] UCLouvain, Belgium,
guillaume.derval@uclouvain.be, pierre.schaus@uclouvain.be
[2] University of Nice Sophia-Antipolis, France,
jcregin@gmail.com

Abstract. Previous research shows that a cardinality reasoning can improve the pruning of the bin-packing constraint. We introduce a filtering on the load and cardinality bounds of the bins, using a flow reasoning similar to the Global Cardinality Constraint. Moreover, we detect impossible assignments of items by combining the load and cardinality of the bins. We experiment our new algorithms on Balanced Academic Curriculum Problem and Tank Allocation Problem instances. Our results show that the introduced filtering can sometimes drastically reduce the size of the search tree and the computation time.

Keywords: Bin-packing, cardinality, flows, constraints

Extending Compact-Table to Basic Smart Tables

Hélène Verhaeghe[1] (PhD student), Christophe Lecoutre[2] (co-author), Yves Deville[1] (advisor), and Pierre Schaus[1] (advisor)

[1]UCLouvain, ICTEAM, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium, $\{firstname.lastname\}@uclouvain.be$
[2]CRIL-CNRS UMR 8188, Université d'Artois, F-62307 Lens, France, $lecoutre@cril.fr$

Abstract. Table constraints are instrumental in modeling combinatorial problems with Constraint Programming. Recently, Compact-Table (CT) has been proposed and shown to be as an efficient filtering algorithm for table constraints, notably because of bitwise operations. CT has already been extended to handle non-ordinary tables, namely, short tables and/or negative tables. In this paper, we introduce another extension so as to deal with basic smart tables, which are tables containing universal values ($*$) as well as restrictions on values ($\neq v$) bounds ($\leq v$ or $\geq v$) and sets ($\in S$). Such tables offer the user a better expressiveness and permit to deal efficiently with compressed tuples. Our experiments show a substantial speedup when compression is possible (and a very limited overhead otherwise).

Keywords: Table Constraints, Filtering, Compression, Compact-Table, Bitset

CoverSize: A Global Constraint for Frequency-based Itemset Mining

Pierre Schaus[1]* and John O.R. Aoga[1,2]** and Tias Guns[3]***

[1]UCLouvain, ICTEAM (Belgium); [2]UAC, ED-SDI (Benin)
[3]VUB Brussels (Belgium) and KU Leuven (Belgium)
{john.aoga,pierre.schaus}@uclouvain.be; tias.guns@{vub.be,cs.kuleuven.be}

Abstract. Constraint Programming is becoming competitive for solving certain data-mining problems largely due to the development of global constraints. We introduce the CoverSize constraint for itemset mining problems, a global constraint for counting and constraining the number of transactions covered by the itemset decision variables. We show the relation of this constraint to the well-known table constraint, and our filtering algorithm internally uses the reversible sparse bitset data structure recently proposed for filtering table. Furthermore, we expose the size of the cover as a variable, which opens up new modeling perspectives compared to an existing global constraint for (closed) frequent itemset mining. For example, one can constrain minimum frequency, or compare the frequency of an itemset in different datasets as is done in discriminative itemset mining. We demonstrate experimentally on the frequent, closed and discriminative itemset mining problems that the CoverSize constraint with reversible sparse bitsets allows to outperform other CP approaches.

* Pierre Schaus is the supervisor
** John AOGA is the student
*** Tias Guns is a co-author

Weight-Aware Core Extraction in SAT-Based MaxSAT Solving[⋆]

Jeremias Berg[⋆⋆] and Matti Järvisalo[⋆⋆⋆]

HIIT, Department of Computer Science, University of Helsinki, Finland

Abstract. Several recent breakthroughs in solver technology for the constraint optimization paradigm of maximum satisfiability (MaxSAT), based on Boolean satisfiability (SAT) solvers, are making MaxSAT today a competitive approach to tackling NP-hard optimization problems in a variety of AI and industrial domains. A great majority of the modern state-of-the-art MaxSAT solvers are core-guided, relying on a SAT solver to iteratively extract unsatisfiable cores of the soft clauses in the working formula and ruling out the found cores via adding cardinality constraints into the working formula until a solution is found. In this work we propose weight-aware core extraction (WCE) as a refinement to the current common approach of core-guided solvers. WCE integrates knowledge of soft clause weights into the core extraction process, and allows for delaying the addition of cardinality constraints into the working formula. Enabled by implementing clause cloning through assumptions, we show that WCE noticeably improves in practice the performance of PMRES, one of the recent core-guided MaxSAT algorithms using soft cardinality constraints. We also explain how the approach can be integrated into other core-guided algorithms employing soft cardinality constraints, and to what extent the presented ideas can be used in other SAT-based MaxSAT approaches.

This paper was accepted at CP 2017. Except for this I have two other papers accepted:
Minimum-Width Confidence Bands via Constraint Optimization,
Berg, J., Oikarinen, E., Järvisalo, M. J. & Puolamäki, K. at CP and
MaxPre An Extended MaxSAT Preprocessor,
Korhonen, T. M. A., Berg, J., Saikko, P. H. A. & Järvisalo, M. J. at SAT.

[⋆] This work was financially supported by Academy of Finland (grants 251170 COIN, 276412, 284591); and DoCS Doctoral School in Computer Science and Research Funds of the University of Helsinki.

[⋆⋆] The Student

[⋆⋆⋆] The Advisor

Branch-and-Check with Explanations for the Vehicle Routing Problem with Time Windows

Edward Lam[1,2] and Pascal Van Hentenryck[3]

[1] CSIRO Data61, Eveleigh NSW 2015, Australia
[2] University of Melbourne, Parkville VIC 3010, Australia
[3] University of Michigan, Ann Arbor MI 48109-2117, USA

Abstract. This paper proposes the framework of branch-and-check with explanations (BCE), a branch-and-cut method where combinatorial cuts are found by general-purpose conflict analysis, rather than by specialized separation algorithms. Specifically, the method features a master problem that ignores combinatorial constraints, and a feasibility subproblem that uses propagation to check the feasibility of these constraints and performs conflict analysis to derive nogood cuts. The BCE method also leverages conflict-based branching rules and strengthens cuts in a post-processing step. Experimental results on the Vehicle Routing Problem with Time Windows show that BCE is a potential alternative to branch-and-cut. In particular, BCE dominates branch-and-cut, both in proving optimality and in finding high-quality solutions quickly.

1 Introduction

Vehicle Routing Problems (VRPs) generalize the Travelling Salesman Problem (TSP). The Capacitated Vehicle Routing Problem (CVRP) is a basic variant that aims to design routes of minimal travel distance that deliver all requests from a single depot while respecting vehicle capacity constraints. The Vehicle Routing Problem with Time Windows (VRPTW) additionally requires requests to be delivered within a given time window.

VRPs have been studied extensively over the past several decades, resulting in significant computational progress (e.g., [31]). Solution techniques include constraint programming (e.g., [28,7,8]), branch-and-bound, branch-and-cut (e.g., [21,4,18]), branch-and-price (e.g., [10]), and combinations thereof (e.g., [13,3,16,25,26]). Branch-and-cut (BC) methods are of particular interest to this paper. Their key idea is to omit difficult constraints from the original formulation and to remove solutions that violate these constraints using cuts generated by separation algorithms. Separation algorithms are typically problem-specific, which limits their applicability and reuse in other problems. Furthermore, developing and implementing separation algorithms often require significant expertise, hindering their use in many applications.

This paper addresses the following research question: *Is it possible to use a general-purpose mechanism to generate cuts, and hence, avoid the difficult aspects of BC.* This paper proposes *branch-and-check with explanations* (BCE) as one possible answer to this question. BCE divides an optimization problem into a master problem that ignores a number of difficult constraints, and a subproblem that checks the feasibility of these constraints and generates cuts using conflict analysis from constraint programming (CP) and Boolean satisfiability (SAT). More precisely, BCE uses CP for three purposes: (1) to fix variables in the master problem through propagation; (2) to generate cuts in the master problem using conflict analysis; and (3) to probe the feasibility of linear programming (LP) relaxation solutions and to derive additional cuts through conflict analysis if the probing process fails. Since the master problem does not operate on the same decision variables as the subproblem, the conflict analysis needs to continue until the variables involved in a nogood appear in the master problem.

BCE opens some interesting opportunities. First, it has the advantage of relying on a general-purpose CP engine for inference and cut separation. Second, it permits conflict-based branching rules. Finally, BCE can recognize special classes of cuts after conflict analysis and then strengthen them using well-known techniques. As a result, BCE offers a natural integration of LP, CP and SAT.

The BCE method is evaluated on the VRPTW. Experimental results indicate that BCE outperforms a BC algorithm: it proves optimality on more instances and finds significantly better solutions to instances for which BC cannot prove optimality. The results also show that a conflict-based branching rule is particularly effective in BCE and that cut strengthening produces interesting improvements to the lower bounds.

The rest of this paper is structured as follows. Section 2 reviews relevant methods for solving the VRPTW. Section 3 develops the BCE model. Section 4 discusses cut strengthening. Section 5 presents experimental results that compare the BCE model with the BC model. Section 6 discusses the limitations and potential improvements of the BCE approach for the VRPTW, as well as its relevance to branch-and-price. Section 7 concludes this paper.

2 Background

BC algorithms for VRPs are often based on a two-index flow model, which generalizes the standard formulation of the TSP. The two-index model omits the subtour elimination, vehicle capacity and time window constraints, which are added as required through cutting planes. At every node of the search tree, BC solves separation subproblems to determine if the LP relaxation solution is feasible with respect to the omitted constraints. If the solution is infeasible, the solution is discarded using a cut, forcing the solver to find another candidate solution. Branching and cutting are repeated until the search tree is explored, upon which the solver proves optimality or infeasibility.

Branch-and-Cut. BC models of the VRPTW rely on several types of cuts. The BC model in [4] inherits the capacity cuts from BC models of the CVRP. Capacity cuts generalize the subtour elimination cuts of the TSP to consider vehicle capacity. Hence, they serve the purpose of excluding both subtours and partial paths that exceed the vehicle capacity. This model also implements infeasible path cuts to exclude partial paths that violate the time windows. Infeasible path cuts require at least one arc in an infeasible partial path to be unused. The BC algorithm from [18] uses subtour elimination constraints from the TSP instead of the capacity cuts. Vehicle capacity constraints are enforced by the same infeasible path cuts that enforce the time windows. The authors also prove that both the subtour elimination cuts and the infeasible path cuts can be strengthened using ideas conceived in [22].

Branch-and-Check. Branch-and-check [29,5] is a form of logic-based Benders decomposition [15]. The method divides a problem into a master and checking problem. The master problem is first solved to find a candidate solution, which is checked using the checking subproblem. If the checking subproblem is infeasible for a candidate solution, a constraint prohibiting this solution, and hopefully many others, is added to the master problem. Branch-and-check iterates between the master problem and the checking subproblem until a globally optimal solution is found. It has been used successfully in various applications (e.g., [14,30]). The key difference between BC and branch-and-check is that checking subproblems encompass an entire optimization problem, whereas separation subproblems only check specific aspects of the problem (i.e., they find cuts from one family).

Constraint Programming. CP with large neighborhood search was instrumental in finding many best solutions to VRPs more than a decade ago [28,7,8]. The main difficulty with CP is proving optimality since VRP objective functions are usually linear, which are known to have weak propagators. This limitations can be alleviated using the WeightedCircuit global constraint [6], for example.

Conflict Analysis. Conflict analysis has a long history in artificial intelligence and CP (e.g., [9,17]). Its popularity grew in the last two decades through the development of SAT solvers (e.g., [23,11]) and their integration in CP solvers (e.g., [24,12]). In CP solvers, propagators generate clauses that explain the inferences for an underlying SAT solver. When a propagator fails, the SAT solver performs conflict analysis, i.e., it walks the implication graph to derive a constraint, known as a nogood, that prevents the same failure from reoccurring in other parts of the search tree. Conflict analysis can also be implemented in mixed integer programming (MIP) solvers but its performance is still an open question [1].

3 The Branch-and-Check Model of the VRPTW

This section proposes the BCE model of the VRPTW. The model is organized around a MIP master problem and a CP checking subproblem.

The MIP Master Problem. The BCE model includes the traditional two-index model. Its data and decision variables are listed in Table 1. The arcs in the

Name	Description
$T > 0$	Time horizon.
$\mathcal{T} = [0, T]$	Time interval.
$Q \geq 0$	Vehicle capacity.
$\mathcal{Q} = [0, Q]$	Range of vehicle load.
$R \in \{1, \dots, \infty\}$	Number of requests.
$\mathcal{R} = \{1, \dots, R\}$	Set of requests.
$s = 0$	Start node.
$e = R + 1$	End node.
$\mathcal{N} = \mathcal{R} \cup \{s, e\}$	Set of all nodes.
$\mathcal{A}$	Arcs of the network. Defined in Eq. (4).
$c_{i,j} \in \mathcal{T}$	Distance cost and travel time along arc $(i, j) \in \mathcal{A}$.
$q_i \in Q$	Vehicle load demand of $i \in \mathcal{N}$.
$a_i \in \mathcal{T}$	Earliest service start time at $i \in \mathcal{N}$.
$b_i \in \mathcal{T}$	Latest service start time at $i \in \mathcal{N}$.
$x_{i,j} \in \{0, 1\}$	Decision variable indicating if a vehicle traverses $(i, j) \in \mathcal{A}$.

Table 1. Data and decision variables of the two-index flow model of the VRPTW. Sets enclosed in braces (resp. square brackets) are integer-valued (resp. real-valued).

$$\min \sum_{(i,j)\in\mathcal{A}} c_{i,j} x_{i,j} \tag{1}$$

subject to

$$\sum_{h:(h,i)\in\mathcal{A}} x_{h,i} = 1 \qquad \forall i \in \mathcal{R}, \tag{2}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{i,j} = 1 \qquad \forall i \in \mathcal{R}. \tag{3}$$

Fig. 1. Initial constraints of the two-index model of the VRPTW.

network are given by

$$\begin{aligned} \mathcal{A} = &\{(s, i) | i \in \mathcal{R}\} \cup \\ &\{(i, j) | i, j \in \mathcal{R}, i \neq j, a_i + c_{i,j} \leq b_j, q_i + q_j \leq Q\} \cup \\ &\{(i, e) | i \in \mathcal{R}\}. \end{aligned} \tag{4}$$

The initial constraints of the model are shown in Fig. 1. The objective function, Eq. (1), minimizes the total distance cost. Constraints (2) and (3) require every request to be visited exactly once. Through its LP relaxation, the MIP master problem provides the lower bounds to the objective value and generates candidate solutions to be tested in the CP subproblem.

Name	Description
$y_{i,j} \in \{0,1\}$	Decision variable indicating if a vehicle traverses $(i,j) \in \mathcal{A}$.
$l_i \in [q_i, Q] \subseteq \mathcal{Q}$	Vehicle load after service at request $i \in \mathcal{N}$.
$t_i \in [a_i, b_i] \subseteq \mathcal{T}$	Time that a vehicle begins service at request $i \in \mathcal{N}$.

Table 2. Decision variables of the CP subproblem.

$$\bigvee_{h:(h,i)\in\mathcal{A}} y_{h,i} \qquad \forall i \in \mathcal{R}, \tag{5}$$

$$\bigvee_{j:(i,j)\in\mathcal{A}} y_{i,j} \qquad \forall i \in \mathcal{R}, \tag{6}$$

$$\neg y_{h,i} \vee \neg y_{h,j} \qquad \forall h,i,j \in \mathcal{N} : (h,i) \in \mathcal{A}, (h,j) \in \mathcal{A}, i \neq j, \tag{7}$$

$$\neg y_{h,j} \vee \neg y_{i,j} \qquad \forall h,i,j \in \mathcal{N} : (h,j) \in \mathcal{A}, (i,j) \in \mathcal{A}, h \neq i, \tag{8}$$

$$\textsc{NoSubtour}(y), \tag{9}$$

$$y_{i,j} \to l_j \geq l_i + q_j \qquad \forall (i,j) \in \mathcal{A}, \tag{10}$$

$$y_{i,j} \to t_j \geq t_i + c_{i,j} \qquad \forall (i,j) \in \mathcal{A}. \tag{11}$$

Fig. 2. Initial constraints of the CP subproblem.

The CP Checking Subproblem. The BCE model uses a CP subproblem to check the solutions found by the master problem for feasibility of the subtour elimination, vehicle capacity and time window constraints. The decision variables are listed in Table 2. Using the y binary variables, instead of the conventional successor and predecessor variables, provides a one-to-one mapping between the y variables and the x variables of the master problem. The initial constraints (without the nogoods) are presented in Fig. 2. Constraints (5) to (8) ensure that every request is visited exactly once. Constraint (9) is a global constraint that prevents subtours. Its propagator is a simple checking algorithm that prevents the head of a partial path from connecting to its tail. Constraints (10) and (11) enforce the vehicle capacity and travel time constraints.

Communication between the Two Models. The two models communicate in three ways: (1) variable assignments in the CP model are transmitted to the MIP model, (2) candidate solutions from the LP relaxation are probed using the CP model to determine if they are valid for the VRPTW and (3) nogoods found by conflict analysis in the CP model are translated into cuts in the MIP model.

Extended Conflict Analysis. When a failure occurs in the CP solver, conflict analysis derives a First Unique Implication Point (1UIP) nogood that is added to the CP subproblem. This constraint should also be added to the master problem but sometimes it cannot be translated into a cut for the master problem because it contains variables that do not appear in the master problem (i.e., the load and time variables). As a result, the BCE algorithm features an extended conflict analysis that continues explaining the failure until the the nogood only contains variables in master problem. This nogood has the form

$$\bigvee_{(i,j)\in\mathcal{C}_1} y_{i,j} \vee \bigvee_{(i,j)\in\mathcal{C}_2} \neg y_{i,j},$$

where $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{A}$ are sets of arcs. This nogood can be rewritten as the cut

$$\sum_{(i,j)\in\mathcal{C}_1} x_{i,j} + \sum_{(i,j)\in\mathcal{C}_2} (1 - x_{i,j}) \geq 1.$$

It is always possible to obtain these cuts since the solver only branches on variables in the master problem. *Observe that the BCE algorithm provides a general-purpose mechanism to separate cuts in the master problem via the extended conflict analysis. These cuts, which we call MIP-1UIP nogoods, are automatically generated and do not rely on specialized separation algorithms.*

Probing the LP Relaxation. The BCE algorithm probes whether the current LP solution is feasible with respect to the subtour elimination, vehicle capacity and time constraints. It temporarily assigns every $y_{i,j}$ variable to the value of its corresponding $x_{i,j}$ variable in the LP relaxation, provided that this value is integral. The resulting tentative assignment can then be propagated by the CP solver. If a failure occurs, conflict analysis generates nogoods for both the CP and MIP models. The MIP cut will exclude the current LP solution, forcing it to find another candidate solution and improving the lower bound.

The Search Algorithm. The BCE algorithm, detailed in Fig. 3, includes the components described earlier. It blends depth-first and best-first search since best-first search is more effective for hard optimization problems, such as VRPs, but complicates the implementation of CP solvers with conflict analysis. The node selection strategy selects the node with the lowest lower bound from the set of open nodes and then explores the node subtree using depth-first search until it reaches a limit on the maximum number of open nodes per subtree. Once it reaches this limit, all unsolved siblings in the subtree are moved into the set of open nodes, and then the algorithm starts a new depth-first search from the node with the next lowest lower bound. Section 6 explains the rationale behind this search procedure.

Once a node is selected (step 1), the CP subproblem infers the implications of the decision (step 2). In the case of failure, the CP solver generates nogoods for both models and then backtracks (step 5b). If the test succeeds, the BCE algorithm checks for suboptimality using the LP relaxation (step 3). If the node is suboptimal, it backtracks (step 5b). Otherwise, the BCE algorithm checks the LP relaxation solution against the omitted constraints and separates cuts using conflict analysis if necessary (step 4). The BCE algorithm iterates between the LP relaxation and the feasibility test until no cuts are generated. Then, if the

1. **Node Selection:** Select an open node. Terminate if no open nodes remain.
2. **Feasibility Check:** Solve the CP model to determine the implications of the branching decision of the node. If propagation fails, perform conflict analysis, add the 1UIP and the MIP-1UIP nogoods to both the CP and MIP models, and go to step 5b. Otherwise, fix $x_{i,j}$ in the MIP model to the values of the $y_{i,j}$ variables.
3. **Suboptimality Check:** Solve the LP relaxation. If the objective value is worse than the incumbent solution, go to step 5b.
4. **LP Probing:** For all $x_{i,j}$ variables with a value of 0 or 1 in the LP relaxation, temporarily fix the $y_{i,j}$ variables in the CP model to the same value. Propagate the CP model. If it fails, perform conflict analysis, generate the 1UIP and the MIP-1UIP nogoods and go back to step 3.
5. **Branching and Backtracking:** If all $x_{i,j}$ variables are integral, store the LP relaxation solution as the incumbent solution and go to step 5b. Otherwise, go to step 5a because the node is fractional.
   (a) **Branching:** Create two children nodes from a fractional $x_{i,j}$ variable. Fix the variable to 0 in one child node and to 1 in the other.
   (b) **Backtracking:** If the number of nodes in the current subtree exceeds the limit or if the subtree is entirely solved, move all unsolved siblings in the subtree to the set of open nodes and go back to step 1. Otherwise, backtrack to an ancestor with an unsolved child node, select the child node and go to step 2.

Fig. 3. The BCE Search Algorithm.

node is fractional and not suboptimal, the BCE algorithm executes a branching step (step 5a). Two branching rules are implemented. The first selects the most fractional variable and the second selects the variable with the highest activity, which is defined as the number of nogoods in which the variable has previously appeared. This branching rule, known as activity-based search or variable state independent decaying sum (VSIDS) in the literature, guides the search tree towards subtrees that can be quickly pruned due to infeasibility.

Illustrating the Extended Conflict Analysis. The following discussion illustrates the extended conflict analysis procedure using the example in Figs. 4 and 5. Literals shown in a grey are fixed by the data at the root level, and hence, are always true. They are discarded in the explanations but are shown for clarity.

The BCE solver first branches on $\neg y_{4,6}$, making it true. The travel time constraint (Constraint (11)) propagates $[\![t_6 \geq 30]\!]$ with the reason

$$\neg y_{4,6} \wedge [\![t_3 \geq 25]\!] \wedge [\![c_{3,6} = 10]\!] \wedge [\![t_5 \geq 20]\!] \wedge [\![c_{5,6} = 10]\!] \rightarrow [\![t_6 \geq 30]\!]$$

because the predecessor of request 6 must be either 3 or 5, and the earliest time to reach 6 is at time $\min(\min(t_3) + c_{3,6}, \min(t_5) + c_{5,6}) = 30$. The BCE solver then branches on $\neg y_{3,6}$. Constraint (5) requires every request to have a predecessor, which leads to the assignment of $y_{5,6}$ with the reason

$$\neg y_{3,6} \wedge \neg y_{4,6} \rightarrow y_{5,6}.$$

Constraint (8) then propagates

$$y_{5,6} \rightarrow \neg y_{5,2}$$

and

$$y_{5,6} \rightarrow \neg y_{5,7}.$$

The BCE solver then branches on $y_{0,1}$, which does not produce any inference, and then branches on $y_{6,2}$, which produces the inferences:

$$\begin{gathered} y_{6,2} \rightarrow \neg y_{6,7}, \\ \neg y_{6,7} \wedge \neg y_{5,7} \rightarrow y_{2,7}, \\ y_{6,2} \wedge [\![t_6 \geq 30]\!] \wedge [\![c_{6,2} = 10]\!] \rightarrow [\![t_2 \geq 40]\!]. \end{gathered}$$

Then, the travel time propagator fails with

$$y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge [\![c_{2,7} = 10]\!] \wedge [\![t_7 \leq 40]\!] \rightarrow \text{false}.$$

Conflict analysis deduces the following:

$$\begin{gathered} y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge [\![c_{2,7} = 10]\!] \wedge [\![t_7 \leq 40]\!] \rightarrow \text{false} \\ y_{2,7} \wedge [\![t_2 \geq 40]\!] \wedge \text{true} \wedge \text{true} \rightarrow \text{false} \\ (\neg y_{6,7} \wedge \neg y_{5,7}) \wedge (y_{6,2} \wedge [\![t_6 \geq 30]\!] \wedge [\![c_{6,2} = 10]\!]) \rightarrow \text{false} \\ y_{6,2} \wedge \neg y_{5,7} \wedge [\![t_6 \geq 30]\!] \wedge \text{true} \rightarrow \text{false} \\ y_{6,2} \wedge \neg y_{5,7} \wedge [\![t_6 \geq 30]\!] \rightarrow \text{false}. \qquad (12) \end{gathered}$$

This explanation contains exactly one literal $(y_{6,2})$ at the current depth, and hence, is rewritten as the 1UIP clause

$$\neg y_{6,2} \vee y_{5,7} \vee [\![t_6 < 30]\!],$$

which is added to the CP model. Conflict analysis must continue because the nogood contains a time literal. It explains $[\![t_6 \geq 30]\!]$ in Eq. (12), which results in the MIP-1UIP explanation

$$y_{6,2} \wedge \neg y_{5,7} \wedge \neg y_{4,6} \rightarrow \text{false}.$$

This explanation is rewritten into the disjunction

$$\neg y_{6,2} \vee y_{5,7} \vee y_{4,6}, \qquad (13)$$

and then into the cut

$$(1 - x_{6,2}) + x_{5,7} + x_{4,6} \geq 1.$$

Note that the literal $y_{5,7}$ was not assigned by the search.

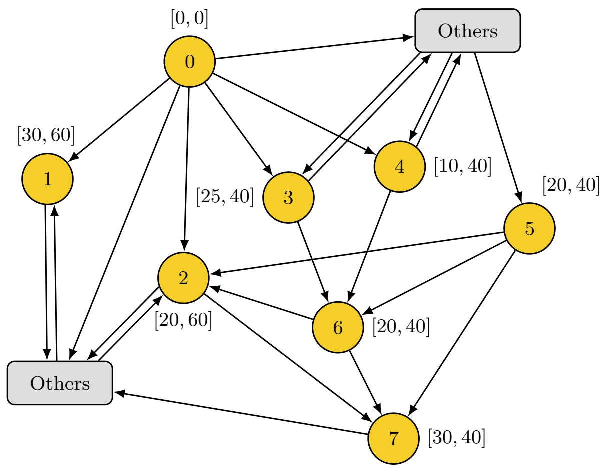


Fig. 4. Example of a network. Next to every request is its time window. The travel time across any arc is 10 units of time. The load demands are not shown as they are not relevant to the discussion.

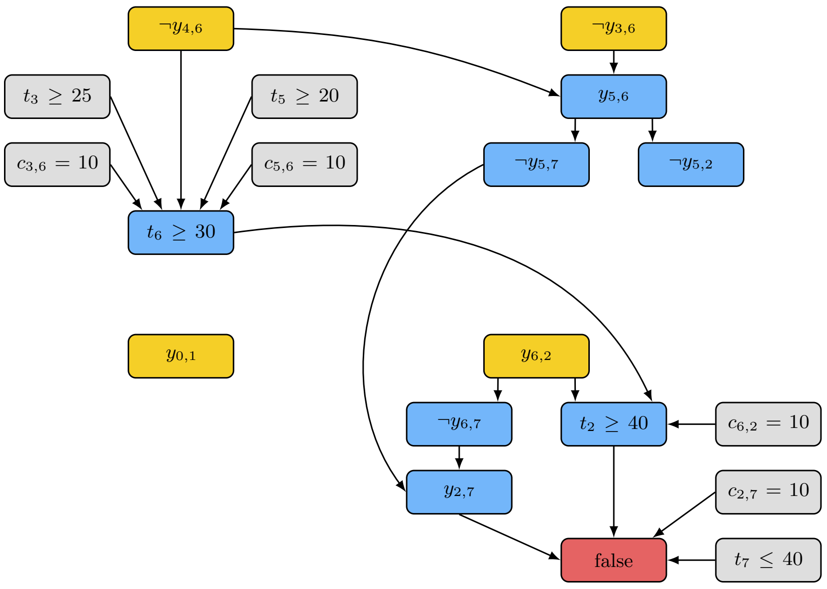


Fig. 5. Example of an implication graph after making the decisions $\neg y_{4,6}$, $\neg y_{3,6}$, $y_{0,1}$ and $y_{6,2}$ on the network in Fig. 4. Yellow literals are branching decisions. Blue literals are propagations. Grey literals are propagated at the root level, and hence, can be excluded from the nogoods since they are always true.

4 Nogood Strengthening

The BCE algorithm presented so far uses a completely general-purpose mechanism for cut separation. Despite its generality, conflict analysis routinely discovers classical cuts. These cuts can be strengthened using proven techniques whenever they are recognized. This section presents a post-processing step that recognizes then strengthens several types of cuts.

Infeasible Path Cuts. Failure of the load or time constraints (Constraints (10) and (11)) frequently results in an infeasible partial path cut. Let $P = i_1, i_2, \ldots, i_k$, with all $i_1, \ldots, i_k \in \mathcal{N}$ distinct, be a partial path. The partial path P is infeasible with respect to the load constraint if $\sum_{u=1}^{k} q_{i_u} > Q$, and it is is infeasible with respect to the time constraint if $t_{i_k} > b_{i_k}$, where $t_{i_1} = a_{i_1}$ and $t_{i_u} = \max(a_{i_u}, t_{i_{u-1}} + c_{i_{u-1},i_u})$ for $u = 2, \ldots, k$. When a load or time window constraint fails, conflict analysis will usually produce the nogood

$$\bigvee_{(i,j)\in A(P)} \neg y_{i,j}, \tag{14}$$

where $A(P) = \{(i_1, i_2), \ldots, (i_{k-1}, i_k)\}$ is the arcs of P. This nogood requires one arc of P to be unused. It can be written equivalently as requiring at least one arc that exits P, i.e.,

$$\bigvee_{(i,j)\in\Delta^+(P)} y_{i,j},$$

where $\Delta^+(P) = \bigcup_{u=1}^{k-1} \{(i_u, j) \in \mathcal{A} | j \neq i_{u+1}\}$. This nogood can be translated into the cut

$$\sum_{(i,j)\in\Delta^+(P)} x_{i,j} \geq 1.$$

Using existing techniques [18], such a cut can be strengthened into

$$\sum_{(i,j)\in\widetilde{\Delta}^+(P)} x_{i,j} \geq 1,$$

where

$$\widetilde{\Delta}^+(P) = \bigcup_{u=1}^{k-1} \Bigg(\Bigg\{ (i_u, j) \in \mathcal{A} : i_u \in \mathcal{R}, j \in \mathcal{R}, j \neq i_1, \ldots, i_{u+1}, \\ \sum_{v=1}^{u} q_{i_v} + q_j \leq Q, t_{i_u} + c_{i_u,j} \leq b_j \Bigg\} \cup \{(i_u, e) \in \mathcal{A}\} \Bigg)$$

is the arcs that branch off P to a feasible request. In other words, the strengthening discards arcs that are not feasible when taking into account the load and time window constraints.

Subtour Elimination Cuts. The propagator of Constraint (9) will fail if the solution contains a subtour $S = i_1, i_2, \dots, i_k$, where $i_1 = i_k$ and all $i_1, i_2, \dots, i_{k-1} \in \mathcal{R}$ are distinct. Conflict analysis will usually find the nogood

$$\bigvee_{(i,j)\in A(S)} \neg y_{i,j}, \tag{15}$$

where $A(S) = \{(i_1, i_2), \dots, (i_{k-1}, i_k)\}$ is the arcs of S. Using the same reasoning as for the infeasible path cuts, this nogood can be rewritten as the cut

$$\sum_{(i,j)\in\Delta^+(S)} x_{i,j} \geq 1. \tag{16}$$

If $a_j + c_{j,i} > b_i$, then no vehicle can depart j for i while respecting the time windows. Hence, i must precede j with respect to time, written as $i \prec j$. Let $\pi(j) = \{i \in \mathcal{N} | i \prec j\}$ be the set of requests that precedes j with respect to time. Proposition 1 strengthens Constraint (16) using these precedence relations [18]. Constraint (16) can also be similarly strengthened using the precedence relations in reverse, i.e., successor relations.

Proposition 1. *Let $\bar{S} = \mathcal{N} \setminus S$ be the nodes not in a subtour S, then for any $u \in S$, Constraint (16) can be strengthened to*

$$\sum_{\substack{(i,j)\in\mathcal{A}: \\ i\in S\setminus\pi(u), \\ j\in\bar{S}\setminus\pi(u)}} x_{i,j} \geq 1.$$

Proof. Consider a subtour S and a feasible path F that visits the request u. Let $v \in \mathcal{R}$ be the last request of F visited by S. By definition, v is visited by S, i.e., $v \in S$. Furthermore, since F is a feasible path, v cannot precede u with respect to time, i.e., $v \notin \pi(u)$. Hence, $v \in S \setminus \pi(u)$. Now consider the successor of v, denoted by $\mathrm{succ}(v) \in \mathcal{N}$. By the definition of v, $\mathrm{succ}(v)$ cannot be visited by S, i.e., $\mathrm{succ}(v) \notin S$. Again, $\mathrm{succ}(v)$ cannot precede u with respect to time since F is a feasible path. Hence, $\mathrm{succ}(v) \in \bar{S} \setminus \pi(u)$. Considering every request in S as v results in the proposition.

General Cuts. Conflict analysis can derive cuts that are do not have the form of Constraint (14) nor Constraint (15). These cuts contain both true literals and false literals, such as those of Constraint (13). They originate from fixing an arc to be unused (i.e., setting $x_{i,j} = 0$ for some $(i,j) \in \mathcal{A}$), which can result in tightening the bounds of a time or load variable. Consequently, an assigned arc can become infeasible. Hence, the originating nogood will contain both true and false literals. We are not aware of VRP cuts in the literture that mix true literals and false literals. This is possibly because tightening bounds is too costly for every call to a separation algorithm. CP maintains the bounds internally as part of propagation, and hence, the bounds are readily available. Because of this, these cuts seem to be fundamentally linked to CP. It is an open research issue to understand whether these cuts can be strengthened.

5 Experimental Results

The Solvers. The BCE solver includes a small CP solver and calls Gurobi 6.5.2 to solve the LP relaxations. The algorithm presented in Fig. 3 has a limit of 500 nodes for the depth-first search. This number was chosen experimentally as it was superior to limits of 100, 1,000, 5,000, and 10,000 nodes. The experiments consider four versions of the solver: with and without cut strengthening, and with the two branching rules. The four versions are compared against published results of a BC model [18], as well as a pure CP model and a pure MIP model. The CP model is the standard VRPTW model based on successor variables (e.g., [19,27]), and is solved using Chuffed. The MIP model is the three-index flow model (e.g., [31]), and is solved using Gurobi. The reported results for the BC model are given an hour of CPU time on a Pentium III CPU at 600 MHz. To be fair, our solvers are run for 10 minutes on a Xeon E5-2660 V3 at 2.6 GHz.

The Results. The solvers are tested on the Solomon benchmarks with 100 requests. The results are reported in Table 3. The pure CP model failed to find any feasible solution and is omitted from the table. The pure MIP model proves optimality on only one instance and finds poor solutions to three other instances. These results were expected and are given to confirm the need for the other approaches. The rest of this section compares the BC and BCE approaches.

Upper Bounds. The four BCE methods find the same or better solutions than the BC algorithm for all instances except C204. Of the best solutions found, all but two (R201, C204) can be found using the activity-based branching rule. For the C instances, BC and BCE with activity-based search and cut strengthening are comparable since they both dominate on seven of the eight instances. For the R and RC instances, BCE with activity-based search improves upon the BC method, which generally finds solutions with costs about five times higher.

Lower Bounds. First observe that BCE with activity-based search and cut strengthening proves optimality on one more instance (RC201) than BC, which is quite remarkable. The bounds found by the BC model are superior to those from all BCE methods except for instance RC201, on which BCE with activity-based search and cut strengthening finds a tighter bound. This is not surprising since the BC algorithm implements families of cuts not present in the BCE model. These families of cuts capture logic that the constraints in the checking subproblem do not. As will be mentioned in Section 6, stronger dual bounds should be available once the BCE model is expanded with optimization constraints.

The Impact of Branching Rules. Activity-based branching performs significantly better than most-fractional branching. Without cut strengthening, activity-based branching finds solutions better than most-fractional branching on all instances except C201, on which all four BCE methods prove optimality. With cut strengthening, activity-based search performs better on 19 of the 27 instances, and worse on only one instance. This is not surprising given that branching on the most fractional variable is known to perform worse than random selection [2].

The Impact of Cut Strengthening. Cut strengthening improves the lower bounds for both branching rules. For the C instances, cut strengthening is critical for proving optimality. For the RC instances except RC208, BCE with activity-based branching and cut strengthening finds solutions better than the other methods. Cut strengthening interferes with the activity-based branching rule for about half of the R instances. The cause of this interference is not yet understood.

The results indicate that BCE is an interesting avenue for solving hard VRPs. The BCE model finds superior primal solutions despite its simplicity and the fact that it is missing many families of cuts and that the checking subproblem does not reason about optimality nor variables with fractional values in the LP relaxation solution. For practitioners without the expertise in BC, BCE provides an interesting and practically appealing alternative.

6 Future Research Directions

The BCE algorithm, presented in this paper as a proof-of-concept, can be improved in many ways. This section explores some potential improvements.

Branching. The branching rules simply assign a fractional variable to 0 in one child and 1 in the other. These branching rules make the search tree highly unbalanced, considerably degrading the performance of the solver. Future implementations should test branching on cutsets, which is the standard branching rule seen in BC models of VRPs. It would also be interesting to test branching on variables in the CP model (e.g., branching on time windows) by propagating these decisions and enforcing the implications in the MIP model.

Search Strategy. VRPs greatly benefit from best-first search. For simplicity, the BCE implementation uses depth-first search, which allows literals to be stored in a stack data structure. It is obviously possible to implement conflict analysis in best-first search but efficient implementations remain an open question today. As explained in Section 3, the BCE implementation blends depth-first search with periodic best-first selection to explore attractive parts of the search tree.

Subtour Elimination. The propagator of Constraint (9) is extremely simple and only eliminates assignments that would create a cycle. This contrasts with separation algorithms, which are able to separate cuts using fractional solutions. It would be highly desirable to study the impact of more advanced propagators and explanations for subtour elimination in CP.

Cut Strengthening. The CP model contains all the omitted constraints; namely, the subtour elimination, vehicle capacity and time window constraints. As a result, conflict analysis can deduce nogoods based on the combined infeasibility of multiple constraints. In contrast, separation algorithms only reason about one family of cuts. It is an open question whether conflict analysis can automatically strengthen the cuts by reasoning about a conjunction of constraints. This will reduce the need to develop dedicated cut strengthenings.

	Branch-and-Check – Most-Fractional						Branch-and-Check – Activity-based											
	No Strengthening			With Strengthening			No Strengthening			With Strengthening			Branch-and-Cut			MIP		
Instance	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time	LB	UB	Time
R201	1055.8	1198.0	-	1117.7	**1143.3**	-	1054.7	1177.6	-	1114.3	1149.9	-	1132.7	1155.6	-	975.6	-	-
R202	762.8	1213.0	-	852.1	1219.6	-	763.2	1133.4	-	850.7	**1109.3**	-	888.6	4980.0	-	715.3	-	-
R203	660.1	1244.8	-	709.8	1253.6	-	659.9	**1025.2**	-	707.7	1052.2	-	748.1	4980.0	-	620.3	-	-
R204	625.3	1166.7	-	639.2	1193.3	-	625.8	**858.4**	-	638.3	887.4	-	661.9	4980.0	-	584.9	-	-
R205	796.3	1222.0	-	889.6	1069.9	-	794.3	1091.3	-	876.9	**1052.5**	-	900.0	4980.0	-	732.3	-	-
R206	686.3	1171.6	-	751.4	1157.4	-	686.0	1040.1	-	745.3	**1018.9**	-	783.6	4980.0	-	644.8	-	-
R207	648.1	1187.5	-	681.5	1168.5	-	647.5	**940.7**	-	685.8	941.4	-	714.8	4980.0	-	603.1	-	-
R208	623.2	1097.4	-	633.5	1187.7	-	623.4	855.0	-	635.3	**832.5**	-	651.8	4980.0	-	577.2	-	-
R209	687.5	1238.1	-	756.6	1172.5	-	686.5	**1046.6**	-	753.1	1073.8	-	785.8	4980.0	-	648.2	-	-
R210	679.7	1225.6	-	749.9	1240.3	-	679.8	1105.6	-	750.8	**1024.9**	-	798.3	4980.0	-	636.6	-	-
R211	621.2	1335.5	-	633.0	1355.9	-	621.2	**1004.2**	-	632.1	1065.1	-	645.1	4980.0	-	577.2	4224.9	-
C201	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	0.0	589.1	**589.1**	11.5	589.1	**589.1**	15.2
C202	548.7	679.8	-	589.1	**589.1**	131.2	548.2	629.9	-	589.1	**589.1**	12.6	589.1	**589.1**	202.9	524.3	-	-
C203	526.5	948.3	-	563.4	672.2	-	524.7	686.5	-	565.9	**601.2**	-	586.0	632.3	-	507.3	-	-
C204	516.3	946.7	-	552.9	1086.7	-	514.7	884.5	-	555.9	660.9	-	584.4	**597.1**	-	488.3	-	-
C205	546.9	685.8	-	586.4	**586.4**	0.2	546.5	613.1	-	586.4	**586.4**	16.3	586.4	**586.4**	334.4	511.4	-	-
C206	539.9	776.9	-	586.0	**586.0**	11.8	538.2	702.6	-	586.0	**586.0**	10.6	586.0	**586.0**	419.0	504.7	4997.5	-
C207	542.7	851.1	-	585.8	**585.8**	20.6	538.3	635.2	-	585.8	**585.8**	8.3	585.8	**585.8**	527.5	503.9	-	-
C208	534.5	857.2	-	585.8	**585.8**	60.0	533.1	652.4	-	585.8	**585.8**	11.2	585.8	**585.8**	569.7	500.3	-	-
RC201	1086.5	1403.6	-	1245.8	**1261.8**	-	1081.0	1338.3	-	1261.8	**1261.8**	44.5	1250.1	1288.2	-	938.3	-	-
RC202	704.5	1465.9	-	912.7	1418.6	-	699.2	1204.2	-	916.9	**1152.3**	-	940.1	6609.4	-	641.0	-	-
RC203	615.0	1402.7	-	750.2	1359.6	-	610.8	1149.0	-	748.8	**1117.6**	-	781.6	6609.4	-	563.1	-	-
RC204	583.9	1410.2	-	657.8	1352.4	-	581.0	1007.1	-	657.0	**923.5**	-	692.7	6609.4	-	532.4	-	-
RC205	822.5	1511.6	-	1075.5	1307.0	-	818.8	1249.8	-	1055.8	**1240.9**	-	1081.7	6609.4	-	746.3	-	-
RC206	785.4	1485.4	-	964.3	1273.9	-	784.9	1270.2	-	950.7	**1202.8**	-	974.8	6609.4	-	698.2	-	-
RC207	647.3	1486.3	-	794.6	1424.5	-	642.9	1193.5	-	800.9	**1172.1**	-	832.4	6609.4	-	594.9	-	-
RC208	572.7	1629.8	-	624.0	1776.1	-	573.9	**1039.5**	-	624.3	1078.4	-	647.7	6609.4	-	527.1	5299.5	-

Table 3. Solutions to the Solomon instances with 100 requests. The table reports the lower bound, upper bound and time to prove optimality for each of the solvers. The best upper bound for each instance is shown in bold. The CP model is omitted as it is unable to find feasible solutions to any instance.

Optimization Constraints. The objective function has been omitted from the checking subproblem because propagators for linear function are known to be weak. Sophisticated propagators for the WeightedCircuit constraint should be implemented, as they may produce considerably stronger nogoods.

Application to Branch-and-Price. Branch-and-cut-and-price, which includes column generation and cut generation, is the current state-of-the-art exact method for solving classical VRPs. Preliminary experiments with an existing branch-and-price solver show that BCE is not beneficial with column generation for the VRPTW as nogoods will not be generated in step 4 of Fig. 3 because the paths already respect the time and capacity constraints. However, step 2 can fail due to incompatibility between the branching decisions of a node. This infeasibility cannot be detected by the pricing problem because it has no knowledge of the global problem, nor detected by the master problem until all paths are generated because artificial variables satisfy the constraints in the interim. Incompatible branching decisions can induce nogoods but this seldom occurs in branch-and-price because its LP relaxation bound is asymptotically tight, allowing it to discard nodes due to suboptimality much earlier than infeasibility. Hence, branch-and-price-and-check is unlikely to prove useful in solving classical VRPs. It is, however, useful for rich VRPs with inter-route constraints (e.g., [20]) because the pricing subproblem, being a shortest path problem, has no knowledge of the interactions between routes in the parent problem.

7 Conclusion

This paper proposed the framework of branch-and-check with explanations (BCE) as a step towards the grand unification of linear programming, constraint programming and Boolean satisfiability. BCE finds cuts using general-purpose conflict analysis instead of specialized separation algorithms. The method features a master problem, which ignores a number of constraints, and a checking subproblem, which uses inference to check the feasibility of the omitted constraints and conflict analysis to derive nogood cuts. It also leverages conflict-based branching rules and can strengthen cuts using traditional insights from branch-and-cut in a post-processing step.

Experimental results on the Vehicle Routing Problem with Time Windows show that BCE is a viable alternative to branch-and-cut. In particular, BCE dominates branch-and-cut, both in proving optimality (with cut strengthening) and in finding high-quality solutions.

BCE offers an interesting alternative to existing branch-and-cut approaches. By using a general-purpose constraint programming solver to derive cuts, BCE can greatly simplify the modelling of problems that traditionally use branch-and-cut. This, in turn, avoids the need for dedicated separation algorithms. BCE is also capable of identifying well-known classes of cuts and strengthening them in a post-processing step. Finally, BCE significantly benefits from conflict-based branching rules, opening further opportunities typically not available in branch-and-cut.

References

1. Achterberg, T.: Conflict analysis in mixed integer programming. Discrete Optimization 4(1), 4 – 20 (2007)
2. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Operations Research Letters 33(1), 42 – 54 (2005)
3. Baldacci, R., Mingozzi, A., Roberti, R.: New route relaxation and pricing strategies for the vehicle routing problem. Operations Research 59(5), 1269–1283 (2011)
4. Bard, J.F., Kontoravdis, G., Yu, G.: A branch-and-cut procedure for the vehicle routing problem with time windows. Transportation Science 36(2), 250–269 (2002)
5. Beck, J.C.: Checking-up on branch-and-check. In: Cohen, D. (ed.) Principles and Practice of Constraint Programming – CP 2010, Lecture Notes in Computer Science, vol. 6308, pp. 84–98. Springer Berlin Heidelberg (2010)
6. Benchimol, P., Hoeve, W.J., Régin, J.C., Rousseau, L.M., Rueher, M.: Improved filtering for weighted circuit constraints. Constraints 17(3), 205–233 (2012)
7. Bent, R., Van Hentenryck, P.: A two-stage hybrid local search for the vehicle routing problem with time windows. Transportation Science 38(4), 515–530 (2004)
8. Bent, R., Van Hentenryck, P.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research 33(4), 875–893 (2006)
9. Dechter, R.: Learning while searching in constraint-satisfaction-problems. In: Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science. pp. 178–185 (1986)
10. Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Operations Research 40(2), 342–354 (1992)
11. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003, Santa Margherita Ligure, Italy, May 5-8, 2003, Selected Revised Papers, pp. 502–518. Springer Berlin Heidelberg (2004)
12. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent, I.P. (ed.) Principles and Practice of Constraint Programming - CP 2009: 15th International Conference, CP 2009 Lisbon, Portugal, September 20-24, 2009 Proceedings. pp. 352–366. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
13. Fukasawa, R., Longo, H., Lysgaard, J., De Aragão, M.P., Reis, M., Uchoa, E., Werneck, R.F.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming 106(3), 491 – 511 (2006)
14. Gendron, B., Scutellà, M.G., Garroppo, R.G., Nencioni, G., Tavanti, L.: A branch-and-benders-cut method for nonlinear power design in green wireless local area networks. European Journal of Operational Research 255(1), 151–162 (2016)
15. Hooker, J.: Logic-based methods for optimization. In: Borning, A. (ed.) Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 874, pp. 336–349. Springer Berlin Heidelberg (1994)
16. Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D.: Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56(2), 497 – 511 (2008)
17. Jussien, N., Barichard, V.: The palm system: explanation-based constraint programming. In: Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000,. pp. 118–133 (2000)

18. Kallehauge, B., Boland, N., Madsen, O.B.G.: Path inequalities for the vehicle routing problem with time windows. Networks 49(4), 273–293 (2007)
19. Kilby, P., Prosser, P., Shaw, P.: A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. Constraints 5(4), 389–414 (2000)
20. Lam, E., Van Hentenryck, P.: A branch-and-price-and-check model for the vehicle routing problem with location congestion. Constraints 21(3), 394–412 (2016)
21. Lysgaard, J., Letchford, A.N., Eglese, R.W.: A new branch-and-cut algorithm for the capacitated vehicle routing problem. Mathematical Programming 100(2), 423–445 – 0025–5610 (2004)
22. Mak, V.: On the Asymmetric Travelling Salesman Problem with Replenishment Arcs. Ph.D. thesis, University of Melbourne (2001)
23. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th annual Design Automation Conference. pp. 530–535. ACM (2001)
24. Ohrimenko, O., Stuckey, P.J., Codish, M.: Propagation via lazy clause generation. Constraints 14(3), 357–391 (2009)
25. Pecin, D., Pessoa, A., Poggi, M., Uchoa, E.: Improved Branch-Cut-and-Price for Capacitated Vehicle Routing, pp. 393–403. Springer International Publishing (2014)
26. Ropke, S., Cordeau, J.F.: Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science 43(3), 267–286 (2009)
27. Rousseau, L.M., Gendreau, M., Pesant, G.: Using constraint-based operators to solve the vehicle routing problem with time windows. Journal of Heuristics 8(1), 43–58 (2002)
28. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M., Puget, J.F. (eds.) Principles and Practice of Constraint Programming — CP98, Lecture Notes in Computer Science, vol. 1520, pp. 417–431. Springer Berlin Heidelberg (1998)
29. Thorsteinsson, E.: Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: Walsh, T. (ed.) Principles and Practice of Constraint Programming — CP 2001, Lecture Notes in Computer Science, vol. 2239, pp. 16–30. Springer Berlin Heidelberg (2001)
30. Tran, T.T., Araujo, A., Beck, J.C.: Decomposition methods for the parallel machine scheduling problem with setups. INFORMS Journal on Computing 28(1), 83–95 (2016)
31. Vigo, D., Toth, P.: Vehicle Routing: Problems, Methods, and Applications, Second Edition. Society for Industrial and Applied Mathematics (2014)

SAT-Encodings for Special Treewidth and Pathwidth

Neha Lodha, Sebastian Ordyniak, and Stefan Szeider

Algorithms and Complexity Group, TU Wien, Vienna, Austria
[neha,ordyniak,sz]@ac.tuwien.ac.at

Abstract. Decomposition width parameters such as treewidth provide a measurement on the complexity of a graph. Finding a decomposition of smallest width is itself NP-hard but lends itself to a SAT-based solution. Previous work on treewidth, branchwidth and clique-width indicates that identifying a suitable characterization of the considered decomposition method is key for a practically feasible SAT-encoding.
In this paper we study SAT-encodings for the decomposition width parameters special treewidth and pathwidth. In both cases we develop SAT-encodings based on two different characterizations. In particular, we develop a new characterization for special treewidth based on elimination orderings. We compare the SAT-encodings based on the considered characterizations empirically.

Author list

- Neha Lodha - Student
- Sebastian Ordyniak - Co -author
- Stefan Szeider - Advisor

SAT-Based Local Improvement for Finding Tree Decompositions of Small Width

Johannes Fichte, Neha Lodha, and Stefan Szeider

TU Wien, Vienna, Austria

Abstract. Many hard problems can be solved efficiently if the problem instance can be decomposed by means of a tree decomposition of small width. In particular for problems beyond NP (such as #P-complete counting problems) tree decomposition-based methods are widely used. However, finding an optimal tree decomposition is itself an NP-hard problem. Existing methods for finding tree decompositions of small width either (a) yield optimal tree decompositions but are applicable only to small instances (e.g., via SAT-encodings) or (b) are based on greedy heuristics which often yield tree decompositions that are far from optimal. In this paper, we propose a new method that combines (a) and (b), where a heuristically obtained tree decomposition is improved locally by means of a SAT encoding. We provide an experimental evaluation of our new method.

Author list

- Johannes Fichte - Co-author
- Neha Lodha - Student
- Stefan Szeider - Advisor

Introducing Pareto Minimal Correction Subsets

Miguel Terra-Neves (student), Inês Lynce (advisor), and Vasco Manquinho (advisor)

INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
`{neves,ines,vmm}@sat.inesc-id.pt`

Abstract. A Minimal Correction Subset (MCS) of an unsatisfiable constraint set is a minimal subset of constraints that, if removed, makes the constraint set satisfiable. MCSs enjoy a wide range of applications, one of them being approximate solutions to constrained optimization problems. However, existing work on applying MCS enumeration to optimization problems focuses on the single-objective case.

In this work, a first definition of Pareto Minimal Correction Subsets (Pareto-MCSs) is proposed with the goal of approximating the Pareto-optimal solution set of multi-objective constrained optimization problems. We formalize and prove an equivalence relationship between Pareto-optimal solutions and Pareto-MCSs. Moreover, Pareto-MCSs and MCSs can be connected in such a way that existing state-of-the-art MCS enumeration algorithms can be used to enumerate Pareto-MCSs.

An experimental evaluation considers the multi-objective virtual machine consolidation problem. Results show that the proposed Pareto-MCS approach outperforms the state-of-the-art approaches.

Preference Elicitation for DCOPs

Atena M. Tabakhi[1], Tiep Le[2], Ferdinando Fioretto[3], and William Yeoh[1]

[1] Department of Computer Science and Engineering, Washington University in St. Louis
{amtabakhi,wyeoh}@wustl.edu
[2] Department of Computer Science, New Mexico State University
tile@cs.nmsu.edu
[3] Department of Industrial and Operations Engineering, University of Michigan
fioretto@umich.edu

Abstract. *Distributed Constraint Optimization Problems* (DCOPs) offer a powerful approach for the description and resolution of cooperative multi-agent problems. In this model, a group of agents coordinate their actions to optimize a global objective function, taking into account their preferences or constraints. A core limitation of this model is the assumption that the preferences of all agents or the costs of all constraints are specified a priori. Unfortunately, this assumption does not hold in a number of application domains where preferences or constraints must be elicited from the users. One of such domains is the *Smart Home Device Scheduling* (SHDS) problem. Motivated by this limitation, we make the following contributions in this paper: **(1)** We propose a general model for preference elicitation in DCOPs; **(2)** We propose several heuristics to elicit preferences in DCOPs; and **(3)** We empirically evaluate the effect of these heuristics on random binary DCOPs as well as SHDS problems.

Keywords: Distributed Constraint Optimization, Smart Homes, Preference Elicitation

Computing LP$^{\text{MLN}}$ *Using ASP and MLN Solvers (System Paper)*

Joohyung Lee (Advisor), Samidh Talsania (Co-author), and Yi Wang (Student)
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, USA
(*e-mail:* {joolee, stalsani, ywang485}@asu.edu)

submitted ; revised ; accepted

Abstract

LP$^{\text{MLN}}$ is a recent addition to probabilistic logic programming languages. Its main idea is to overcome the rigid nature of the stable model semantics by assigning a weight to each rule in a way similar to Markov Logic is defined. We present two implementations of LP$^{\text{MLN}}$, LPMLN2ASP and LPMLN2MLN. System LPMLN2ASP translates LP$^{\text{MLN}}$ programs into the input language of answer set solver CLINGO, and using weak constraints and stable model enumeration, it can compute most probable stable models as well as exact conditional and marginal probabilities. System LPMLN2MLN translates LP$^{\text{MLN}}$ programs into the input language of Markov Logic solvers, such as ALCHEMY, TUFFY, and ROCKIT, and allows for performing approximate probabilistic inference on LP$^{\text{MLN}}$ programs. We also demonstrate the usefulness of the LP$^{\text{MLN}}$ systems for computing other languages, such as ProbLog and Pearl's Causal Models, that are shown to be translatable into LP$^{\text{MLN}}$.

This paper is under review for the second round.

An Efficient SMT Approach to Solve MRCPSP/max Instances with Tight Constraints on Resources

Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret

University of Girona, Spain
{miquel.bofill,jordi.coll,josep.suy,mateu.villaret}@imae.udg.edu

Abstract. The Multi-Mode Resource-Constrained Project Scheduling Problem with Minimum and Maximum Time Lags (MRCPSP/max) is a generalization of the well known Resource-Constrained Project Scheduling Problem. Recently, it has been shown that the benchmark datasets typically used in the literature can be easily solved by relaxing some resource constraints, which in many cases are dummy. In this work we propose new datasets with tighter resource limitations. We tackle them with an SMT encoding, where resource constraints are expressed as specialized pseudo-Boolean constraints and then translated into SAT. We provide empirical evidence that this approach is state-of-the-art for instances highly constrained by resources.

Combining Stochastic Constraint Optimization and Probabilistic Programming

From Knowledge Compilation to Constraint Solving

Anna L.D. Latour[1], Behrouz Babaki[2], Anton Dries[2], Angelika Kimmig[2], Guy Van den Broeck[3], and Siegfried Nijssen[4]

[1] LIACS, Leiden University[⋆], Leiden, The Netherlands
`a.l.d.latour@liacs.leidenuniv.nl`
[2] Department of Computer Science, KU Leuven, Leuven, Belgium
[3] Computer Science Department, UCLA, Los Angeles, USA
[4] ICTEAM, Université catholique de Louvain, Louvain-la-Neuve, Belgium
`siegfried.nijssen@uclouvain.be`

Abstract. We show that a number of problems in Artificial Intelligence can be seen as Stochastic Constraint Optimization Problems (SCOPs): problems that have both a stochastic and a constraint optimization component. We argue that these problems can be modeled in a new language, SC-ProbLog, that combines a generic Probabilistic Logic Programming (PLP) language, ProbLog, with stochastic constraint optimization. We propose a toolchain for effectively solving these SC-ProbLog programs, which consists of two stages. In the first stage, decision diagrams are compiled for the underlying distributions. These diagrams are converted into models that are solved using Mixed Integer Programming or Constraint Programming solvers in the second stage. We show that, to yield linear constraints, decision diagrams need to be compiled in a specific form. We introduce a new method for compiling small Sentential Decision Diagrams in this form. We evaluate the effectiveness of several variations of this toolchain on test cases in viral marketing and bioinformatics.

Acknowledgements. We thank Luc De Raedt for his support, for his advice and for the numerous other ways in which he contributed to this work. This research was supported by the Netherlands Organisation for Scientific Research (NWO) and NSF grant #IIS-1657613.

[⋆] The inspiration for this work came during a research visit to the Computer Science Department of KU Leuven. The work itself was done during a research visit to the ICTEAM institute of Université catholique de Louvain.

Constraint Handling in Flight Planning

Anders N. Knudsen (student), Marco Chiarandini, and Kim S. Larsen

Department of Mathematics and Computer Science
University of Southern Denmark, Campusvej 55, DK-5230 Odense M, Denmark
{andersnk,marco,kslarsen}@imada.sdu.dk

Abstract. Flight routes are paths in a network, whose nodes represent waypoints in a 3D space. A common approach to route planning is first to calculate a cheapest path in a 2D space, and then to optimize the flight cost in the third dimension. We focus on the problem of finding a cheapest path through a network describing the 2D projection of the 3D waypoints. In European airspaces, traffic flow is handled by heavily constraining the flight network. The constraints can have very diverse structures, among them a generalization of the forbidden pairs type. They invalidate the FIFO property, commonly assumed in shortest path problems. We formalize the problem and provide a framework for the description, representation and propagation of the constraints in path finding algorithms, best-first and A* search. In addition, we study a lazy approach to deal with the constraints. We conduct an experimental evaluation based on real-life data and conclude that our techniques for constraint propagation work best together with an iterative search approach, in which only constraints that are violated in previously found routes are introduced in the constraint set before the search is restarted.

Treewidth in Non-Ground Answer Set Solving and Alliance Problems in Graphs

Bernhard Bliem (student) and Stefan Woltran (thesis advisor)

Institute of Information Systems 184/2
TU Wien
Favoritenstrasse 9–11, 1040 Vienna, Austria
[bliem,woltran]@dbai.tuwien.ac.at

Abstract. To solve hard problems efficiently via answer set programming (ASP), a promising approach is to take advantage of the fact that real-world instances of many hard problems exhibit small treewidth. Algorithms that exploit this have already been proposed – however, they suffer from an enormous overhead. In the thesis, we present improvements in the algorithmic methodology for leveraging bounded treewidth that are especially targeted toward problems involving subset minimization. This can be useful for many problems at the second level of the polynomial hierarchy like solving disjunctive ground ASP. Moreover, we define classes of non-ground ASP programs such that grounding such a program together with input facts does not lead to an excessive increase in treewidth of the resulting ground program when compared to the treewidth of the input. This allows ASP users to take advantage of the fact that state-of-the-art ASP solvers perform better on ground programs of small treewidth. Finally, we resolve several open questions on the complexity of alliance problems in graphs. In particular, we settle the long-standing open questions of the complexity of the Secure Set problem and whether the Defensive Alliance problem is fixed-parameter tractable when parameterized by treewidth.

Keywords: answer set programming, treewidth, secure set, defensive alliance, parameterized complexity

1 Introduction

The problem solving paradigm Answer Set Programming (ASP) [12, 31, 47, 46] has become quite popular for tackling computationally hard problems. It offers its users a very convenient declarative language that allows for succinct specifications, and there are highly efficient systems available [33, 32, 30, 2, 3, 45, 4, 52, 24].

Although ASP systems have made huge advances in performance, they still struggle with several tough problems. This is not always just an issue of computational complexity in the classical sense. Interestingly, ASP systems may perform quite well in practice on one problem whereas the performance on another problem of the same complexity can be significantly worse. Often classical complexity theory is thus only of limited help to explain ASP solving performance in practice.

In such cases, it may be insightful to consider the *parameterized* complexity of the problems [21, 29, 18, 49]. This theoretical framework investigates the complexity of a problem not only in terms of the input size, but also of other parameters.

In this work, we are particularly interested in the effect of the structural parameter *treewidth* [51] on the performance of ASP solvers. Intuitively, the smaller the treewidth of a graph, the closer the graph resembles a tree. It is well known that many graph problems become easy if we restrict the input to trees and it has turned out that for many important problems this even holds for the more general class of instances of bounded treewidth [5]. Luckily, it has been observed that real-world instances usually exhibit small treewidth [10, 53, 42]. Treewidth is not only relevant for graph problems. It can also be applied to instances of all kinds of problems by choosing a suitable representation of the instance as a graph. For instance, treewidth has also been considered for constraint satisfaction problems [20], where it is known under the name of "induced width" and is crucial for the performance of a technique called bucket elimination [19].

There have already been some investigations concerning treewidth and ground ASP (i.e., ASP programs without variables, also known as propositional programs) [35, 50, 27]. An important result is the algorithm from [40] for deciding whether a ground ASP program has a solution in linear time on instances of bounded treewidth. This algorithm employs a technique called *dynamic programming on tree decompositions*, which is very common for algorithms that exploit small treewidth. The algorithm from [40] has also been implemented and proposed as an alternative solver for ground ASP [48]. For certain problems, this dynamic-programming-based solver was able to outperform state-of-the-art solvers if the instances had a very small treewidth and very large size.

Although the encouraging results from [48] confirmed that small treewidth can be successfully exploited for ASP solving in experimental settings, the restrictions on problems and instances that make this approach perform well were still too severe for most practical applications. The main obstacles that prevented this approach from being useful for a broad range of applications were the facts that, on the one hand, the naive dynamic programming approach involves an enormous overhead (especially in terms of memory) and, on the other hand, state-of-the-art ASP solvers often perform so well that the theoretical superiority of the dynamic programming algorithm only pays off for instances of tremendous size. In fact, experiments in [7] indicated that state-of-the-art ASP solvers are "sensitive" to the treewidth of their input in the sense that smaller treewidth strongly correlates with higher solving performance.

These issues hint at interesting research challenges. In particular, two approaches seem promising for successfully exploiting small treewidth for ASP solving in practice:

– The first research challenge is to improve the dynamic-programming-based methodology in order to reduce its overhead and redundant computations. For solving ground ASP, these issues are especially severe compared to other problems because the corresponding computational problems are even harder than NP under standard complexity-theoretic assumptions. (In fact, deciding whether a ground ASP program with disjunctions has an answer set is Σ_2^P-complete.) This high complexity of ground ASP is mirrored in the dynamic programming algorithm [40], which uses brute force to first of all find all models of all parts of the decomposed program, and it subsequently uses brute force again *for each such partial model* to find all potential counterexamples that may cause the candidate to be discarded.

This pattern also frequently occurs in dynamic programming algorithms for other problems that search for solutions satisfying some form of subset minimality. Besides ground ASP, this is the case, for instance, for the problem of finding subset-minimal models of a propositional formula. In general, problems involving subset-minimization are quite common in AI, and dynamic programming algorithms have been proposed for, e.g., circumscription, abduction or abstract argumentation (see [39, 36, 22]). Such algorithms typically store a great number of redundant objects because the subsets that may invalidate a solution candidate are themselves solution candidates. Moreover, the specifications of such algorithms themselves contain redundancies because the potential counterexamples are usually manipulated in almost the same way as the solution candidates.

– The second research challenge is to solve ASP by not doing dynamic programming at all but instead exploiting small treewidth *implicitly* by relying on the assumption that state-of-the-art solvers perform better when given ground programs of small treewidth (as indicated by the experiments in [7]). Since problems are usually encoded in non-ground ASP, here the objective is to investigate which non-ground encoding techniques significantly blow up the treewidth of the grounding compared to the treewidth of the input.

In addition to leveraging treewidth for ASP solving, we are interested in several variants of a graph problem called Secure Set [13]. It belongs to the class of so-called *alliance problems* [43, 25, 54], which are problems that ask for groups of vertices that help each other out in a certain way. Practical applications of alliance problems include finding groups of websites that form communities [28] or distributing resources in a computer network in such a way that simultaneous requests can be satisfied [37]. Intuitively, a set S of vertices in a graph is secure if every subset of S has as least as many neighbors in S as neighbors not in S. The Secure Set problem asks whether a given graph contains a secure set at most of a certain size.

The reason why we are concerned with Secure Set is that this problem has quite interesting properties, especially for ASP researchers: Attempts of encoding this problem in ASP have resulted in very involved specifications indicating that Secure Set may require the full expressive power of ASP [1]. However, it is unfortunately unclear whether this is really necessary because its complexity has still remained unresolved although the problem has been introduced already in 2007 [13].

One of the variants of Secure Set that we consider in the proposed thesis is the Defensive Alliance problem [43, 44]. This problem has received quite some attention in the literature [25]. It is known to be NP-complete, but its complexity when parameterized by treewidth has remained open.

2 Background

We assume some familiarity with ASP; introductions can be found in [12, 31, 47, 46]. In the thesis, we study both ground ASP programs, i.e., programs without variables, but also programs utilizing the full language described in the ASP-Core-2 specification [14]. In particular, we include weak constraints and aggregates.

To solve a non-ground ASP program, ASP systems usually first invoke a *grounder* that transforms a program into a set of ground rules. The answer sets of the original program are the *stable models* (as defined in [34]) of the resulting ground program.

A "naive" grounder blindly instantiates variables by all possible ground terms. Grounders in practice, on the other hand, employ sophisticated techniques in order to keep the resulting ground program as small as possible. As these techniques differ between systems, we define a simplified notion of grounding that is easier to study. For a meaningful investigation of the relationship between the treewidth of the input and the treewidth of the grounding, we need to assume that the grounder performs some basic simplifications. These simplifications are so basic that they can be assumed to be implemented by all reasonable grounders. The intuition is that a rule from the "naive" grounding is omitted in our grounding whenever its positive body contains an atom that cannot possibly be derived.

Definition 1. *Let Π be a non-ground ASP program, let Π^+ denote the positive program obtained from Π by removing all negated atoms and replacing disjunctions with conjunctions (i.e., splitting disjunctive into normal rules), and let M^+ be the unique minimal model of Π^+. The* grounding *of Π, denoted by $\mathrm{gr}(\Pi)$, is such that, for every substitution s from variables to constants, $s(r) \in \mathrm{gr}(\Pi)$ iff $s(B^+(r)) \subseteq M^+$.*

In our work, we are interested in the treewidth of ground ASP programs. For this, we represent programs as graphs as follows.

Definition 2. *The* primal graph *of a ground ASP program Π is an undirected graph whose vertices are the atoms in Π and there is an edge between two atoms if they appear together in a rule in Π. The* treewidth *of a ground ASP program Π is the treewidth of its primal graph.*

Deciding whether a disjunctive ground ASP program has a stable model is Σ_2^{P}-complete in general [23], and it can be done in linear time for ground programs of bounded treewidth [35]. Treewidth is an important parameter studied in the context of *parameterized complexity theory*. Here, decision problems consist not only of an instance and a yes-no question, but additionally of a parameter of the instance. For introductions, we refer to [21, 29, 18, 49]. The central notion of tractability is called *fixed-parameter tractability*.

Definition 3. *A problem is* fixed-parameter tractable *(FPT) w.r.t. a parameter k of the instances if it admits an algorithm that runs in time $\mathcal{O}(f(k) \cdot n^c)$, where f is an arbitrary computable function that only depends on k, n is the input size and c is an arbitrary constant. We call such an algorithm an* FPT algorithm.

Note that the factor $f(k)$ in this running time may be exponential in the parameter k, but if k is bounded by a constant, then the algorithm runs in polynomial time. Importantly, the degree c of the polynomial must be a constant and may not depend on the parameter, otherwise the algorithm is not FPT.

Dynamic programming on tree decompositions is perhaps the most common technique for obtaining FPT algorithms when the parameter is treewidth. It is employed in the algorithm for solving ground ASP in [40], for instance. The basic idea is the following: Given a graph G, a tree decomposition of G is a tree whose nodes correspond to subgraphs of G according to certain conditions. If the treewidth of a graph is bounded by a constant, then we can find (in linear time) a tree decomposition whose nodes correspond to subgraphs of constant size [11]. We can then solve many problems by first applying brute force at each subgraph in order to solve a subproblem corresponding to this subgraph and then trying to combine the obtained partial solutions. Due to the bound on the treewidth, we can afford this brute force approach because each of the considered subgraphs has bounded size. Formal definitions and examples of this technique can be found in, e.g., [49].

3 Contributions

Our contributions can be arranged in three groups: First, we present improvements in the dynamic programming methodology; second, we define non-ground ASP classes that can be shown to preserve bounded treewidth of the input in grounding; third, we provide complexity results and algorithms for alliance problems in graphs.

3.1 Improvements in the Dynamic Programming Methodology

We present an improved dynamic programming methodology for problems that involve subset minimization. Specifically, for any problem P whose solutions are exactly the subset-minimal solutions of some base problem B, we formalize how a dynamic programming algorithm for B can automatically be transformed into a dynamic programming algorithm for P. We prove that the resulting algorithm runs in linear time on instances of bounded treewidth if the base algorithm does. Moreover, we prove that it is correct if the base algorithm is correct and, intuitively, it only computes partial solutions that do not "revoke decisions" made by associated partial solutions further down in the tree decomposition. The resulting algorithm has two advantages compared to solving P directly in a naive way: first, it is usually easier to specify because we only need to design an algorithm for the base problem and need not care about subset minimization; second, it is potentially more efficient because it stores fewer redundant items.

Indeed, this methodology has been empirically shown to lead to significant performance benefits for several problems [6]. An improved version of the classical dynamic programming algorithm for ground ASP has been implemented using these ideas [26] and proved to be significantly more efficient than the algorithm from [40]. Our result formalizes the common scheme that underlies these algorithms. We thus provide a formal framework that makes it possible to transfer the mentioned optimizations easily to other problems. Thereby we make the impressive performance benefits that have been reported in [6, 26] accessible to algorithm designers working on related problems. This is primarily useful for problems on the second level of the polynomial hierarchy as subset minimization is a recurring theme in many such problems.

3.2 Non-Ground ASP Classes that Preserve Bounded Treewidth

We define non-ground ASP classes for which grounding, according to Definition 1, preserves bounded treewidth of the input. By restricting the syntax, we define two classes of programs called *guarded* and *connection-guarded* programs [7]. Guarded programs guarantee that the treewidth of any fixed program after grounding stays small whenever input has small treewidth. We formally prove this property and show that, despite their restrictions, guarded programs can still express problems that are complete for the second level of the polynomial hierarchy.

Connection-guarded programs are even more expressive than guarded programs. We show that the treewidth of any fixed connection-guarded program after grounding is small whenever the treewidth *and the maximum degree* of (a graph representation of) the input facts is small.

These results bring us closer to the goal of implicitly taking advantage of the apparent sensitivity to treewidth of modern ASP solvers because they give us insight into what happens to the treewidth of the input during grounding. Thus, by writing a program in guarded ASP, we can be sure that the grounder does not destroy the property of bounded treewidth. In the case of connection-guarded ASP, the same holds for the combination of treewidth and maximum degree.

In the thesis, we also present a complexity analysis of computational problems corresponding to these classes when the parameter is the treewidth of the input, the maximum degree of the input, or the combination of both. The results of this analysis show that, for any fixed guarded ASP program, answer set solving is FPT when parameterized by the treewidth of the input; moreover, for any fixed connection-guarded ASP program, answer set solving is FPT when parameterized by the combination of treewidth and maximum degree. This is not obvious because our ASP classes support weak constraints and aggregates, which are not accounted for in the FPT algorithms [40, 26] for ground ASP. Furthermore, we prove hardness results showing that for connection-guarded ASP programs *both* the treewidth and the maximum degree must be bounded for obtaining fixed-parameter tractability. We do this by presenting a connection-guarded ASP encoding of a problem that is NP-hard even if the treewidth of the instances is fixed and by presenting a guarded encoding of a problem that is $\mathsf{\Sigma_2^P}$-hard even if the degree of the instances is fixed.

As a side-product of these investigations, we obtain *metatheorems* for proving FPT results. That is, we can prove that a problem is FPT when parameterized by treewidth by simply expressing the problem in guarded ASP. We compare this metatheorem to the common approach of proving fixed-parameter tractability by expressing a problem in monadic second-order logic and invoking the well-known theorem by Courcelle [16, 17]. Similarly, we can prove that a problem is FPT when parameterized by the combination of treewidth and maximum degree by expressing the problem in connection-guarded ASP. This result is appealing because we are not aware of any metatheorems that allow us to obtain FPT results for the combination of treewidth and degree as the parameter.

3.3 Alliance Problems in Graphs

We perform a complexity analysis of alliance problems in graphs, both in the classical setting and when parameterized by treewidth. First, we settle the complexity of Secure Set by proving that the problem, along with several variants, is Σ_2^{P}-complete (and thus at the second level of the polynomial hierarchy).

Next we turn to the complexity of Secure Set and Defensive Alliance when the problems are parameterized by treewidth. We illustrate the use of our ASP classes as FPT classification tools by presenting simple encodings for alliance problems in graphs. By encoding the NP-complete Defensive Alliance problem in connection-guarded ASP, we easily obtain the already known result that the problem is FPT when parameterized by the combination of treewidth and maximum degree. More importantly, we obtain the new result that the co-NP-complete problem of deciding whether a given set is secure in a graph is FPT for the parameter treewidth by encoding the problem in guarded ASP.

We also give several negative results. We prove that both Defensive Alliance and Secure Set, as well as several problem variants, are not FPT when parameterized by treewidth (under commonly held complexity-theoretic assumptions). These questions have been open since the problems have been introduced in 2002 and 2007, respectively. They have explicitly been stated as open problems in [41] (for Defensive Alliance) and in [38] (for Secure Set).

Despite the parameterized hardness of Secure Set, we can give at least a slightly positive result: We show that the Secure Set problem can still be solved in polynomial time for instances of bounded treewidth although the degree of the polynomial depends on the treewidth.

4 Current Status

The largest part of the research for the proposed thesis has already been done and is in the process of being integrated and written down. Most of the results have been published in conference proceedings and journals:

The work on improving the dynamic programming methodology for problems involving subset minimization has been published in [6]. The class of connection-guarded ASP programs, which preserves bounded treewidth of the input in grounding whenever the maximum degree is also bounded has been published in [7]. That paper neither contained the thorough complexity analysis performed in the proposed thesis nor the work on the class of guarded programs, which may be attractive because this class does not require the degree of input graphs to be bounded. The Σ_2^P-completeness result of the Secure Set problem has been published in [9]. An extended version [8], which is currently under review for a journal, additionally contains the parameterized complexity results. The proposed thesis extends this by results on the parameterized complexity of the Defensive Alliance problem as well.

5 Open Issues

The class of connection-guarded ASP programs may be of interest for algorithmic purposes because it allows us to classify a problem as FPT when parameterized by treewidth plus degree. A common technique for classifying problems parameterized by treewidth as FPT is expressing them in monadic second-order logic (MSO). Our result may lead to an extension of MSO that can be used for classifying problems as FPT when the parameter is treewidth + degree.

From a more practical perspective, it is promising to look closely into what ASP solvers and in particular their heuristics are doing when they are presented with a grounding of small treewidth. This could provide us insight into why state-of-the-art ASP solvers perform better on instances of small treewidth even though they do not "consciously" exploit this fact. With the gained understanding, we may be able to improve their performance by explicitly taking information from a tree decomposition into account during solving. This could perhaps lead to a hybrid ASP solving approach that uses classical conflict-driven clause learning in combination with techniques based on tree decompositions.

We showed that Secure Set is not FPT when parameterized by treewidth (unless the class $\mathsf{W}[1]$ is equal to FPT). It would be interesting to study which additional restrictions beside bounded treewidth need to be imposed on Secure Set instances to achieve fixed-parameter tractability. In particular, we do not know whether it becomes FPT when additionally the degree is bounded.

Regarding our polynomial-time algorithm for Secure Set on instances of bounded treewidth, it would be interesting to study if this result can be extended to instances of bounded clique-width, a parameter related to treewidth.

Moreover, we have not considered a problem that is closely related to Defensive Alliance, namely Offensive Alliance. Possibly some of our techniques can also be applied to obtain complexity results for this problem.

Defensive Alliance differs from Secure Set in the size of the subsets of solution candidates that need to be checked. For future work it would be interesting to study the complexity of a problem that generalizes both of them, where the size of the subsets is a parameter.

Finally, for the parameterized hardness results that we obtained we do not have corresponding membership results. This is an obvious task for future work.

References

1. Michael Abseher, Bernhard Bliem, Günther Charwat, Frederico Dusberger, and Stefan Woltran. Computing secure sets in graphs using answer set programming. *J. Logic Comput.*, 2015. Accepted for publication.
2. Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In Pedro Cabalar and Tran Cao Son, editors, *Proceedings of LPNMR 2013*, volume 8148 of *LNCS*, pages 54–66. Springer, 2013.
3. Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In Calimeri et al. [15], pages 40–54.
4. Mario Alviano, Wolfgang Faber, Nicola Leone, Simona Perri, Gerald Pfeifer, and Giorgio Terracina. The disjunctive datalog system DLV. In Oege de Moor, Georg Gottlob, Tim Furche, and Andrew Jon Sellers, editors, *Revised Selected Papers of Datalog 2010*, volume 6702 of *LNCS*, pages 282–301. Springer, 2011.
5. Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.
6. Bernhard Bliem, Günther Charwat, Markus Hecher, and Stefan Woltran. D-FLAT^2: Subset minimization in dynamic programming on tree decompositions made easy. *Fund. Inform.*, 147(1):27–61, 2016.
7. Bernhard Bliem, Marius Moldovan, Michael Morak, and Stefan Woltran. The impact of treewidth on ASP grounding and solving. In Carles Sierra and Fahiem Bacchus, editors, *Proceedings of IJCAI 2017*. The AAAI Press, 2017. Accepted for publication.
8. Bernhard Bliem and Stefan Woltran. Complexity of secure sets. *CoRR*, abs/1411.6549, 2014. Updated to version 3 on July 11, 2017.
9. Bernhard Bliem and Stefan Woltran. Complexity of secure sets. In Ernst W. Mayr, editor, *Revised Papers of WG 2015*, volume 9224 of *LNCS*, pages 64–77. Springer, 2016.
10. Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernet.*, 11(1-2):1–21, 1993.
11. Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
12. Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.
13. Robert C. Brigham, Ronald D. Dutton, and Stephen T. Hedetniemi. Security in graphs. *Discrete Appl. Math.*, 155(13):1708–1714, 2007.
14. Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. https://www.mat.unical.it/aspcomp2013/ASPStandardization, 2015. Version: 2.03c.
15. Francesco Calimeri, Giovambattista Ianni, and Mirosław Truszczyński, editors. *Proceedings of LPNMR 2015*, volume 9345 of *LNCS*. Springer, 2015.
16. Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.
17. Bruno Courcelle. The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *RAIRO Theor. Inform. Appl.*, 26:257–286, 1992.
18. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer International Publishing, Cham, Switzerland, 2015.

19. Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.
20. Rina Dechter. *Constraint Processing.* Elsevier Morgan Kaufmann, Amsterdam, The Netherlands, 2003.
21. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, New York, NY, USA, 1999.
22. Wolfgang Dvořák, Reinhard Pichler, and Stefan Woltran. Towards fixed-parameter tractable algorithms for abstract argumentation. *Artificial Intelligence*, 186:1–37, 2012.
23. Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 15(3-4):289–323, 1995.
24. Islam Elkabani, Enrico Pontelli, and Tran Cao Son. Smodels$^{\mathrm{A}}$ - A system for computing answer sets of logic programs with aggregates. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Proceedings of LPNMR 2005*, volume 3662 of *LNCS*, pages 427–431. Springer, 2005.
25. Henning Fernau and Juan A. Rodríguez-Velázquez. A survey on alliances and related parameters in graphs. *Electron. J. Graph Theory Appl. (EJGTA)*, 2(1):70–86, 2014.
26. Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In Marcello Balduccini and Tomi Janhunen, editors, *Proceedings of LPNMR 2017*, volume 10377 of *LNCS*, pages 132–145. Springer, 2017.
27. Johannes Klaus Fichte and Stefan Szeider. Backdoors to tractable answer set programming. *Artificial Intelligence*, 220:64–103, 2015.
28. Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3):66–71, 2002.
29. Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. Springer, 2006.
30. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten Schaub. Progress in clasp series 3. In Calimeri et al. [15], pages 368–383.
31. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, Williston, VT, USA, 2012.
32. Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp*: A conflict-driven answer set solver. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Proceedings of LPNMR 2007*, volume 4483 of *LNCS*, pages 260–265. Springer, 2007.
33. Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89, 2012.
34. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of JICSLP 1988*, volume 2, pages 1070–1080. The MIT Press, 1988.
35. Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artificial Intelligence*, 174(1):105–132, 2010.
36. Georg Gottlob, Reinhard Pichler, and Fang Wei. Tractable database design and datalog abduction through bounded treewidth. *Inf. Syst.*, 35(3):278–298, 2010.
37. Teresa W. Haynes, Stephen T. Hedetniemi, and Michael A. Henning. Global defensive alliances in graphs. *Electron. J. Combin.*, 10, 2003.

38. Yiu Yu Ho and Ronald D. Dutton. Rooted secure sets of trees. *AKCE Int. J. Graphs Comb.*, 6(3):373–392, 2009.
39. Michael Jakl, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Fast counting with bounded treewidth. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Proceedings of LPAR 2008*, volume 5330 of *LNCS*, pages 436–450. Springer, 2008.
40. Michael Jakl, Reinhard Pichler, and Stefan Woltran. Answer-set programming with bounded treewidth. In Craig Boutilier, editor, *Proceedings of IJCAI 2009*, pages 816–822. The AAAI Press, 2009.
41. Masashi Kiyomi and Yota Otachi. Alliances in graphs of bounded clique-width. *Discrete Appl. Math.*, 223:91–97, 2017.
42. András Kornai and Zsolt Tuza. Narrowness, pathwidth, and their application in natural language processing. *Discrete Appl. Math.*, 36(1):87–92, 1992.
43. Petter Kristiansen, Sandra M. Hedetniemi, and Stephen T. Hedetniemi. Introduction to alliances in graphs. In Ilyas Cicekli, Nihan Kesim Cicekli, and Erol Gelenbe, editors, *Proceedings of ISCIS 2002*, pages 308–312. CRC Press, 2002.
44. Petter Kristiansen, Sandra M. Hedetniemi, and Stephen T. Hedetniemi. Alliances in graphs. *J. Combin. Math. Combin. Comput.*, 48:157–178, 2004.
45. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.
46. Vladimir Lifschitz. What is answer set programming? In Dieter Fox and Carla P. Gomes, editors, *Proceedings of AAAI 2008)*, pages 1594–1597. The AAAI Press, 2008.
47. Victor W. Marek and Mirosław Truszczyński. Stable models and an alternative logic programming paradigm. In Krzysztof Apt, Victor W. Marek, Mirosław Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*, pages 375–398. Springer, New York, NY, USA, 2011.
48. Michael Morak, Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. A dynamic-programming based ASP-solver. In Tomi Janhunen and Ilkka Niemelä, editors, *Proceedings of JELIA 2010*, volume 6341 of *LNCS*, pages 369–372. Springer, 2010.
49. Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, United Kingdom, 2006.
50. Reinhard Pichler, Stefan Rümmele, Stefan Szeider, and Stefan Woltran. Tractable answer-set programming with weight constraints: Bounded treewidth is not enough. *Theory Pract. Log. Program.*, 14(2):141–164, 2014.
51. Neil Robertson and Paul D. Seymour. Graph minors. III. Planar tree-width. *J. Combin. Theory Ser. B*, 36(1):49–64, 1984.
52. Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
53. Mikkel Thorup. All structured programs have small tree-width and good register allocation. *Inform. and Comput.*, 142(2):159–181, 1998.
54. Ismael González Yero and Juan A. Rodríguez-Velázquez. Defensive alliances in graphs: A survey. *CoRR*, abs/1308.2096, 2013.

Search strategies for floating point constraint systems [⋆]

Heytem Zitoun[1⋆⋆], Claude Michel[1⋆⋆⋆], Michel Rueher[1†], and Laurent Michel[2‡]

[1] Université Côte d'Azur, CNRS, I3S, France
firstname.lastname@i3s.unice.fr
[2] University of Connecticut, Storrs, CT 06269-2155
ldm@engr.uconn.edu

Abstract. The ability to verify critical software is a key issue in embedded and cyber physical systems typical of automotive, aeronautics or aerospace industries. Bounded model checking and constraint programming approaches search for counter-examples that exemplify a property violation. The search of such counter-examples is a long, tedious and costly task especially for programs performing floating point computations. Indeed, available search strategies are dedicated to finite domains and, to a lesser extent, to continuous domains. In this paper, we introduce new strategies dedicated to floating point constraints. They take advantage of the properties of floating point domains (e.g., domain density) and of floating point constraints (e.g., floating point arithmetic) to improve the search for floating point constraint problems. First experiments on a set of realistic benchmarks show that such dedicated strategies outperform standard search and splitting strategies.

[⋆] This work was partially supported by ANR COVERIF (ANR-15-CE25-0002).
[⋆⋆] Student
[⋆⋆⋆] Advisor
[†] Advisor
[‡] Co-author

Combining Solvers to Solve a Cryptanalytic Problem[⋆]

David Gerault[1], Pascal Lafourcade[1], Marine Minier[2], and Christine Solnon[3]

[1] Université Clermont Auvergne, LIMOS, France
[2] Université de Lorraine, LORIA, UMR 7503, F-54506, France
[3] Université de Lyon, INSA-Lyon, LIRIS, CNRS UMR5205, F-69621, France

Abstract. We describe Constraint Programming models to solve a cryptanalytic problem: the chosen key differential attack against the standard block cipher AES. We more particularly show how combining two solvers, namely Picat_Sat and Chuffed, greatly speeds-up the solution process and allows us to solve all instances in less than twelve hours while dedicated cryptanalysis tools need weeks.

1 Introduction

Since 2001, AES (Advanced Encryption Standard) is the encryption standard for block ciphers [7]. It guarantees communication confidentiality by using a secret key K to cipher an original plaintext X into a ciphertext $AES_K(X)$, in such a way that the ciphertext can further be deciphered into the original one using the same key, *i.e.*, $X = AES_K^{-1}(AES_K(X))$. It uses either 128-, 192-, or 256-bit keys. Cryptanalysis aims at testing whether confidentiality is actually guaranteed. In particular, differential cryptanalysis [2] evaluates whether it is possible to find the key within a reasonable number of trials by considering plaintext pairs (X, X') and studying the propagation of the initial difference $\delta X = X \oplus X'$ between X and X' while going through the ciphering process (where $\oplus$ is the xor operator). Today, differential cryptanalysis is public knowledge, and block ciphers such as AES have proven bounds against differential attacks. Hence, [1] proposed a new type of attack called related-key attack that allows an attacker to also inject differences between the keys K and K' (even if the secret key K remains unknown from the attacker).

To mount related-key attacks, the cryptanalyst must find related-key differentials, defined as a plaintext difference δX, a key difference δK, and a ciphertext difference δC. The optimal ones maximize the probability that, for random plaintexts X and key K, an input difference $(\delta X, \delta K)$ leads to the output difference δC, *i.e.*, it holds that $AES_K(X) \oplus AES_{K \oplus \delta K}(X \oplus \delta X) = \delta C$. Finding the optimal related-key differentials for AES is a highly combinatorial problem that hardly scales. Two main approaches have been proposed to solve this problem: a graph traversal approach [8], and a Branch & Bound approach [4]. The approach of [8] requires about 60 GB of memory when the key has 128 bits, and it has not been extended to larger keys. The approach of [4] only takes several megabytes of memory, but it requires several days of computation when the key has 128 bits, and several weeks when the key has 192 bits.

[⋆] This research was conducted with the support of the FEDER program of 2014-2020 and the region council of Auvergne

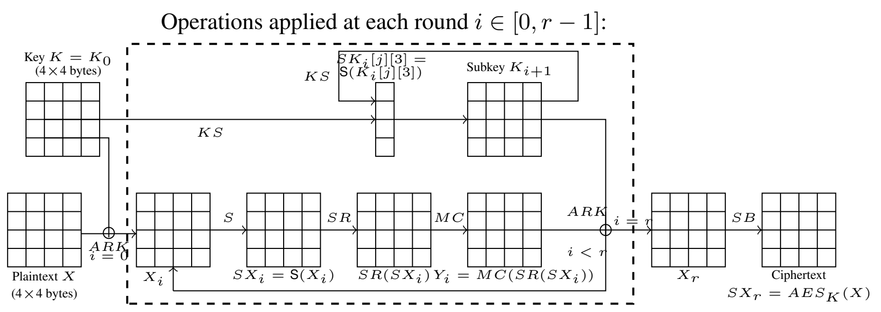


Fig. 1. AES-128 cipher. Each 4×4 array represents a group of 16 bytes. Before the first round, ARK is applied on X and K to obtain X_0. Then, for each round $i \in [0, r-1]$, S is applied on X_i to obtain SX_i, SR and MC are applied on SX_i to obtain Y_i, KS is applied on K_i to obtain K_{i+1} (and during KS, S is applied on $K_i[j][3]$ to obtain $SK_i[j][3]$, $\forall j \in [0, 3]$), and ARK is applied on K_{i+1} and Y_i to obtain X_{i+1}. The ciphertext SX_r is obtained by applying SB on X_r.

During the process of designing new ciphers, this search generally needs to be performed several times, so it is desirable that it can be done rather quickly. Another point that should not be neglected is the time needed to design and implement these approaches: To ensure that the computation is completed within a “reasonable” amount of time, it is necessary to reduce the branching by introducing clever reasoning. Of course, this hard task is also likely to introduce bugs, and checking the correctness or the optimality of the computed solutions may not be so easy.

In [11], we introduced a first Constraint Programming (CP) model to solve this problem when the key has 128 bits. In [10], we extended this model to larger keys of 192 and 256 bits, and we improved it by introducing a new way for modeling byte equivalence classes. We also showed how to speed-up the solution process by combining two solvers: Picat_Sat [16] and Chuffed [5]. In this paper, we first describe the problem in Section 2, and the CP model of [10] in Section 3. In these two sections, we only consider the case where keys have 128 bits, as this simplifies the presentation. The extension to the case where keys have 192 or 256 bits is quite straightforward and we refer the reader to [10] for more details. Then, we study the solver combination more thoroughly in Section 4.

2 Problem Statement

AES block cipher. AES ciphers blocks of length $n = 128$ bits, and each block is a 4×4 matrix of bytes. The length of keys is $l \in \{128, 192, 256\}$. In this section, we only consider keys of length $l = 128$, and keys are 4×4 matrices of bytes. Given a 4×4 matrix of bytes M, we note $M[j][k]$ the byte at row $j \in [0, 3]$ and column $k \in [0, 3]$.

AES is an iterative process, and we note X_i the ciphertext at the beginning of round $i \in [0, r]$. Each round is composed of the following operations, as displayed in Fig. 1:

- `SubBytes` (S) is a non-linear permutation which is applied on each byte of X_i separately, *i.e.*, $\forall j, k \in [0,3]$, $X_i[j][k]$ is replaced by $\mathsf{S}(X_i[j][k])$, according to a lookup table. We note $SX_i = \mathsf{S}(X_i)$.
- `ShiftRows` (SR) is a linear mapping that rotates on the left by 1 byte position (resp. 2 and 3 byte positions) the second row (resp. third and fourth rows) of SX_i.
- `MixColumns` (MC) is a linear mapping that multiplies each column of $SR(SX_i)$ by a 4×4 fixed matrix chosen for its good properties of diffusion [6]. In particular, it has the Maximum Distance Separable (MDS) property: For each column, the total number of bytes which are different from 0, before and after applying MC, is either equal to 0 or strictly greater than 4. We note $Y_i = MC(SR(SX_i))$.
- `KeySchedule` (KS) is the operation that generates subkeys. The subkey at round 0 is the initial key, *i.e.*, $K_0 = K$. For each round $i \in [0, r-1]$, the subkey K_{i+1} is generated from K_i by applying KS. It first replaces each byte $K_i[j][3]$ of the last column by $\mathsf{S}(K_i[j][3])$ (where S is the `SubBytes` operator), and we note $SK_i[j][3] = \mathsf{S}(K_i[j][3])$. Then, each column of K_{i+1} is obtained by performing a xor operation between bytes coming from K_i, SK_i, or K_{i+1}.
- `AddRoundKey` (ARK) performs a xor between bytes of Y_i and subkey K_{i+1} to obtain X_{i+1}.

The set of all bytes (for all rounds) is denoted *Bytes* (*i.e.*, $\mathit{Bytes} = \{X[j][k], X_i[j][k], SX_i[j][k], Y_i[j][k], K_i[j][k], SK_i[j][3] \mid i \in [0,r], j, k \in [0,3]\}$). The difference between two bytes B and B' is denoted δB (*i.e.*, $\delta B = B \oplus B'$), and the set of all differential bytes is denoted *diffBytes* (*i.e.*, $\mathit{diffBytes} = \{\delta B \mid B \in \mathit{Bytes}\}$).

Optimal related-key differentials. Mounting attacks to recover the key K requires finding a related-key differential characteristic, *i.e.* a plaintext difference $\delta X = X \oplus X'$ and a key difference $\delta K = K \oplus K'$, such that δX becomes δSX_r after r rounds with a probability $Pr(\delta X \to \delta SX_r)$ as high as possible. This can be achieved by tracking the propagation of the initial differences through the cipher using known propagation rules. Once such an optimal differential characteristic is found, the cryptanalyst asks for the encryption of plaintext pairs (X, X') satisfying the input difference (*i.e.*, $X \oplus X' = \delta X$), with key K for X and key $K' = K \oplus \delta K$ for $\delta X'$. When this difference propagates as expected, he can infer informations leading to a recovery of the secret key K in $\mathcal{O}(\frac{1}{Pr(\delta X \to \delta SX_r)})$ trials.

The AES operators SR, MC, ARK, and KS are linear, *i.e.*, they propagate differences in a deterministic way (with probability 1). However, the S operator is not linear: Given a byte difference $B \oplus B' = \delta B$, the probability that δB becomes $\mathsf{S}(B) \oplus \mathsf{S}(B') = \delta SB$ is $Pr(\delta B \to \delta SB)$. It is equal to 1 if $\delta B = 0$ (*i.e.*, $B = B'$). However, if $\delta B \neq 0$, then $Pr(\delta B \to \delta SB) \in \{\frac{2}{256}, \frac{4}{256}\}$.

The probability that δX becomes δSX_r is equal to the product of all $Pr(\delta B \to \delta SB)$ such that δB (resp. δSB) is a byte difference before (resp. after) passing through the S operator when ciphering X with K and X' with K', *i.e.*, the product of all $Pr(\delta X_i[j][k] \to \delta SX_i[j][k])$ and all $Pr(\delta K_i[j][3] \to \delta SK_i[j][3])$ with $i \in [0, r]$ and $j, k \in [0, 3]$. The goal is to find δX and δK that maximize this probability.

Two step solving process. To find δX and δK, we search for the values of all differential bytes in *diffBytes*. Both [4] and [8] propose to solve this problem in two steps. In Step

1, a Boolean variable ΔB is associated with every differential byte $\delta B \in \mathit{diffBytes}$ such that $\Delta B = 0 \Leftrightarrow \delta B = 0$ and $\Delta B = 1 \Leftrightarrow \delta B \in [1, 255]$. The goal of Step 1 is to find a *Boolean solution* that assigns values to Boolean variables such that the AES transformation rules are satisfied. During this first step, the `SubBytes` operation S is not considered. Indeed, it does not introduce nor remove differences. Therefore, we have $\Delta X_i[j][k] = \Delta SX_i[j][k]$ and $\Delta K_i[j][3] = \Delta SK_i[j][3]$. As we search for a solution with maximal probability, the goal of Step 1 is to search for a Boolean solution which minimizes the number of variables $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ which are set to 1.

In Step 2, the Boolean solution is transformed into a *byte solution*: For each differential byte $\delta B \in \mathit{diffBytes}$, if the corresponding Boolean variable ΔB is assigned to 0, then δB is also assigned to 0; otherwise, we search for a byte value in $[1, 255]$ to be assigned to δB such that the AES transformation rules are satisfied and the probability is maximized. Note that some Boolean solutions may not be transformable into byte solutions. These Boolean solutions are said to be *byte-inconsistent*. In this paper, we focus on Step 1 which is the most challenging step: the CP model for Step 2 is rather straightforward when using table constraints, and all instances are efficiently solved by Choco [15] (see [10] for more details).

3 CP Model

In this section, we briefly describe the CP model used to solve Step 1, and refer the reader to [11, 10] for more details. We first introduce basic variables and constraints, that model the AES transformation rules in a straightforward way. Then, we show how to tighten this model by introducing new variables and constraints that model equality relations at the byte level.

Variables. For each differential byte $\delta B \in \mathit{diffBytes}$, we define a Boolean variable ΔB whose domain is $D(\Delta B) = \{0, 1\}$: it is assigned to 0 if $\delta B = 0$, and to 1 otherwise.

`XOR` constraint. As ARK and KS mainly perform `XOR` operations, we first define a `XOR` constraint. Let us consider three differential bytes δA, δB and δC such that $\delta A \oplus \delta B = \delta C$. If $\delta A = \delta B = 0$, then $\delta C = 0$. If ($\delta A = 0$ and $\delta B \neq 0$) or ($\delta A \neq 0$ and $\delta B = 0$) then $\delta C \neq 0$. However, if $\delta A \neq 0$ and $\delta B \neq 0$, then we cannot know if δC is equal to 0 or not: This depends on whether $\delta A = \delta B$ or not. When abstracting differential bytes δA, δB and δC with Boolean variables ΔA, ΔB and ΔC (which only model the fact that there is a difference or not), we obtain the following definition of the `XOR` constraint: `XOR`$(\Delta A, \Delta B, \Delta C) \Leftrightarrow \Delta A + \Delta B + \Delta C \neq 1$.

`AddRoundKey` and `KeySchedule` constraints. Both ARK and KS are modeled with `XOR` constraints between ΔX_i, ΔY_i, and ΔK_i variables (for ARK), and between ΔK_i and ΔSK_i variables (for KS).

`ShiftRows` and `MixColumns`. SR simply shifts variables. The MDS property of MC is ensured by posting a constraint on the sum of some variables of ΔX_i and ΔY_i variables, which must belong to the set $\{0, 5, 6, 7, 8\}$.

Objective function. We introduce an integer variable obj_{Step1} that must be minimized, and we post an equality constraint between obj_{Step1} and the sum of Boolean variables on which a non linear S operation is performed (all variables $\Delta X_i[j][k]$ and $\Delta K_i[j][3]$ with $i \in [0, r]$ and $j, k \in [0, 3]$).

Let v be the optimal value of obj_{Step1}. It may happen that none of the Boolean solutions with $obj_{Step1} = v$ is byte-consistent, or that the maximal probability p of Boolean solutions with $obj_{Step1} = v$ is such that it is possible to have a better probability with a larger value for obj_{Step1} (*i.e.*, $p < (\frac{4}{256})^{v+1}$). In this case, we need to search for new Boolean solutions, such that obj_{Step1} is minimal while being strictly greater than v. This is done by adding the constraint $obj_{Step1} > v$ before solving again Step 1.

Limitations. This basic CP model is complete, *i.e.*, for any solution at the byte level (on δ variables), there exists a solution at the Boolean level (on Δ variables). However, preliminary experiments have shown us that there is a huge number of Boolean solutions which are byte inconsistent. For example, when the number of rounds is $r = 4$, the optimal cost is $obj_{Step1} = 11$, and there are more than 90 millions of Boolean solutions with $obj_{Step1} = 11$. However, none of these solutions is byte-consistent. In this case, most of the Step 1 solving time is spent at generating useless Boolean solutions which are discarded in Step 2.

Class variables. When reasoning at the Boolean level, many solutions are not byte-consistent because constraints at the byte level have been ignored. However, the output of the XOR operations allow us to infer some information about the equalty or difference between the byte values represented by some of the differential bits. Thus, we introduce new variables that model equality relations between differential bytes. More precisely, for each differential byte $\delta A \in \mathit{diffBytes}$, we introduce a variable $Class_{\delta A}$ that models the equivalence class of δA: two differential bytes δA and δB belong to the same equivalence class (*i.e.*, $Class_{\delta A} = Class_{\delta B}$) iff $\delta A = \delta B$. The domain of $Class_{\delta A}$ is $[0; 255]$, as there are 256 byte values.

We add a constraint to assign class 0 to differential bytes with value 0, *i.e.*, $\Delta A = 0 \Leftrightarrow Class_{\delta A} = 0$. Also, we break symmetries due to the fact that equivalence classes may be swapped by enforcing an order on them with a precede constraint [12].

Revisiting `XOR` *with Class variables.* When defining the $\texttt{XOR}(\Delta A, \Delta B, \Delta C)$ constraint, if $\Delta A = \Delta B = 1$, then we cannot know whether ΔC is equal to 0 or 1. However, whenever $\Delta C = 0$ (resp. $\Delta C = 1$), we know for sure that the corresponding byte δC is equal to 0 (resp. different from 0), meaning that the two bytes δA and δB are equal (resp. different), *i.e.*, that $Class_{\delta A} = Class_{\delta B}$ (resp. $Class_{\delta A} \neq Class_{\delta B}$). The same reasoning may be done for ΔA and ΔB because $(\delta A \oplus \delta B = \delta C) \Leftrightarrow (\delta B \oplus \delta C = \delta A) \Leftrightarrow (\delta A \oplus \delta C = \delta B)$. Therefore, we redefine the `XOR` constraint as follows:

$$\begin{aligned}\texttt{XOR}(\Delta A, \Delta B, \Delta C) \Leftrightarrow \quad & \Delta A + \Delta B + \Delta C \neq 1 \\ & \wedge\ (Class_{\delta A} = Class_{\delta B}) \Leftrightarrow (\Delta C = 0) \\ & \wedge\ (Class_{\delta A} = Class_{\delta C}) \Leftrightarrow (\Delta B = 0) \\ & \wedge\ (Class_{\delta B} = Class_{\delta C}) \Leftrightarrow (\Delta A = 0)\end{aligned}$$

Propagation of MDS at Byte Level. The MDS property ensures that, for each column, the total number of bytes which are different from 0, before and after applying MC, is either equal to 0 or strictly greater than 4. This property also holds for any xor difference between two different columns of X and Y matrices. To propagate this property, for each pair of columns in X and Y matrices, we add a constraint on the number of bytes of these columns that are equal (*i.e.*, that have the same $Class$ values).

New constraints derived from KS. The `KeySchedule` mainly performs xor and S operations. As a consequence, each byte $\delta K_i[j][k]$ may be expressed as a xor between bytes of the original key difference δK_0, and bytes of δSK_{i-1} (which are differences of key bytes that have passed through S during the previous round). Hence, for each byte $\delta K_i[j][k]$, we precompute the set $V(i,j,k)$ such that $V(i,j,k)$ only contains bytes of δK_0 and δSK_{i-1} and $\delta K_i[j][k] = \bigoplus_{\delta A \in V(i,j,k)} \delta A$. For each set $V(i,j,k)$, we introduce a set variable $V_1(i,j,k)$ which is constrained to contain the subset of $V(i,j,k)$ corresponding to the Boolean variables equal to 1. We use these set variables to infer that two differential key bytes that have the same V_1 set are equal. Also, if $V_1(i,j,k)$ is empty (resp. contains one or two elements), we infer that $\Delta K_i[j][k]$ is equal to 0 (resp. a variable, or a xor between 2 variables).

4 Decomposition of Step 1 into two sub-steps

Our cryptanalytic problem must be solved for three key lengths $l \in \{128, 192, 256\}$, and for each key length, it must be solved for different round numbers r: when $l = 128$ (resp. 192 and 256), r ranges from 3 to 5 (resp. 10 and 14). Hence, we have 3, 8, and 12 instances for AES-128, AES-192, and AES-256, respectively. For each instance, the goal is to find all solutions for a given value of obj_{Step1}. In this paper, we consider only one value for obj_{Step1}, *i.e.*, the smallest value for which at least one of the Step1 solutions is byte-consistent as this is the most challenging problem (see [10]).

The CP model for Step 1 was implemented with the MiniZinc modeling language [14], and we compared four CP solvers: Gecode [9], Choco 4 [15], Picat_Sat [16] and Chuffed [5]. The best results were obtained with Picat_Sat and Chuffed. When $l = 128$, the best performing solver is Chuffed, which is able to solve all instances within an hour. On the other hand, for larger keys ($l \in \{192, 256\}$), Picat_Sat gradually becomes more efficient. However, for the most difficult instance of the problem ($l = 192$ and $r = 10$), neither Chuffed nor Picat_Sat can enumerate all solutions within a week.

However, empirical evaluation shows us that Picat_Sat finds the first solution rather quickly, but is not efficient at enumerating all possible solutions when there are too many solutions. Furthermore, when looking at solutions, we observe some kind of structure in the repartition of the Δ variables that are set to 1 and that pass through the SubBytes operator S. When $l = 128$, these variables are, for each round $i \in [0, r-1]$: $SX_i = \{\Delta X_i[j][k], j, k \in [0,3]\}$ and $SK_i = \{\Delta K_i[j][3], j \in [0,3]\}$[4].

To take advantage of the complementarity of Chuffed and Picat_Sat, we decompose Step 1 into two sub-problems. To this aim, we introduce two new integer variables for

[4] When $l \in \{192, 256\}$, the definition of SK_i is different.

each round $i \in [0, r-1]$, called $SumX_i$ and $SumK_i$, and we post the constraints: $SumX_i = \sum_{\Delta X_i[j][k] \in SX_i} \Delta X_i[j][k]$ and $SumK_i = \sum_{\Delta K_i[j][3] \in SK_i} \Delta K_i[j][3]$.

Step 1a. We use Picat_Sat to search for all possible consistent assignments for $SumX_i$ and $SumK_i$ variables: these assignments must satisfy all Step1 constraints. This search is done incrementally, by adding constraints each time a new solution is found. Let the values of $SumX_i$ and $SumK_i$ in the solution be x_i and k_i: we add the constraint $\bigvee_{i=0}^{r-1} (SumX_i \neq x_i \vee SumK_i \neq k_i)$ before searching for a new solution.

Step 1b. For each solution of Step 1a, we use Chuffed to search for all boolean solutions, given the values of $SumX_i$ and $SumK_i$ variables in the Step1a solution.

Experimental results. Figure 2 details the solving times for Step 1 without decomposing the problem, both with Chuffed and Picat_Sat, and when decomposing Step 1 into Step1a and Step1b. For very small instances, Chuffed alone is better for all versions of the AES. For larger instances Picat_Sat performs better than Chuffed and can solve almost all of them within the time limit of 48 hours. However, one instance is out of reach for Picat_Sat even after a week of computation. On the other hand, it is solved within 12 hours after decomposing the problem in Step 1a and Step 1b. We do not give the detailed results of each step: Step 1b is always very quickly solved by Chuffed (in 107 seconds for the hardest instance), and most of the solving time is spent for Step 1a by Picat_Sat. Table 1 gives the number of solutions of Steps 1a and 1b for each instance, showing that Step 1a has much less solutions than Step 1b. For example, for AES-192 with $r = 10$ rounds, Step 1a only has 5 solutions whereas Step 1b has 27548 solutions.

5 Conclusion

We have shown that we can solve all instances of our cryptanalysis problem in less than twelve hours when decomposing the problem into two steps and taking advantage of the complementarity of Chuffed and Picat_Sat. This is much faster than the Branch & Bound approach of [4], which needs several weeks to solve some of these instances. It is also faster and much less memory consuming than the approach of [8], that needs 60GB and 30 minutes on a 12-core computer to pre-compute the graph for AES-128.

New results for differential cryptanalysis. For $r = 4$ rounds, we have found a byte-consistent solution with $obj_{Step1} = 12$ and a probability equal to 2^{-79}. This solution is

	AES-128			AES-192								AES-256											
r	3	4	5	3	4	5	6	7	8	9	10	3	4	5	6	7	8	9	10	11	12	13	14
Step 1a	1	1	8	3	1	1	2	1	1	1	5	3	2	1	2	1	2	1	1	1	1	1	1
Step 1b	4	8	1113	15	4	2	6	4	8	240	27548	33	14	4	3	1	3	16	4	4	4	4	4

Table 1. Number of solutions of Steps 1a and 1b, for AES-128, 192 and 256, and for different number of rounds r.

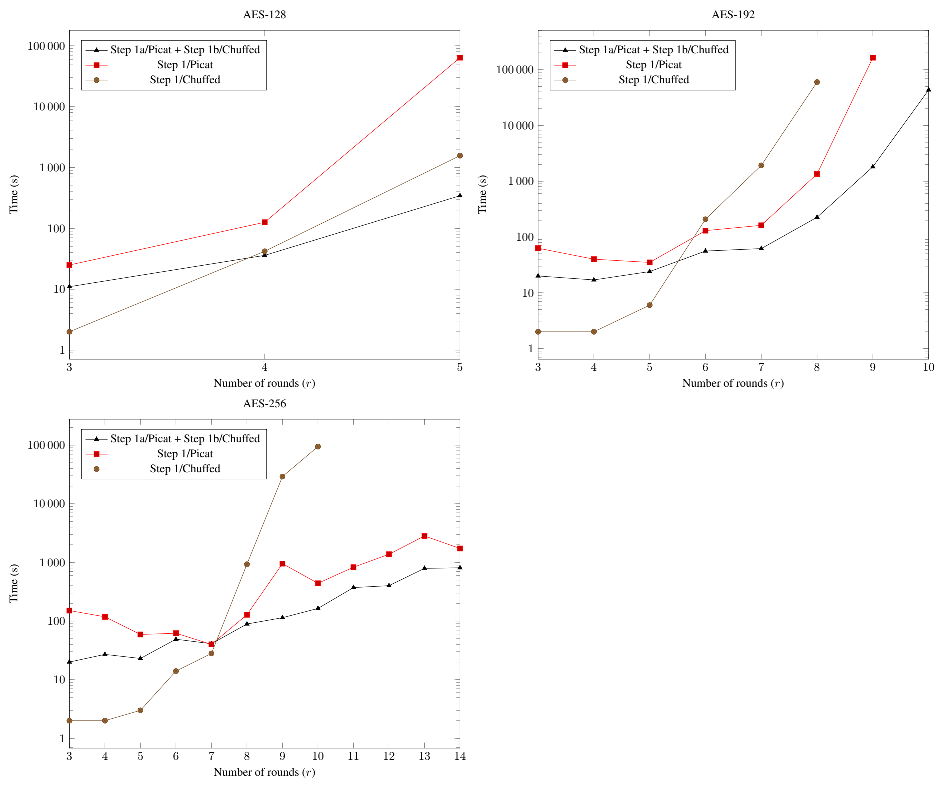


Fig. 2. Results for AES-128 (up left), AES-192 (up right), and AES-256 (down left). Each point gives the time needed to enumerate all solutions (no point if time exceeds 48 hours).

better than the solution claimed to be optimal in [4] and [8]: In these papers, the authors say that the best byte-consistent solution has $obj_{Step1} = 13$, and a probability equal to 2^{-81}. Furthermore, we have shown that the solution proposed in [4] for $l = 192$ and $r = 11$ is inconsistent. We have also found better solutions when $l = 256$, and we have computed the actual optimal solution for AES with $l = 256$. Its probability is 2^{-146} instead of 2^{-154} for the solution of [3] (see [10] for more details).

Further work. These cryptanalysis problems open new challenges for the CP community. In particular, these problems are not easy to model. Naive CP models (such as the first model we introduced in [13]) do not scale well. The introduction of equality constraints at the byte level (as proposed in [11]) and the decomposition of Step 1 into two sub-steps solved by two different solvers allow us to solve the hardest instances within twelve hours but this kind of modeling tricks are not straightforward to design. Hence, a challenge is to define new CP frameworks, dedicated to cryptanalysis problems, in order to ease the development of efficient CP models for these problems.

References

1. Biham, E.: New types of cryptoanalytic attacks using related keys (extended abstract). In: Advances in Cryptology - EUROCRYPT '93. Lecture Notes in Computer Science, vol. 765, pp. 398–409. Springer (1993)
2. Biham, E., Shamir, A.: Differential cryptoanalysis of feal and n-hash. In: Advances in Cryptology - EUROCRYPT '91. Lecture Notes in Computer Science, vol. 547, pp. 1–16. Springer (1991)
3. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and related-key attack on the full AES-256. In: Advances in Cryptology - CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 231–249. Springer (2009)
4. Biryukov, A., Nikolic, I.: Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. In: Advances in Cryptology - EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 322–344. Springer (2010)
5. Chu, G., Stuckey, P.J.: Chuffed solver description (2014), available at `http://www.minizinc.org/challenge2014/description\_chuffed.txt`
6. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag (2002)
7. FIPS 197: Advanced Encryption Standard. Federal Information Processing Standards Publication 197 (2001), u.S. Department of Commerce/N.I.S.T.
8. Fouque, P.A., Jean, J., Peyrin, T.: Structural evaluation of aes and chosen-key distinguisher of 9-round aes-128. In: Advances in Cryptology - CRYPTO 2013. Lecture Notes in Computer Science, vol. 8042, pp. 183–203. Springer (2013)
9. Gecode Team: Gecode: Generic constraint development environment (2006), available from `http://www.gecode.org`
10. Gérault, D., Lafourcade, P., Minier, M., Solnon, C.: Revisiting aes related-key differential attacks with constraint programming. Cryptology ePrint Archive, Report 2017/139 (2017), `http://eprint.iacr.org/2017/139`
11. Gerault, D., Minier, M., Solnon, C.: Constraint programming models for chosen key differential cryptanalysis. In: Principles and Practice of Constraint Programming - CP 2016. Lecture Notes in Computer Science, vol. 9892, pp. 584–601. Springer (2016)
12. Law, Y.C., Lee, J.H.M.: Global Constraints for Integer and Set Value Precedence, pp. 362–376. Springer Berlin Heidelberg (2004)
13. Minier, M., Solnon, C., Reboul, J.: Solving a Symmetric Key Cryptographic Problemwith Constraint Programming (Jul 2014), `https://hal.inria.fr/hal-01092574`, modRef 2014, Workshop of the CP 2014 Conference, September 2014, Lyon, France
14. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Principles and Practice of Constraint Programming - CP 2007. Lecture Notes in Computer Science, vol. 4741, pp. 529–543. Springer (2007)
15. Prudhomme, C., Fages, J.G.: An introduction to choco 3.0: an open source java constraint programming library. In: CP Workshop on "CP Solvers: Modeling, Applications, Integration, and Standardization" (2013)
16. Zhou, N.F., Kjellerstrand, H., Fruhman, J.: Constraint Solving and Planning with Picat. Springer (2015)

On Improving Run-time Checking in Dynamic Languages

(Presented in the Context of (C)LP Systems)

Nataliia Stulova*[1,2]

[1] IMDEA Software Institute, Madrid, Spain
nataliia.stulova@imdea.org
[2] School of Computer Science,
Technical University of Madrid(UPM),
Madrid, Spain

Abstract. In order to detect incorrect program behaviors, a number of approaches have been proposed, which include a combination of language-level constructs (procedure-level annotations such as assertions/contracts, gradual types, etc.) and associated tools (such as static code analyzers and run-time verification frameworks). However, it is often the case that these constructs and tools are not used to their full extent in practice due to a number of limitations such as excessive run-time overhead and/or limited expressiveness. The issue is especially prominent in the context of dynamic languages without an underlying strong typing system, such as Prolog. In our work we propose several practical solutions for minimizing the run-time overhead associated with assertion-based verification while keeping the correctness guarantees provided by run-time checks. We present the solutions in the context of the Ciao system, where a combination of an abstract interpretation-based static analyzer and run-time verification framework is available, although our proposals can be straightforwardly adapted to any other similar system.

Keywords: runtime verification, assertions, Prolog, logic programming

1 Introduction

Detecting incorrect program behaviors is an important part of the software development life cycle. It is also a complex and tedious one, in which dynamic languages bring special challenges.

A number of techniques have been proposed to aid in the process, among which we center our attention on the use of language-level constructs to describe expected program behavior, and of associated tools to compare actual program behavior against expectations, such as static code analyzers/verifiers and run-time verification frameworks. Approaches that fall into this category are the assertion-based frameworks used in (Constraint) Logic Programming [1,2,3,4],

* Supervised by José F. Morales[1] and Manuel V. Hermenegildo[1,2]

soft/gradual typing approaches in functional programming [5,6,7]and contract-based extensions in object-oriented programming [8,9,10]. These tools are aimed at detecting violations of the expected behavior or certifying the absence of any such violations, and often involve a certain degree of run-time testing, specially for non-trivial properties.

In practice, however, run-time overhead often remains impractically high, specially for complex properties, such as, for example, deep data structure tests. This reduces the attractiveness of run-time checking to programmers, which may allow sporadic checking of very simple conditions, but tend to turn off run-time checking for more complex properties. Some approaches even opt for limiting the expressiveness of the assertion language in order to reduce the overhead..

Our research objective is twofold:

– First, we aim for an expressive assertion language reflects the features of the related programming language, which both allows a programmer to write precise program specifications and does not impose a learning burden.
– At the same time, our goal is to efficiently checks such specifications, mitigating the associated run-time overhead as much, as possible without compromising the safety guarantees the checks provide.

While our work is general and system-independent, we present it for concreteness in the context of the Ciao run-time checking framework. The Ciao model [11,2] is well understood, and different aspects of it have been incorporated in popular (C)LP systems, such as Ciao, SWI, and XSB [12,13,14].

2 Current Research Results

2.1 Supporting Higher-Order Properties

Higher-order programming is a widely adopted programming style that adds flexibility to the software development process. Within the (Constraint) Logic Programming ((C)LP) paradigm, Prolog has included higher-order constructs since the early days, and there have been many other proposals for combining the first-order kernel of (C)LP with different higher-order constructs, e.g., [15,16,17]).Many of these proposals are currently in use in different (C)LP systems and have been found very useful in programming practice, inheriting the well-known benefits of code reuse (templates), elegance, clarity, and modularization.

When higher-order constructs are introduced in the language it becomes necessary to describe properties of arguments of predicates/procedures that are themselves also predicates/procedures. While the combination of contracts and higher-order has received some attention in functional programming [18,19], within (C)LP the combination of higher-order with the previously mentioned assertion-based approaches has received comparatively little attention to date. Current Prolog systems simply use basic atomic types (i.e., stating simply that the argument is a `pred`, `callable`, etc.) to describe predicate-bearing variables (see 1). Other approaches [20] are oriented instead to meta programming, de-

```
:- pred min(X,Y,Cmp,Min) : callable(Cmp).

min(X,Y,P,Min) :- P(R,X,Y), R <= 0, Min = X.
min(X,Y,_,Y  ).

less( 0,A,A).             lt('=',A,A).
less(-1,A,B) :- A < B.    lt('<',A,B) :- A < B.
less( 1,_,_).             lt('>',_,_).

test_min :- min(4,2,lt,2). % lt/3 is passed, but less/3 is expected
```

Fig. 1. A simple program with a higher-order predicate `min/4` that accepts a custom comparator predicate.

scribing meta-types but there is no notion of directionality (modes), and only a single pattern is allowed per predicate.

Our proposal [21] contributes towards filling this gap between higher-order (C)LP programs and assertion-based extensions for error detection and program validation. Our starting point is the Ciao assertion model, which we have enhanced with a new class of properties, "predicate properties" (*predprops* in short), and proposed a syntax and semantics for them. These new properties can be used in assertions for higher-order predicates to describe the properties of the higher-order arguments. An example of a predprop is provided in Fig. 2, where an *anonymous* assertion (note the variable symbol `Cmp` in place of a predicate symbol) is used to describe a comparison predicate, that is not known at compilation time. By reusing the original assertion language syntax to describe call and success conditions of predicate-bearing arguments we allow both for better integration of these new constructs into the verification framework and at the same time lessening the burden on a programmer, who needs to provide such annotations.

```
:- comparator(Cmp) {
  :- pred Cmp(Res,M,N) : (num(M), num(N)) => between(-1,1,Res).
}.

:- pred min(X,Y,Cmp,Min) : comparator(Cmp).
```

Fig. 2. A *predprop* `comparator` example and an *anonymous* assertion in its definition.

Our predprop properties specify conditions for predicates that are independent of the usage context. This corresponds in functional programming to the notion of *tight* contract satisfaction, and it contrasts with alternative approaches such as *loose* contract satisfaction. In the latter, contracts are attached to higher-order arguments by implicit function wrappers. The scope of checking is local to the function evaluation. Although this is a reasonable and pragmatic solution, we believe that our approach is more general and more amenable to combination with static verification techniques. For example, avoiding wrappers allows us to remove checks (e.g., by static analysis) without altering the program semantics. Moreover, our approach can easily support *loose* contract satisfaction, since it is straightforward in our framework to optionally include wrappers as special predprops.

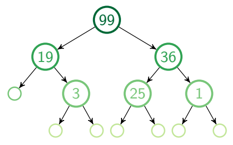
2.2 Trading Memory for Speed

While having become an integral part of software development process, run-time testing can generally incur high penalty in execution time and/or space over the standard program execution without tests. A number of techniques have been proposed to date to reduce this overhead, including simplifying the checks at compile time via static analysis [22,11] or reducing the frequency of checking, including for example testing only at a reduced number of points [4,23]. Our proposal of [24] describes an approach to run-time testing that is efficient while being minimally obtrusive and remaining exhaustive. It is based on the use of memoization to cache intermediate results of check evaluation in order to avoid repeated checking of previously verified properties over the same data structure.

Memoization has of course a long tradition in (C)LP in uses such as tabling resolution [25,26].Memoization has also been used in program analysis [27,28], where tabling resolution is performed using abstract values. However, in tabling and analysis what is tabled are call-success patterns and in our case the aim is to cache the results of test execution.

We concentrate our attention on the checks of the *regular types* [29], a useful subset of properties that are often used in assertions. An example in Fig. 3 shows a binary tree (left) and its possible implementation as a regular type together with an instance of that type describing the tree (right).

Our approach is based on an observations that run-time checks of regular types are monotonous instantiation checks, e.g. terms only become more and more instantiated with every consequent state that the program enters. We extend the Ciao run-time checking framework with a common cache, accessible to run-time checks, that stores tuples (x, t), where x is a term address and t is an identifier of a regular type. After the initial check on the term is performed, the corresponding tuple is added to the cache and for all the consequent checks on the term the check substituted by the (computationally cheaper) cache lookup operation.



```
tree(e).
tree(t(L,_,R)) :- tree(L), tree(R).

t(                ,99,                        )
  t( ,19,        )    t(         ,36,        )
    e    t( ,3, )       t( ,25, )    t( ,1, )
           e   e          e    e       e   e

```

Fig. 3. A minimalistic tree data structure implementation as a regular type `tree/1`

While studying the resulting performance of the enhanced verification framework we have observed the using a cache does not necessarily lead to a significant checks cost reduction. An important issue that must be taken into account is the character of cache rewrites: as the terms grow in size cache collisions happen more often, which leads to elements eviction. This in turn cancels out the advantage that using a cache gives: the ability to just lookup the regular type of some term without actually performing the check of it. However, this effect can be remedied by limiting the depth of terms that are stored in the cache (e.g., not caching terms with depth more than some n).

The idea of using memoization techniques to speed up checks has attracted some attention recently [30]. Their work (developed independently from ours) is based on adding fields to data structures to store the properties that have been checked already for such structures. In contrast, our approach has the advantage of not requiring any modifications to data structure representation, or to the checking code, program, or core run-time system.

2.3 Intertwining Compile- and Run-time Checks

A complementary approach to run-time overhead reduction consists in using static analysis to minimize the number and cost of the run-time checks that need to be placed in the program to detect incorrect program behaviors. This idea was pioneered by the Ciao system where a number of (abstract interpretation-based) static analyses are combined in order to verify assertions to the largest extent possible at compile time, and for simplifying and reducing the number of remaining properties that that need to be introduced in the program as run-time checks. However, while there has been evidence from use, there has been little systematic experimental work presented to date measuring the actual impact of analysis on reducing run-time checking overhead.

In our work [31] we generalize the existing practices as four *assertion checking modes*, each of which represents a trade-off between code annotation depth, execution time slowdown, and program behavior safety guarantees:

- *Unsafe*: no run-time checks are generated from program assertions, program execution is fast but incorrect program behaviors may stay undetected; the run-time overhead is nonexistent.
- *Client-Safe*: run-time checks are generated from the assertions of the program's interface (providing behavior guarantees of it for the clients), yet internal program assertions remain unchecked; the run-time overhead is taken as a minimal unavoidable one.
- *Safe-RT*: run-time checks are generated from all program assertions; this mode of checking is characterized with the strongest behavior safety guarantees yet is at the same time associated with highest check costs;
- *Safe-CT-RT*: a variation of *Safe-RT* checking mode with an additional static analysis phase, during which some of program assertions may be proven to always hold and generating run-time checks from them can be omitted; depending on the kind of analysis and the program the overhead generated

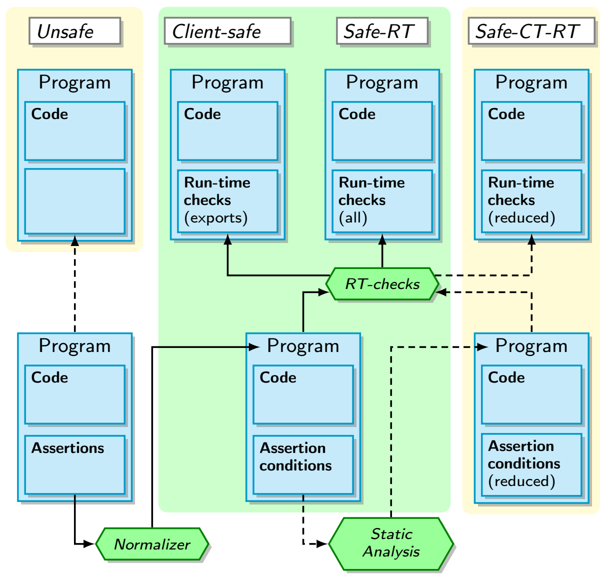


Fig. 4. Source transformation differences per checking mode.

by the checks from remaining assertions may be closer to that of *Client-Safe Safe-RT* modes.

We also define a transformation-based approach in order to implement each one of these modes (see Fig. 4).

We then concentrate on the reduction of the number of run-time tests via (abstract interpretation-based) program analysis. To this end we propose a technique that enhances analysis precision by taking into account that any assertions that cannot be proved statically will be the subject of run-time testing. In practice it means that it is safe to make the two following assumptions for any predicate that has an assertion specifying properties, that should hold on calls and success:

– the calls conditions hold after the analysis has entered the predicate definition, since either the checks for these calls conditions have already succeeded or the program has exited with error;
– the relevant success conditions hold after the predicate has exited (since, again, at this point either these success conditions have already succeeded or the program has exited with error)

2.4 Benefiting from Information Hiding

While dynamic languages offer for programmers great flexibility in term creation and manipulation, for the very same reason the need in exhaustive run-time checks arises in order to guarantee the data manipulation safety and correctness. Reusable libraries, i.e., library modules that are pre-compiled independently of the client, pose special challenges in this context. The key issue here is that there is virtually no control on how and where valid terms can be created, and thus it is quite common in the client-library interaction that any of these modules can create any data shape and pass it. As a consequence, (often expensive) run-time checks on the module boundaries become the necessary evil. In our work [32] we propose a possible solution to overhead reduction for run-time checks on module boundaries based on the *information hiding* principle (which is adopted in many other systems in form of encapsulation or opaque data types).

Currently, most mature Prolog implementations adopt some flavor of a module system, *predicate-based* in SWI [33], SICStus [34], YAP [35], ECLiPSe [36], and *atom-based* in XSB [13]. The difference between the two systems is the strictness of the term visibility rules: in an atom-based system local to the module terms are not visible outside it if they are not a part of the module interface. The Ciao approach [37] has until now been closer to a predicate-based module system.

We propose an extension of the predicate-based module system, that allows to specify only a subset of module terms as local (*hidden*) ones. Our argument is that in this setup we have the guarantee of the homogeneity of the structure of the module terms, as only one module is allowed to construct/deconstruct some particular data shapes. With this there is no need to perform thorough checks of properties that verify the correctness of the data term structure at the module boundaries. Instead, it would suffice to perform checks of *shallow* versions of data shape properties: the weakened forms of the original properties that are semantically equivalent to them in the context of the possible program executions. These versions typically require asymptotically fever execution steps and in some cases of the calls across module boundaries allow to achieve constant run-time overhead.

3 Conclusions

Specification-based runtime verification approach has attracted significant interest in recent decades, both from academia and industry. As it is the case with any other open problem, finding an approach that would suit each and every system is not feasible, thus opting for tailored partial solutions is inevitable and more practical.

In our work we have concentrated on the peculiarities of the run-time verification task in the context of dynamic languages and their use in (C)LP systems. We have proposed an enhancement for the Ciao assertion language that allows to capture the concrete execution contexts of higher-order terms and thus adapts the verification framework to this new use case. We have also proposed several solutions for reducing the overhead, associated with run-time checks, that treat the issue from several different angles.

While for concreteness of presentation our work was carried out within the Ciao language and combined static/dynamic verification framework, our results are general and system-independent. We believe they can be straightforwardly transferred to the contexts of other declarative languages, and given the recent advances in the Horn clause-based verification we feel that our results could also be adapted to the broader specter of languages (e.g. imperative ones).

References

1. W. Drabent, S. Nadjm-Tehrani, and J. Małuszyński. The Use of Assertions in Algorithmic Debugging. In *Intl. Conf. on Fifth Generation Computer Systems*, pages 573–581, 1988.
2. G. Puebla, F. Bueno, and M. Hermenegildo. Combined Static and Dynamic Assertion-Based Debugging of Constraint Logic Programs. In *Logic-based Program Synthesis and Transformation (LOPSTR'99)*, number 1817 in LNCS, pages 273–292. Springer-Verlag, March 2000.
3. Claude Laï. Assertions with Constraints for CLP Debugging. In Pierre Deransart, Manuel V. Hermenegildo, and Jan Maluszynski, editors, *Analysis and Visualization Tools for Constraint Programming*, volume 1870 of *Lecture Notes in Computer Science*, pages 109–120. Springer, 2000.
4. E. Mera, P. López-García, and M. Hermenegildo. Integrating Software Testing and Run-Time Checking in an Assertion Verification Framework. In *25th Int'l. Conference on Logic Programming (ICLP'09)*, volume 5649 of *LNCS*, pages 281–295. Springer-Verlag, July 2009.
5. Robert Cartwright and Mike Fagan. Soft Typing. In *PLDI'91*, pages 278–292. SIGPLAN, ACM, 1991.
6. Sam Tobin-Hochstadt and Matthias Felleisen. The Design and Implementation of Typed Scheme. In *POPL*, pages 395–406. ACM, 2008.
7. Asumu Takikawa, Daniel Feltey, Earl Dean, Matthew Flatt, Robert Bruce Findler, Sam Tobin-Hochstadt, and Matthias Felleisen. Towards practical gradual typing. In John Tang Boyland, editor, *29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic*, volume 37 of *LIPIcs*, pages 4–27. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.
8. Leslie Lamport and Lawrence C. Paulson. Should your specification language be typed? *ACM Transactions on Programming Languages and Systems*, 21(3):502–526, May 1999.
9. Gary T. Leavens, K. Rustan M. Leino, and Peter Müller. Specification and verification challenges for sequential object-oriented programs. *Formal Asp. Comput.*, 19(2):159–189, 2007.
10. Manuel Fähndrich and Francesco Logozzo. Static contract checking with abstract interpretation. In *Proceedings of the 2010 International Conference on Formal Verification of Object-oriented Software*, volume 6528 of *FoVeOOS'10*, pages 10–30, Berlin, Heidelberg, 2011. Springer-Verlag.
11. M. Hermenegildo, G. Puebla, and F. Bueno. Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging. In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.

12. M. V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, J.F. Morales, and G. Puebla. An Overview of Ciao and its Design Philosophy. *Theory and Practice of Logic Programming*, 12(1–2):219–252, January 2012. http://arxiv.org/abs/1102.5497.
13. Terrance Swift and David Scott Warren. XSB: Extending Prolog with Tabled Logic Programming. *TPLP*, 12(1-2):157–187, 2012.
14. Edison Mera and Jan Wielemaker. Porting and refactoring Prolog programs: the PROSYN case study. *TPLP*, 13(4-5-Online-Supplement), 2013.
15. D.H.D. Warren. Higher-order extensions to Prolog: are they needed? In J.E. Hayes, Donald Michie, and Y-H. Pao, editors, *Machine Intelligence 10*, pages 441–454. Ellis Horwood Ltd., Chicester, England, 1982.
16. Gopalan Nadathur and Dale Miller. Higher–Order Logic Programming. In D. Gabbay, C. Hogger, and A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5. Oxford University Press, 1998.
17. D. Cabeza, M. Hermenegildo, and J. Lipton. Hiord: A Type-Free Higher-Order Logic Programming Language with Predicate Abstraction. In *Ninth Asian Computing Science Conference (ASIAN'04)*, number 3321 in LNCS, pages 93–108. Springer-Verlag, December 2004.
18. Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In Mitchell Wand and Simon L. Peyton Jones, editors, *ICFP*, pages 48–59. ACM, 2002.
19. Christos Dimoulas and Matthias Felleisen. On contract satisfaction in a higher-order world. *ACM Trans. Program. Lang. Syst.*, 33(5):16, 2011.
20. C. Beierle, R. Kloos, and G. Meyer. A Pragmatic Type Concept for Prolog Supporting Polymorphism, Subtyping, and Meta-Programming. In *Proc. of the ICLP'99 Workshop on Verification of Logic Programs, Las Cruces*, Electronic Notes in Theoretical Computer Science, volume 30, issue 1. Elsevier, 2000.
21. N. Stulova, J. F. Morales, and M. V. Hermenegildo. Assertion-based Debugging of Higher-Order (C)LP Programs. In *16th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'14)*. ACM Press, September 2014.
22. F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla. On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs. In *Proc. of the 3rd. Int'l WS on Automated Debugging–AADEBUG*, pages 155–170. U. Linköping Press, May 1997.
23. E. Mera, T. Trigo, P. López-García, and M. Hermenegildo. Profiling for Run-Time Checking of Computational Properties and Performance Debugging. In *Practical Aspects of Declarative Languages (PADL'11)*, volume 6539 of *Lecture Notes in Computer Science*, pages 38–53. Springer-Verlag, January 2011.
24. N. Stulova, J. F. Morales, and M. V. Hermenegildo. Practical Run-time Checking via Unobtrusive Property Caching. *Theory and Practice of Logic Programming, 31st Int'l. Conference on Logic Programming (ICLP'15) Special Issue*, 15(04-05):726–741, September 2015.
25. H. Tamaki and M. Sato. OLD Resol. with Tabulation. In *ICLP*, pages 84–98. LNCS, 1986.
26. S. Dietrich. *Extension Tables for Recursive Query Evaluation.* PhD thesis, Departament of Computer Science, State University of New York, 1987.
27. R. Warren, M. Hermenegildo, and S. K. Debray. On the Practicality of Global Flow Analysis of Logic Programs. In *Fifth International Conference and Symposium on Logic Programming*, pages 684–699. MIT Press, August 1988.

28. K. Muthukumar and M. Hermenegildo. Compile-time Derivation of Variable Dependency Using Abstract Interpretation. *Journal of Logic Programming*, 13(2/3):315–347, July 1992.
29. P.W. Dart and J. Zobel. Efficient Run-Time Type Checking of Typed Logic Programs. *Journal of Logic Programming*, 14:31–69, October 1992.
30. Emmanouil Koukoutos and Viktor Kuncak. Checking Data Structure Properties Orders of Magnitude Faster. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification*, volume 8734 of *Lecture Notes in Computer Science*, pages 263–268. Springer International Publishing, 2014.
31. N. Stulova, J. F. Morales, and M. V. Hermenegildo. Reducing the Overhead of Assertion Run-time Checks via static analysis. In *18th Int'l. ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'16)*, pages 90–103. ACM Press, September 2016.
32. N. Stulova, J. F. Morales, and M. V. Hermenegildo. Term Hiding and its Impact on Run-time Check Simplification (Extended Abstract). In *Proceedings of the Technical Communications of the 33rd International Conference on Logic Programming (ICLP 2017), Melbourne, Australia, August 28 - September 1, 2017.* OASIcs, August 2017.
33. J. Wielemaker. *The SWI-Prolog User's Manual 5.9.9*, 2010.
34. Swedish Institute for Computer Science, PO Box 1263, S-164 28 Kista, Sweden. *SICStus Prolog User's Manual*, 4.1.1 edition, December 2009.
35. Vítor Santos Costa, Luís Damas, and Ricardo Rocha. The YAP Prolog System. *Theory and Practice of Logic Programming*, 2011. http://arxiv.org/abs/1102.3896v1.
36. Cisco Systems. *ECLIPSE User Manual*, 2006.
37. D. Cabeza and M. Hermenegildo. A New Module System for Prolog. In *International Conference CL 2000*, volume 1861 of *LNAI*, pages 131–148. Springer-Verlag, July 2000.

Efficiently Computing Weighted Egalitarian Solutions For Multi-Objective Constraint Optimization Problems

Emir Demirović[1] and Nicolas Schwind[2]

[1] Student
Vienna University of Technology, Vienna, Austria
demirovic@dbai.tuwien.ac.at,
[2] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
nicolas-schwind@aist.go.jp

Abstract. Solving a multi-objective constraint optimization problem (MO-COP) typically requires computing an exponentially large number of Pareto optimal solutions. This is particularly problematic in a product configuration context where an end user is asked to select her preferred Pareto optimal solution from such a large set. The notion of *representative solutions* has been recently proposed to handle the lack of decisiveness. Given an integer k, the problem consists in extracting a restricted set of k Pareto optimal solutions whose objective vectors are "close enough" to any other Pareto optimal solution. Although the idea of representative solutions appears to be a desirable concept in MO-COPs, the problem of finding k representative solutions is computationally hard (Σ_2^P-hard), which makes the approach inapplicable to real-world instances. However, the efficiency issue can be addressed by approximating the representative solutions using weighted egalitarian solutions (ω-solutions). In this paper, we address the efficiency issue by providing two techniques for computing the ω-solutions, which can be used to approximate the set of representative solutions. This provides us with a reasonable trade-off between representativity and efficiency, allowing one to make representative solutions a practical answer to the decisiveness issue for large MO-COP problems.

Keywords: multi-objective constraint optimization, representative solutions, weighted egalitarian solutions

1 Introduction

This paper deals with multi-objective constraint optimization problems (MO-COPs), in which several objective functions are to be minimized or maximized simultaneously. Usually, in such problems, there is no solution that is optimal with respect to all objectives. Therefore, trade-offs between objectives must be made. However, such trade-offs are domain-dependent or based on some underlying user's preferences, so usually no *a priori* decision making is possible. Thus, solutions are typically preferred based on the sole notion of Pareto optimality, that is, a solution is *Pareto optimal* if it is not dominated by any other solution with respect to all objectives.

Two main problems arise from these considerations: the lack of efficiency and decisiveness: in the general case, an MO-COP is associated with an exponentially large set of Pareto optimal solutions. This is problematic in a product configuration context, where an end user is asked to select her preferred solution from the Pareto optimal set. Indeed, according to psychological studies not more than seven alternatives can be pairwise compared by a user [7], and studies in recommender systems conclude that it is best to provide only three choices [17].

A recent approach for MO-COPs has been proposed to address the decisiveness issue [15]. It consists in computing a single Pareto optimal solution, called *weighted egalitarian solution*. Intuitively, this approach consists in "targeting" a specific area of the objective vector space "close" to the line defined by the weighted vector, which represents preferences among different objectives. However, expressing quantitative relative importance between preferences may be cumbersome for the end user making these weighted vectors often unavailable. Another approach is based on *diverse solutions* [5, 13]. The goal is to compute a set of solutions that are pairwise distant. Though diverse solutions are suitable for problems such as software testing where diverse inputs are sought for testing purpose, they are arguably not suitable for MO-COPs as, e.g. they do not provide an overview of the shape of the Pareto optimal set of solutions. Recently, Schwind et al. [14] introduced a filtering function which is more suitable for MO-COPs called *representativity*, by taking their inspiration from the problem of locating facilities in the field of discrete location theory [10, 4, 6]. Given an MO-COP and an integer k, the idea is to compute k solutions whose covering radius is minimized in the space of Pareto optimal solutions. Therefore, one has the guarantee that every Pareto optimal solution of the MO-COP is close enough to one of the k representative solutions.

However, the issue of computational efficiency has remained unresolved by considering representative solutions. Indeed, computing k representative solutions is a Σ_2^{P}-hard problem even if $k = 1$: it is one of the hardest problems lying on the second level of the polynomial hierarchy [14]. An approximation approach based on the notion of *weighted egalitarian solutions* has been introduced in [14]. The algorithm proposed in [14] scales well with the number of objectives, but not with respect to the problem size (e.g. number of variables). Thus, it cannot be used for practical applications.

In this paper, we provide two new techniques for computing weighted egalitarian solutions. The first encodes the sorted cost vector through auxiliary variables and then uses lexicographical optimization, while the second iteratively builds partial candidate solutions. To the best of our knowledge, this is the first time algorithms for computing weighted egalitarian solutions have been sufficiently developed to efficiently tackle large-size MO-COP instances, offering direct improvements over the algorithm in [14]. In addition, we proved that computing ω-solutions to approximate the set of representative solutions is NP-hard, from which one can expect a substantial efficiency gain when compared to computing the k representative solutions (Σ_2^{P}-hard).

The rest of the paper is organized as follows. The next section provides the necessary preliminaries on MO-COPs, computational complexity, representative solutions, and weighted egalitarian solutions. In Section 3 we present our main contributions which include two techniques for computing weighted egalitarian solutions and prove the complexity shift. We conclude in Section 5.

2 Preliminaries

2.1 MO-COPs

Following the notation used in [14, 15], a multi-objective constraint optimization problem (MO-COP) is defined as a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}^{hard}, \mathcal{C}^{soft} \rangle$, where: $\mathcal{X} = \{x_1, \ldots, x_n\}$ are variables; $\mathcal{D} = \{D_1, \ldots, D_n\}$ are domains; $\mathcal{C}^{hard}$ are *hard* constraints, i.e., each constraint $C \in \mathcal{C}^{hard}$ is a subset of $D_{i_1} \times \cdots \times D_{i_C}$ for some $\{D_{i_1}, \ldots, D_{i_C}\} \subseteq \mathcal{D}$; and $\mathcal{C}^{soft}$ are *soft*, polyadic constraints, i.e., each constraint $C' \in \mathcal{C}^{soft}$ is a mapping from $D_{i_1} \times \cdots \times D_{i_{C'}}$ to $\mathbb{N}^m$, for some $\{D_{i_1}, \ldots, D_{i_{C'}}\} \subseteq \mathcal{D}$. Here, m represents the number of objectives of the MO-COP. Each constraint from $\mathcal{C}^{hard} \cup \mathcal{C}^{soft}$ involves a set of variables called its *scope*. For simplicity, when not explicitly stated we consider given an MO-COP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}^{hard}, \mathcal{C}^{soft} \rangle$ with m objectives. An *assignment* A associates each $x_i \in X$ with a value from D_i; A is a *solution* if it satisfies all hard constraints, i.e., there is no $C \in \mathcal{C}^{hard}$ such that $(A(x_{i_1}), \ldots, A(x_{i_C})) \in C$, where $\{x_{i_1}, \ldots, x_{i_C}\}$ is the scope of C. Given a vector U, we denote by U^i or $U(i)$ its i^{th} component. The *cost vector* of A is the vector $V(A)$ defined for each $t \in \{1, \ldots, m\}$ as

$$V(A)^t = \sum_{C \in \mathcal{C}^{soft}} C(A(x_{i_1}), \ldots, A(x_{i_C}))(t)$$

where for each C, $\{x_{i_1}, \ldots, x_{i_C}\}$ is the scope of C.

A *Pareto optimal solution*[3] is a solution S for which there is no solution S' such that for all $i \in \{1, \ldots, m\}$ and for some $j \in \{1, \ldots, m\}$, $V(S')^i \leq V(S)^i$ and $V(S')^j < V(S)^j$. $\mathcal{S}_{Par}$ denotes the set of all Pareto optimal solutions.

2.2 Representative solutions

Representative solutions [14] are fixed-sized subsets $\mathcal{S}_*$ of Pareto optimal solutions for which the maximum "distance" between any Pareto optimal solution from $\mathcal{S}_{Par}$ and one of the solutions from $\mathcal{S}_*$ is minimized. That is, given an integer k corresponding to the desired size of $\mathcal{S}_*$, computing $\mathcal{S}_*$ consists in minimizing the function $\Omega : \{\mathcal{S} \in 2^{\mathcal{S}_{Par}} \mid |\mathcal{S}| = k\} \mapsto \mathbb{N}$ defined for every set $\mathcal{S} \subseteq \mathcal{S}_{Par}(P)$ as:

$$\Omega(\mathcal{S}) = \max_{A \in \mathcal{S}_{Par}(P)} \min_{A' \in \mathcal{S}} d(V(A), V(A')),$$

[3] Without loss of generality, we consider here a minimization setting.

where d is a distance between cost vectors. For simplicity, we assume that d corresponds to the Manhattan distance defined for all $V_1, V_2 \in \mathbb{N}^m$ as $d(V_1, V_2) = \sum_{i=1}^{m} |V_1^i - V_2^i|$. The *radius* of $\mathcal{S}$ is denoted $\Omega(\mathcal{S})$. Therefore, the goal is to compute a set $\mathcal{S}_*$of k Pareto optimal solutions whose radius $\Omega(\mathcal{S}_*)$ is minimal.

We assume that the reader is familiar with the complexity class NP (see [12] for more details). Higher complexity classes are defined using oracles. In particular, $\Sigma_2^{\mathsf{P}} = \mathsf{NP}^{\mathsf{NP}}$ corresponds to the class of decision problems that are solved in non-deterministic polynomial time by deterministic Turing machines using an oracle for NP in polynomial time.

Computing k representative solutions is a hard problem, even for $k = 1$:

Theorem 1 ([14]). *Given an MO-COP P and two integers α, k, the problem of deciding whether there exists $\mathcal{S} \subseteq \mathcal{S}_{Par}(P)$ such that $|\mathcal{S}| = k$ and $\Omega(\mathcal{S}) \leq \alpha$ is Σ_2^{P}-hard, even when $k = 1$.*

2.3 Weighted Egalitarian Solutions

A *weight vector* [16] is a vector $\omega \in]0, 1]^m$ such that $\sum_{k=1}^{m} \omega^k = 1$. Given a weight vector ω, an *ω-weighted egalitarian solution* [15] (named "ω-solution" in the following) is a solution A for which there is no solution A' such that $V(A)_{<,\omega} \leq_{lex} V(A')_{<,\omega}$, where $\leq_{lex}$ is the lexicographic ordering induced by the natural ordering, and $V(A)_{<,\omega}$ is the vector obtained by sorting in a non-increasing order the vector $(\omega^1.V(A)^1, \ldots, \omega^m.V(A)^m)$ (see [15] for more details). ω-solutions exhibit a number of interesting properties: (i) they are Pareto optimal; (ii) they correspond to solutions that are "close" to the line whose equation is specified by ω in the solution space; (iii) for any Pareto optimal solution A, there exists a weight vector ω such that A is an ω-solution. Hence, given a weight vector, one can use this concept to "target" a specific area. Note that ω-solutions differ from solutions minimizing a weighted sum of the objectives. In particular, points (ii) and (iii) do not hold for "weighted sum" solutions [15].

The concept of ω-solution was initially introduced in [15] to select the "best" Pareto optimal solution according to some user's preferences among the objectives. This requires ω to be defined in accordance with these preferences, which are not necessarily available or not expressed quantitatively. However, one can take advantage of the concept of ω-solutions to approximate a set of k representative solutions as follows [14]. First, one starts with a set W_* of k weight vectors whose radius is minimal among all possible sets of k weight vectors, that is, W_* is a set that is "representative" of the space of all weight vectors. We assume that such a set W_* is computed "offline" i.e., it does not depend on the MO-COP to be solved but only on m and k. To do so, we finely discretize the space of weight vectors and then use an exact algorithm [14] to compute the representative set W_*. Second, we compute an ω-solution for each weight vector $\omega \in W_*$. Then the challenge lies in the efficiency of the procedure to compute a particular ω-solution for a given weight vector ω; this is what we address in the next section.

We refer the reader to [15, 14] for more details, examples, and illustrations on MO-COPs, representative solutions and ω-solutions.

3 Computing ω-solutions

We stress that the process of computing ω-solutions to approximate a set of representative solutions is NP-hard (cf. Theorem 1) compared to Σ_2^{P}-hard, from which one can expect a substantial efficiency gain when compared to computing the k representative solutions (Σ_2^{P}-hard). This is given in the following proposition:

Proposition 1. *Given an MO-COP P, a weight vector ω and, a vector $U \in \mathbb{N}^m$ the problem of deciding whether there exists an ω-solution S such that for all $i \in \{1, \ldots, m\}$, $V(S)^i \leq U^i$, is* NP*-complete.*

We consider computing the ω-solution where all weights are equal ($\omega_i = 1/m$) and refer to these as *egalitarian solutions*. Weighted solutions can be obtained by scaling the objective as preprocessing. We introduce *objective variables* $e_i = V(A)^i$ for an MO-COP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}^{hard}, \mathcal{C}^{soft} \rangle$ with m objectives. We developed two techniques: an encoding-based approach and an iterative algorithm. The former encodes variables that capture values of the sorted cost vector and then uses lexicographical optimization, while the latter incrementally builds egalitarian solutions.

Encoding-based technique. Let variables $\{l_1, \ldots, l_m\}$ be such that given a solution A, each l_i takes the i^{th} largest value among $(A(e_1), \ldots, A(e_m))$. To do so, we make use of a set of auxiliary Boolean variables $\{S_{(i,j)} : i, j \in [1, ...m]\}$:

$$\sum_j S_{(i,j)} = 1 \qquad \forall i \in \{1, \ldots, m\} \tag{1}$$

$$\sum_i S_{(i,j)} = 1 \qquad \forall j \in \{1, \ldots, m\} \tag{2}$$

$$(S_{(i,j)} = 1) \Rightarrow (l_i = e_j) \qquad \forall i, j \in \{1, \ldots, m\} \tag{3}$$

$$l_i \geq l_{i+1} \qquad \forall i \in \{1, \ldots, m-1\} \tag{4}$$

The above equations guarantee that for every solution A, $A(S_{(i,j)}) = 1$ if the objective variable e_j takes the i^{th} highest value among $(A(e_1), \ldots, A(e_m))$, and $A(S_{(i,j)}) = 0$ otherwise. Equation 1 and 2 states that every objective is assigned a unique ordering number. Equations 3 and 4 ensure that the appropriate assignments take place. We note that the addition of the previously discussed variables and constraints to the MO-COP does not affect its set of solutions, and the soft constraints from $\mathcal{C}^{soft}$ remain unchanged. Therefore, we extended the initial MO-COP with variables l_i which encode the sorted values of the cost vector components $V(A)^i$. To obtain the egalitarian solution any lexicographical optimization algorithm can be used, first minimizing l_0, then l_1, and so on.

Iterative algorithm. The algorithm is summarized in Algorithm 1. It starts with a set of partial candidate assignments for the objective variables, initially containing the empty assignment (Line 3). In each iteration, new partial assignments are computed for each set in the set of candidates (Lines 4-11). These newly computed assignments are identical to the assignments they are based on, but with the addition of one more objective variable being assigned (Line 10). Dominated assignments with respect to $\leq_{lex}$ are removed (Line 12). Therefore, after the *i-th* iteration, we obtain the set of non-dominated candidate partial assignments that have *(i+1)* objective variables assigned.

To compute new candidate assignments based on a partial assignment pa (Line 8) we define MO-COP$_{(e_i,pa)}$ which extends MO-COP with the equations:

$$pa_{ex}(e_k) = pa(e_k) \qquad \forall e_k \in E_a(pa) \tag{5}$$

$$e_k \geq e_i \geq e_j \qquad \forall e_j \in E_{una}(pa), e_k \in E_a(pa) \tag{6}$$

where $E_a(pa)$ and $E_{una}(pa)$ are the sets of assigned and unassigned objective variables in the partial assignment pa. To compute new partial assignments, for each $e_i \in E_{una}$, find the solution pa_{ex} which minimizes e_i in MO-COP$_{(e_i,pa)}$. If such a solution exists, a new candidate assignment pa_{new} is generated which has the same assignments for the objective variables as pa_{ex} with the exception $pa_{new}(e_i) = pa_{ex}(e_i)$. Therefore, pa_{new} extends pa_{ex} by one objective variable assignment which minimized the highest value among the $E_{una}(pa_{ex})$.

Algorithm 1: Computing the egalitarian solution.

```
   input: An MO-COP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}^{hard}, \mathcal{C}^{soft} \rangle$
1  . output: Set of assignments each corresponding to a egalitarian solution.
2  begin
3      $PA = \{\{\emptyset\}\}$
4      for $j \leftarrow 1$ to $m$ do
5          $PA' \longleftarrow \{\emptyset\}$
6          foreach $pa \in PA$ do
7              foreach $e_i \in E_{una}(pa)$ do
8                  $pa_{ex} \longleftarrow solve(\text{MO-COP}_{(e_i,pa)})$
9                  if $pa_{ex}$ exists then
10                     $pa_{new} \longleftarrow pa_{ex} \cup (p_{new}(e_i) = pa_{ex}(e_i))$
11                     $PA' \longleftarrow PA' \cup pa_{new}$
12         $PA \longleftarrow removeDominated(PA')$
13     return $PA$
```

4 Experimental Results

We empirically evaluated the performance of our techniques on medium- and large- scale benchmarks. Given that these instances are too large for computing the representative solutions [14], we instead analyze the computational time difference between computing the set of all Pareto optimal solutions (PF) and ω-solutions, for each $\omega \in W_*$ with $k = 7$, where W_* is computed "offline" as explained in Section 2.3. Computing the PF can be seen as a preprocessing step for the calculation of representative solutions. We used [2] to compute the PF. In our experiments we aim to show the efficiency gain that Proposition 1 suggests. We used an Intel Core i7-3612QM 2.10GHz with 8 GB of RAM using one single core and one hour for each instance. The programs were implemented in C++ using IBM ILOG CPLEX 12.6.3. We note that assessing the quality of the approximation is out of the scope of this paper. This was done in [14] for small instances, but for larger ones the algorithm in [14] could not compute the representative solutions. This demonstrates the need for developing efficient approximation algorithms for representative solutions, as representative solutions are of high importance but the exact algorithm cannot compute them.

To evaluate the performance of our algorithms, We used random multi-criteria set covering instances. These correspond to MO-COPs with $D_i = \{0, 1\}$. Two sets of instances were considered. The first set (used in [2]) consists of 60 instances with $|\mathcal{X}| = \{100, 150\}, |\mathcal{C}^{hard}| = |\mathcal{X}|/5$, $m = \{3, 4, 5\}$, the scope of $C \in \mathcal{C}^{hard}$ is randomly selected (10 variables per C on average), and $V(A)^t = \sum_{x_i \in \mathcal{X}} c_{i,t} * A(x_i)$ with $c_{i,t}$ chosen uniformly at random from $[1, 2, ..., 1000]$. The second set are classical set covering instances *scp4x*, *scp5x*, and *scp6x* (used in e.g. [3, 8, 11]) from the *OR-Library* [1]. These are large single-objective instances with $|\mathcal{X}| \in \{1000, 2000\}$, $|\mathcal{C}^{hard}| = 200$, and $V(A) = \sum_{x_i \in \mathcal{X}} c_i * A(x_i)$ with c_i taking uniformly-distributed values from $[1, 1000]$. A second objective was generated by randomly permuting the original coefficients c_i. For each instance, five new instances were created, each with a different randomized second objective, obtaining 125 new bi-objective instances.

Aggregated results are given in Table 1, where we compare computational times between calculating seven ω-solutions using our encoding-based (*Enc*) and iterative techniques (*Iter*), and the PF using the BDD approach. The superscript ^{n} indicates that n instances have not been computed within the time limit and do not contribute towards the average time computation. From the results, we conclude that the iterative algorithm is more robust than the encoding-based approach and performs faster for more than two objectives. We believe this is the case since even though the former performs more NP-hard solver calls, the problems that need to be solved in each iteration are easier to compute as Equations 5-6 offer better lower bounds than Equations 1-4 for the ILP solver.

Overall, the results demonstrate that calculating ω-solutions, as an approximation of representative solutions, is significantly faster than computing the complete set of Pareto optimal solutions. Moreover, for the larger instances the PF could not be computed within the time limit, but our approach was able to provide solutions within minutes, offering an alternative to [14]. This shows the applicability and scalability of our approach, providing a reasonable trade-off between representativity and computational efficiency.

$\|\mathcal{X}\|$	m	#	Iter	Enc	BDD [2]
100	3	10	0.9	0.8^{1}	6
100	4	10	1.8	3.6	10
100	5	10	3	31	12
150	3	10	1.2	1.1	132
150	4	10	2.5	14.2	270^{2}
150	5	10	5.4	96^{5}	376^{5}
1000 (scp4x)	2	50	26	17	$-^{10}$
2000 (scp5x)	2	50	39.5	43	$-^{10}$
1000 (scp6x)	2	25	113	89	$-^{10}$

Table 1. Comparison of mean computational times (in seconds) for both sets of instances. Superscripts indicate the number of unsolved instances.

5 Conclusion

We studied the problem of computing weighted egalitarian solutions (ω-solutions) for multi-objective constraint optimization problem. We introduced two new techniques for computing ω-solutions and proved that the problem of computing ω-solutions to approximate the set of representative solutions is no longer Σ_2^{P}-hard. Given the complexity shift, one can expect a substantial efficiency gain and this was indeed the case in our experiments. Out of our two approaches, the iterative algorithm which incrementally builds the solution has proven to be more efficient in terms of computational time and scalability. To the best of our knowledge, it is the first time algorithms for computing weighted egalitarian solutions have been sufficiently developed to efficiently tackle large-size MO-COP instances, offering a direct improvement to the approximation of the representative solutions [14]. The approximation can be viewed as a trade-off between representativity and computational efficiency. Furthermore, for the larger instances where computing all Pareto optimal solutions is not feasible, our approach was able to quickly generate an approximation of the representative solutions.

We believe investigating advanced strategies for weight selection is a good direction for future work, as the weights directly influence the quality of the approximation. In addition, developing algorithms for computing the set of representative solutions would be beneficial, since it would allow the quality estimation of the approximation. Lastly, we plan on exploring the possibility of extending other multi-objective techniques to integrate the concept of ω-solutions natively (e.g. in [2] and [9]).

References

1. J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society*, pages 1069–1072, 1990.

2. D. Bergman and A. A. Ciré. Multiobjective optimization by decision diagrams. In *CP'16*, pages 86–95.
3. C. Gao, X. Yao, T. Weise, and J. Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, 2015.
4. J. Halpern and O. Maimon. Algorithms for the m-center problems: A survey. *European Journal of Operational Research*, 10(1):90–99, 1982.
5. E. Hebrard, B. Hnich, B. O'Sullivan, and T. Walsh. Finding diverse and similar solutions in constraint programming. In *AAAI'05*, pages 372–377.
6. T. Ilhan and M. C. Pinar. An efficient exact algorithm for the vertex p-center problem. Technical report, Bilkent University Technical Report, Department of Industrial Engineering 06533 Ankara Turkey, 2001.
7. S. S. Iyengar and M. R. Lepper. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology*, 79(6):995–1006, 2000.
8. S. Kadioglu and M. Sellmann. Dialectic search. In *CP'09*, pages 486–500.
9. G. Kirlik and S. Sayin. A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems. *European Journal of Operational Research*, 232(3):479–488, 2014.
10. E. Minieka. The m-center problem. *SIAM Review*, 12:138–139, 1970.
11. N. Musliu. Local search algorithm for unicost set covering problem. In *Advances in Applied Artificial Intelligence, IEA/AIE'06*, pages 302–311.
12. C. M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994.
13. T. Petit and A. C. Trapp. Finding diverse solutions of high quality to constraint optimization problems. In *IJCAI'15*, pages 260–267.
14. N. Schwind, T. Okimoto, M. Clement, and K. Inoue. Representative solutions for multi-objective constraint optimization problems. In *KR'16*, pages 601–604.
15. N. Schwind, T. Okimoto, S. Konieczny, M. Wack, and K. Inoue. Utilitarian and egalitarian solutions for multi-objective constraint optimization. In *ICTAI'14*, pages 170–177.
16. V. Torra. The weighted OWA operator. *International Journal of Intelligent Systems*, 12(2):153–166, 1997.
17. M. Zanker, S. Gordea, M. Jessenitschnig, and M. Schnabl. A hybrid similarity concept for browsing semi-structured product items. In *EC-Web'06*, pages 21–30.

A Simple Complete Search for Logic Programming

Jason Hemann[1] (student), Daniel P. Friedman[1] (advisor), William E. Byrd[2], Matthew Might[2] (coauthors)

Indiana University, Bloomington IN 47402, USA
University of Utah, Salt Lake CIty UT 84112, USA

Abstract. We present a straightforward, call-by-value embedding of a simple complete search for a small logic programming language. The entire language is 54 lines of Racket—half of which implement unification. Evidence suggests our combination of expressiveness, concision, and elegance is compelling: since microKanren's release, it has spawned over 50 embeddings in over two dozen host languages.

Keywords: miniKanren, microKanren, logic programming, relational programming, streams, search, Racket

1 Introduction

Logic programming is a highly declarative approach to problem solving that has proven itself applicable to a wide variety of tasks [9]. Consider the below stylized Prolog definition of the `append` relation:

```
append(L,S,O) :- [] = L, S = O
              ; [A|D] = L, [A|R] = O, append(D,S,R).
```

We can use this one definition to solve a variety of problems.

```
(1) ?- append([t,u,v],[w,x],Q).
    Q = [t,u,v,w,x] ?

(2) ?- Q = [L,S], append(L,S,[t,u,v,w,x]).
    L = [], Q = [[],[t,u,v,w,x]], S = [t,u,v,w,x] ?
    ...
    L = [t,u,v,w,x], Q = [[t,u,v,w,x],[]], S = [] ?
```

In (1), we ask "What are the possible results of appending `[t u v]` to `[w x]`?" In (2), we ask for a `Q` composed of terms `L` and `S` such that concatenating `L` and `S` yields `[t u v w x]`; we can find all such possibilities. Our definition of `append` also has many more uses. Prolog is the traditional choice but we can easily transliterate this definition to miniKanren [4], a logic programming DSL shallowly embedded in many hosts, including Scheme and Racket [3].

It is difficult for the uninitiated to understand how logic programming works, or to ferret out the essential details from a language's implementation. An abundance of powerful, useful features combined with years—or even decades—worth of optimizations and improvements can obscure the more fundamental aspects of an implementation's inner workings. Even the small language implementations are rarely designed to be easily understood.

Previous miniKanren implementations, for instance, enmesh the macros that provide miniKanren's syntax with the execution of the logic program itself. This demands the reader have a detailed understanding of macros, and such tight coupling makes it difficult for aspiring implementers in languages without macro support. microKanren separates these concerns and allow functional programmers in a call-by-value language to implement the core logic programming features without the syntactic sugar.

We present here the improved interleaving search of microKanren. miniKanren's interleaving depth-first search is based on Kiselyov et al.'s LogicT transformer [8]. These operators provide a complete search without the performance penalties associated with, for example, breadth-first search. miniKanren interleaves between successfully finding answers, with additional interleaving added to avoid starvation and to avoid problems associated with the eagerness of its host language. We combine the hand-off of control with relation definition, and in doing so decrease the amount of interleaving while maintaining a complete search. We achieve a minimal placement of interleaving points for arbitrary relation definitions.

2 microKanren

We now implement microKanren's search operators: `disj`, `conj`, `define-relation`, and `call/initial-state`. For space, we omit definitions of `==` (a syntactic equality constraint) and `call/fresh` (which scopes new logic variables).

The binary operators `disj` and `conj` act as goal combinators, and they allow us to write composite goals representing the disjunction or conjunction of their arguments.

```
#| Goal × Goal → Goal |#
(define ((disj g1 g2) s/c) ($append (g1 s/c) (g2 s/c)))

#| Goal × Goal → Goal |#
(define ((conj g1 g2) s/c) ($append-map g2 (g1 s/c)))
```

We define `disj` and `conj` in terms of two other functions, `$append` and `$append-map` that operate on finite lists. With these operators, our streams will always be empty or answer-bearing; in fact, they will be fully computed. The result of a goal constructed from `==` must be a finite list, of length 0 or 1. If both of `disj`'s arguments are goals that produce finite lists, then the result of invoking `$append` on those lists is itself a finite list. If both of `conj`'s arguments are goals that produce finite lists, then the result of invoking `$append-map` with a goal and a finite list must itself be a finite list. If `call/fresh`'s argument `f` is a function whose body is a goal, and that goal produces a finite list, then `(call/fresh f)` evaluates to such a goal.

Invoking a goal constructed from these operators in the initial state returns a list of all successful computations, computed in a depth-first, preorder traversal of the search tree generated by the program.

3 Recursion and `define-relation`

It's important that we enrich our implementation to allow recursive relations. Much of the power of logic programming comes from writing relations (e.g. `append`) that refer to themselves or one another in their definitions. At present there are several obstacles. Suppose we'd used `define` to build a function that we hope would behave like a relation:

```
(define (peano n)
  (disj (== n 'z)
        (call/fresh (λ (r) (conj (== n `(s ,r))
                                 (peano r))))))
```

This function purports to be a relation that holds for a particular encoding of Peano numbers. What happens when we use the `peano` relation in the program below? We're hoping to generate some Peano numbers.

```
> ((call/fresh (λ (n) (peano n)))
   '(() . 0))
```

We invoke `(call/fresh ...)` with an initial state. Invoking that goal creates and lexically binds a new fresh variable over the body. The body, `(peano n)`, evaluates to a goal that we pass the state `(() . 1)`. This goal is the disjunction of two subgoals. To evaluate the `disj`, we evaluate its two subgoals, and then call `$append` on the result. The first evaluates to `(((0 . z)) . 1)`, a list of one state.

Invoking the second of the `disj`'s subgoals however is troublesome. We again lexically scope a new variable, and invoke the goal in body with a new state, this time `(() . 2)`. The `conj` goal has two subgoals. To evaluate these, we run the first in the current state, which results in a stream. We then run the second of `conj`'s goals over each element of the resulting stream and return the result. Running this second goal begins the whole process over again. In a call-by-value host, this execution won't terminate. Simply using `define` in this manner will not suffice.

We instead introduce the `define-relation` operator. This allows us to write recursive relations; with a sequence of uses of `define-relation`, we can create mutually recursive relations. Unlike the other operators, `define-relation` is a macro.

```
(define-syntax-rule (define-relation (defname . args) g)
  (define ((defname . args) s/c) (delay/name (g s/c))))
```

Racket's `define-syntax-rule` gives a simple way to construct non-recursive macros. The first argument is a pattern that specifies how to invoke the macro. The macro's first symbol, `define-relation`, is the name of the macro we're defining. Its second argument is a template to be filled in with the appropriate pieces from the pattern. We do implement `define-relation` in terms of Racket's `define`.

This macro expands a name, arguments, and a goal expression to a `define` expression with the same name and number of arguments and whose body is a goal. It takes a state and returns a stream, but unlike the others we've seen before, this goal returns an immature stream. When given a state `s/c`, this goal returns a promise that evaluates the original goal `g` in the state `s/c` when forced, returning a stream. A promise that returns a stream is itself an immature stream.

`define-relation` does two useful things for us: it adds the relation name to the current namespace, and it ensures that the function implementing our relation is total. It turns out that we will *never* re-evaluate an immature stream. Unlike `delay`, `delay/name` doesn't *memoize* the result of forcing the promise, so it is like a "by name" variant of `delay`.

We implement `define-relation` as a macro of necessity. It is critical that the expression `g` not be evaluated prematurely: the objective is to delay the invocation of `g` in `s/c`. In a call-by-value language, a function would (prematurely) evaluate its argument and would not delay the computation.

Below, we revisit the `peano` example, but this time using `define-relation`. Non-termination of relation invocations is no longer an issue. Instead, the goal `(peano n)`, when invoked, immediately returns an immature stream.

```
(define-relation (peano n)
  (disj (== n 'z)
        (call/fresh (λ (r) (conj (== n `(s ,r))
                                 (peano r))))))
```

We can also write recursive relations whose goals quite clearly will never produce answers.

```
(define-relation (unproductive n)
  (unproductive n))
```

We can now introduce `$append` and `$append-map`. Their definitions are in fact those of `append` and `append-map`, functions over lists that are standard to many languages [11], but augmented with support for immature streams.

```
(define ($append $1 $2)
  (cond
    ((null? $1) $2)
    ((promise? $1) (delay/name ($append (force $1) $2)))
    (else (cons (car $1) ($append (cdr $1) $2)))))
```

If the recursive argument to `$append` is an immature stream, we return an immature stream, which, when forced, continues appending the second to the first. Likewise, in `$append-map`, when `$` is an immature stream, we return an immature stream that will continue the computation but still forcing the immature stream. Rather than `delay/name`, `force`, and `promise?`, we could have used `(λ () ...)`, procedure invocation, and `procedure?`. Using λ to construct a procedure delays evaluation, and `procedure?` would be our test for an immature stream. We choose Racket's special-purpose primitives for added clarity, but implementers targeting other languages can use anonymous procedures if these primitives aren't available. In languages without macros, the programmer could explicitly add a delay at the top of each relation; this has though the unfortunate consequence of exposing the implementation of streams.

```
#| Goal × Stream → Stream |#
(define ($append-map g $)
  (cond
    ((null? $) '())
    ((promise? $) (delay/name ($append-map g (force $))))
    (else ($append (g (car $)) ($append-map g (cdr $))))))
```

After these changes, it's possible to execute a program and produce neither the empty stream nor an answer-bearing one. We might produce instead an immature stream.

```
> ((call/fresh (λ (n) (peano n)))
   '(() . 0))
#<promise>
```

To resolve this we need to do something special when we invoke a goal in the initial state.

4 `call/initial-state`

At the very least, we would like to know if our programs are *satisfiable* or not. That is, we would hope to get at least one answer if one exists, and the empty list if there are none. The `call/initial-state` operator ensures that if we return, we return with a list of answers.

```
#| Maybe Nat⁺ × Goal→ Mature |#
(define (call/initial-state n g) (take n (pull (g '(() . 0)))))
```

`call/initial-state` takes an argument `n` which represents the number of answers to retrieve. `n` may just be a positive natural number, in which case we return at most that many answers. Otherwise, we provide `#f`, indicating microKanren should return *all* answers. It also takes a goal as an argument. The function `pull` takes a stream as argument, and if `pull` terminates, it returns a mature stream. As streams may be unproductive, it is not always possible to produce a mature stream. As a result, `pull`, and consequently `take` and `call/initial-state`, are partial functions. These are the only partial functions in the microKanren implementation.

```
#| Stream → Mature |#
(define (pull $) (if (promise? $) (pull (force $)) $))
```

`take` receives the mature stream that is the result of `pull` and, `n`, the argument dictating whether to return all, or just the first n elements of the stream.

```
#| Maybe Nat⁺ × Mature → List |#
(define (take n $)
  (cond
    ((null? $) '())
    ((and n (zero? (- n 1))) (list (car $)))
    (else (cons (car $) (take (and n (- n 1)) (pull (cdr $)))))))
```

Our microKanren is now capable of creating, combining, and searching for answers in infinite streams.

```
> (call/initial-state 2
    (call/fresh (λ (n) (peano n))))
'((((0 . z)) . 1) (((1 . z) (0 . (s 1))) . 2))
```

Thus, we have brought microKanren programs into the delay monad [2,5]: rather than always returning a list implementation of non-deterministic choice, we either have no values, a value now (possibly more than one), or something we can search later for a value. `pull`, since it forces an actual value out of a promise, is akin to run in the delay monad. `take` bears a similar relationship to run in the list monad.

5 Interleaving, Completeness, and Search

Although microKanren is now capable of creating and managing infinite streams, it doesn't manage them as well as we'd like. Consider what happens in the following program execution:

```
> (call/initial-state 1
    (call/fresh (λ (n) (disj (unproductive n)
                             (peano n)))))
```

We would like the program to return a stream containing the `n`s for which `unproductive` holds and in addition the `n`s for which `peano` holds. We know from Section 3 that there are no `n`s for which `unproductive` holds, but infinitely many for `peano`. The stream should contain only `n`s for which `peano` holds. It's perhaps surprising, then, to learn that this program loops infinitely.

Streams that result from using `unproductive` will always be, as the name suggests, unproductive. When executing the program above, such an unproductive stream will be the recursive argument `$1` to `$append`. Unproductive streams are necessarily immature. According to our definition of $append, we always return the immature stream. When we force this immature stream, it calls $append on the forced stream value of (the delayed) $1 and $2. Since unproductive is unproductive, this process continues without ever returning any of the results from peano.

Such surprising results are not solely the consequence of goals with unproductive streams. Consider the definition of church.

```
(define-relation (church n)
  (call/fresh (λ (b) (conj (== n ‘(λ (s) (λ (z) ,b)))
                           (peano b)))))
```

The relation church holds for Church numerals. Using a newly created variable b, it constructs a list resembling a lambda-calculus expression whose body is the variable b. It uses peano to generate the body of the numeral. We can thus use it to generate Church numerals in a manner analogous to our use of peano. But consider the program below, wherein the resulting stream is productive, but only contains elements for which peano holds.

```
> (call/initial-state 3
    (call/fresh (λ (n) (disj (peano n)
                             (church n)))))
’((((0 . z)) . 1) (((1 . z) (0 . (s 1))) . 2)
  (((2 . z) (1 . (s 2)) (0 . (s 1))) . 3))
```

Under the default Racket printing convention, “.” is suppressed when it precedes a “(”. We retain the “.” for legibility—the Racket parameter current-print controls this behavior.

Our implementation of $append in Section 3 induces a depth-first search. Depth-first search is the traditional search strategy of Prolog and can be implemented quite efficiently. As we've seen though, depth-first search is an *incomplete* search strategy: answers can be buried infinitely deep in a stream. The stream that results from a disj goal produces elements of the stream from the second goal only after exhausting the elements of the stream from the first.

```
#| Stream × Stream → Stream |#
(define ($append $1 $2)
  (cond
    ...
    ((promise? $1) (delay/name ($append (force $1) $2)))))
```

As a result, even if answers exist microKanren may fail to produce them. We will remedy this weakness in $append, and provide microKanren with a simple complete search. We want microKanren to guarantee each and every answer should occur at a finite position in the stream. Fortunately, this doesn't require a significant change.

```
#| Stream × Stream → Stream |#
(define ($append $1 $2)
  (cond
    ...
    ((promise? $1) (delay/name ($append $2 (force $1))))))
```

That's it. This one change to the `promise?` line of `$append` is sufficient to make `disj` *fair* and to transform our search from an incomplete, depth-first search to a complete one.

Interestingly, we haven't reconstructed some particular, fixed, complete search strategy. Instead, the search strategy of microKanren programs is program- and query-specific. The particular definitions of a program's relations, together with the goal from which it's executed, dictates the order we explore the search tree. By contrast, Spivey and Seres implement breadth-first search, also a complete search, in a language similar to microKanren [12].

Relying on non-strict evaluation simplifies their implementation; manually managing delays would make the call-by-value version less elegant than their implementation. Even excepting that, their implementation requires a somewhat more sophisticated transformation than does ours. Kiselyov et al. describe a different mechanism to achieve a complete search, but they too rely on non-strict evaluation [8]. We achieve a simpler implementation of a complete search by using the delays as markers for interleaving our streams.

6 Conclusion and Related Work

There has been an extensive research on logic programming implementation [1]. Spivey and Seres's [12] present a Haskell embedding of a language quite similar to microKanren. They begin with depth-first search language, and through transformations derive an implementation of breadth-first search.

Hinze [6,7] and Kiselyov et al. [8] implement backtracking with asymptotic performance improvements over stream-based approaches like that used in microKanren and the works cited above. These context-passing implementations are also more complicated to understand and to implement. We chose to use streams in part to more easily communicate ideas.

The fair search operators in Kiselyov et al.'s LogicT monad provide the basis of the interleaving search in earlier miniKanren implementations. The LogicT transformer augments an arbitrary monad with backtracking and control operators similar to those we use. Because we have access to the whole logic program in our embedding and take special care to control interleaving in recursions, we can use less frequent interleaving and maintain a complete search.

Our development led us to a number of interesting, still-open problems. Hinze [6] shows our list-based implementation of nondeterminism is asymptotically slower than a continuation-based "context-passing" implementation. We would like to combine our manual control of delays with a context-passing implementation á la Hinze and Kiselyov et al. [8].

While `define-relation` is sufficient to ensure our search is complete, it in general causes more interleaving than necessary. For instance, mutually-recursive relations only need one interleaving point between them, and we don't need to interleave at all deterministic relations. We could statically "push down" the delays into the body of a relation, reducing the amount of interleaving we perform while retaining a complete search.

We would also like to mechanically prove the correctness of microKanren's search with a dependently-typed implementation whose types encode correctness of substitutions and unification. Earlier work by Kumar [10] in mechanizing facets of a miniKanren implementation might provide a starting point.

References

1. Balbin, I., Lecot, K.: Logic Programming: A Classified Bibliography. Springer Science & Business Media (2012)
2. Capretta, V.: General recursion via coinductive types. Logical Methods in Computer Science 1(2) (2005)
3. Flatt, M., PLT: Reference: Racket. Tech. Rep. PLT-TR-2010-1, PLT Design Inc. (2010), http://racket-lang.org/tr1/
4. Friedman, D.P., Byrd, W.E., Kiselyov, O.: The Reasoned Schemer. MIT Press, Cambridge, MA (2005)
5. Ganz, S.E., Friedman, D.P., Wand, M.: Trampolined Style. In: Proc. 4th ICFP. pp. 18–27. ACM (1999)
6. Hinze, R.: Deriving backtracking monad transformers. In: ACM SIGPLAN Notices. vol. 35, pp. 186–197. ACM (2000)
7. Hinze, R.: Prolog's control constructs in a functional setting: Axioms and implementation. International Journal of Foundations of Computer Science 12(02), 125–170 (2001)
8. Kiselyov, O., Shan, C., Friedman, D.P., Sabry, A.: Backtracking, interleaving, and terminating monad transformers: (functional pearl). In: Danvy, O., Pierce, B.C. (eds.) Proceedings of the 10th ACM SIGPLAN ICFP. pp. 192–203. ACM (September 2005)
9. Kowalski, R.A.: Logic for Problem Solving. North-Holland/Elsevier (1979)
10. Kumar, R.: Mechanising Aspects of miniKanren in HOL (2010), australian National University. Bachelors thesis
11. Shivers, O.: List Library. Scheme Request for Implementation. SRFI-1 (1999), http://srfi.schemers.org/srfi-1/srfi-1.html
12. Spivey, J., Seres, S.: Embedding Prolog in Haskell. In: Meier, E. (ed.) Haskell 99 (1999)

MDDs: Sampling and Probability Constraints

Guillaume Perez[1] and Jean-Charles Régin[2]

Université Nice-Sophia Antipolis, I3S UMR 6070, CNRS, France
[1] PhD student, [2] Advisor
guillaume.perez06@gmail.com, jcregin@gmail.com

Abstract. We propose to combine two successful techniques of Artificial Intelligence: sampling and Multi-valued Decision Diagrams (MDDs). Sampling, and notably Markov sampling, is often used to generate data resembling to a corpus. However, this generation has usually to respect some additional constraints, for instance to avoid plagiarism or to respect some rules of the application domain. We propose to represent the corpus dependencies and these side constraints by an MDD and to develop some algorithms for sampling the solutions of an MDD while respecting some probabilities or a Markov chain. In that way we obtain a generic method which avoids the development of ad-hoc algorithm for each application as it is currently the case. In addition, we introduce new constraints for controlling the probabilities of the solutions that are sampled. We experiments our method on a real life application: the geomodeling of a petroleum reservoir, and on the generation of French alexandrines. The obtained results show the advantage and the efficiency of our approach.

NightSplitter: a scheduling tool to optimize (sub)group activities

Tong Liu[1], Roberto Di Cosmo[2], Maurizio Gabbrielli[1] and Jacopo Mauro[3]

[1] DISI, University of Bologna, Bologna, Italy
[2] INRIA and University Paris Diderot, Paris, France
[3] Department of Informatics, University of Oslo, Oslo, Norway
{t.liu, maurizio.gabbrielli}@unibo.it, roberto@dicosmo.org, jacopom@ifi.uio.no

Abstract. Humans are social animals and usually organize activities in groups. However, they are often willing to split temporarily a bigger group in subgroups to enhance their preferences. In this work we present NightSplitter, an on-line tool that is able to plan movie and dinner activities for a group of users, possibly splitting them in subgroups to optimally satisfy their preferences. We first model and prove that this problem is NP-complete. We then use Constraint Programming (CP) or alternatively Simulated Annealing (SA) to solve it. Empirical results show the feasibility of the approach even for big cities where hundreds of users can select among hundreds of movies and thousand of restaurants.
[4]

[4] In this work, Tong Liu is a PhD student, other authors are advisors.

Dependency Learning for QBF

Tomáš Peitl, Friedrich Slivovsky, and Stefan Szeider

Algorithms and Complexity Group, TU Wien, Austria

Abstract. Quantified Boolean Formulas (QBFs) can be used to succinctly encode problems from domains such as formal verification, planning, and synthesis. One of the main approaches to QBF solving is Quantified Conflict Driven Clause Learning (QCDCL). By default, QCDCL assigns variables in the order of their appearance in the quantifier prefix so as to account for dependencies among variables. Dependency schemes can be used to relax this restriction and exploit independence among variables in certain cases, but only at the cost of nontrivial interferences with the proof system underlying QCDCL. We propose a new technique for exploiting variable independence within QCDCL that allows solvers to learn variable dependencies on the fly. The resulting version of QCDCL enjoys improved propagation and increased flexibility in choosing variables for branching while retaining ordinary (long-distance) Q-resolution as its underlying proof system. In experiments on standard benchmark sets, an implementation of this algorithm shows performance comparable to state-of-the-art QBF solvers.

Productive Corecursion in Logic Programming[*]

EKATERINA KOMENDANTSKAYA

Heriot-Watt University, Edinburgh, Scotland, UK
(*e-mail:* `ek19@hw.ac.uk`)

YUE LI

Heriot-Watt University, Edinburgh, Scotland, UK
(*e-mail:* `yl55@hw.ac.uk`)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Logic Programming is a Turing complete language. As a consequence, designing algorithms that decide termination and non-termination of programs or decide inductive/coinductive soundness of formulae is a challenging task. For example, the existing state-of-the-art algorithms can only semi-decide coinductive soundness of queries in logic programming for regular formulae. Another, less famous, but equally fundamental and important undecidable property is productivity. If a derivation is infinite and coinductively sound, we may ask whether the computed answer it determines actually computes an infinite formula. If it does, the infinite computation is productive. This intuition was first expressed under the name of computations at infinity in the 80s. In modern days of the Internet and stream processing, its importance lies in connection to infinite data structure processing.

Recently, an algorithm was presented that semi-decides a weaker property – of productivity of logic programs. A logic program is productive if it can give rise to productive derivations. In this paper we strengthen these recent results. We propose a method that semi-decides productivity of individual derivations for regular formulae. Thus we at last give an algorithmic counterpart to the notion of productivity of derivations in logic programming. This is the first algorithmic solution to the problem since it was raised more than 30 years ago. We also present an implementation of this algorithm.

KEYWORDS: Horn Clauses, (Co)Recursion, (Co)Induction, Infinite Term Trees, Productivity.

[*] This work has been partially supported by EPSRC grant EP/K031864/1-2