The Seventeenth International Conference on Principles and Practice of Constraint Programming (CP 2011)

Doctoral Program Proceedings

Perugia, Italy
12–16 September 2011

Preface

The Doctoral Program was organized as a full-day event at the Seventeenth International Conference on Principles and Practice of Constraint Programming, CP 2011, and took place in Perugia, Italy, on September 12, 2011.

The purpose of the Doctoral Program is to provide an opportunity for current PhD students to meet each other as well as researchers in the eld. Participants present their work via both a poster and a talk, and discuss their research with a mentor with similar research interests.

The program is open to all PhD students involved in Constraint Programming or related elds at any level. Applicants were required to submit a short paper (no more than 6 pages) containing original unpublished work which is at least in part the work of the student. It can either be completed or in progress.

Any students with (long, short, or application) papers accepted in the CP conference did not need to submit a paper to the Doctoral Program, but were automatically eligible for entry (assuming, of course, that the paper is at least in part the work of the student).

The papers collected here are those which have been submitted directly to the Doctoral Program. They present a wide variety of work, both completed and in progress, being undertaken by the current generation of PhD students in quite different phases of their studies. We also included the abstracts of papers by students who were accepted to the Doctoral Program because they have a paper in the main CP conference. The corresponding full papers can be found in the CP 2011 conference proceedings (volume 6876 of LNCS, Springer Verlag).

The Doctoral Program would not be possible without the support we received from many people and organizations. We would like to thank the program committee members, the CP conference chair Stefano Bistarelli, and the sponsors of the CP conference, in particular PSI Metals and the Italian Association for Arti cial Intelligence (AI*IA).

Christopher Jefferson and Guido Tack
Doctoral Program Chairs, 2011

Doctoral Program Organization

Doctoral Program Chairs:

Christopher Jefferson, University of St. Andrews, UK
Guido Tack, Katholieke Universiteit Leuven, Belgium

Program Committee:

Lucas Bordeaux, Microsoft Research
Sebastian Brand, NICTA Victoria Research Lab, University of Melbourne
Andrei Bulatov, Simon Fraser University
Sophie Demassey, École des Mines de Nantes
Stefano Gualandi, Politecnico di Milano
Christopher Jefferson, University of St. Andrews
Inês Lynce, IST/INESC-ID, Technical University of Lisbon
Chris Mears, Monash University
Peter Nightingale, University of St Andrews
Claude-Guy Quimper, Université Laval
Andrea Rendl, Austrian Institute of Technology
Horst Samulowitz, IBM Research
Meinolf Sellmann, IBM Research
Guido Tack, Katholieke Universiteit Leuven
Willem-Jan Van Hoeve, Carnegie Mellon University
Roland Yap, National University of Singapore
Magnus Agren, Tomologic
Stanislav Zivný, Computing Laboratory, University of Oxford

Additional Reviewers:

Martin Cooper
Andreas Schutt

Doctoral Program Participants

The following 30 students were selected to participate in the doctoral program.

Students with full papers in the main conference:

Alessio Bon etti
Jessica Davies
Alexis De Clercq
Samir A. Mohamed Elsayed
Jean-Guillaume Fages
Antti Hyvärinen
Roger Kameugne
Ronan Le Bras
Vianney le Clément
Amit Metodi
Eoin O' Mahony
Marie Pelleau
Joseph Scott
Walid Trabelsi

Students with short doctoral program papers:

Markus Aschinger
Dario Campagna
Nicholas Downing
Guillaume Escamocher
Natalia Flerova
Maria Francisco
Matthew Gwynne
Katherine Lai
Arnaud Letort
Jean-Baptiste Mairy
Florence Massen
Faten Nabli
Dario Pacino
Anna Roubickova
Silvia Tomasi
Hu Xu

Table of Contents

Doctoral Program Papers

Abstracts of Papers in the Main CP 2011 Conference

A new logic-based formalism for Configuration problems*

Markus Aschinger, Conrad Drescher, Georg Gottlob (Supervisor)
firstname.lastname@cs.ox.ac.uk

Department of Computer Science, University of Oxford

Abstract. In this paper we present the core of a logic-based high-level representation language for expressing configuration problems. It shall allow to model these problems in an intuitive and declarative way, the dynamic aspects of configuration notwithstanding. Our logic enforces that configurations contain only finitely many components and reasoning can be reduced to the task of model construction.

1 Introduction

In this work we describe ongoing work on a new logic-based formalism expressive enough for dealing with practical configuration problems. Configuration systems are one of the most successful applications of AI-techniques. In industrial environments, they support the configuration of complex products and, compared to manual processes, help to reduce error rates and increase throughput [7]. The following definition by Mittal and Frayman [5] describes what is typically meant by a configuration problem.

Definition 1. *Given a fixed, predefined set of components, where a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints, some description of the desired configuration and some criteria for making optimal selections.*
Build: One or more configurations that satisfy all the requirements, where a configuration is a set of components and a description of the connections between the components in the set, or, detect inconsistencies in the requirements.

For instance, a car configurator must compute a valid vehicle variant satisfying the user requirements and all applicable commercial and technical restrictions derived from the marketing and engineering policies of the manufacturer. In typical configuration problems, the number of components in a solution is unknown beforehand because of various reasons, for example if the number of components depends on the choices made for

* Work funded by EPSRC Grant EP/G055114/1 Constraint Satisfaction for Configuration: Logical Fundamentals, Algorithms and Complexity

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 1–6, Perugia, Italy, 12 September 2011.

other components. This brings up the need for creating new components on-the-fly throughout the solving process.
To date there is still a lack of high-level knowledge representation (KR) tools being able to cope with this demand without requiring knowledge about specific solving algorithms. In this work a new logical formalism will be investigated to deal with the aforementioned challenges that is expressive enough for practical configuration problems. The aim is to develop a high-level representation language suitable for modelling configuration knowledge for practical applications. This will consist of syntactic restrictions and, at the same time, extensions of First Order Logic (FO). The logical language is to be defined with the goal that the configuration knowledge can be represented compactly and conveniently. We plan to transform the logic representation into a low-level input format for various solvers, e.g. SAT or Integer Programming, in an automated transformation step. This approach allows the modelling of general configuration problems in an intuitive and declarative way without the need of having knowledge about the underlying solving algorithms.

2 Related work

Over the years several different approaches for Configuration have been investigated, e.g. expert systems, rule-based systems, nonmonotonic reasoning, case-based reasoning, description logics and constraint processing. A recent survey is given by Junker in [3]. The constraint satisfaction problem (CSP) is the most widely used approach for the formalisation of configuration problems. Although this allows a very natural way of modelling, the standard CSP formulation is not appropriate for configuration problems in a knowledge representation (KR) sense: it does not feature variables that are conditionally activated depending upon the values of the other components in the solution.
In the area of constraint-based configuration, a number of extensions of the traditional CSP paradigm have been developed to cope with the dynamic aspects of configuration problems. In Conditional CSPs only a relevant subset of the variables and constraints is taken into consideration for generating a solution because of additional activation constraints [4]. Composite CSP is another formalism informally introduced by Sabin and Freuder. It allows to model CSPs where variables can have subproblems (sub-CSPs) as values [6]. However, in both formalisms the number of possibly activated variables and constraints is defined in advance. For this reason it comes as no surprise that these formalisms in fact have the same expressive power as classic CSPs since there exist polynomial-time many-one reductions between composite CSPs and conditional CSPs, and also between conditional CSPs and classic CSPs [8].
There have also been some previous attempts to capture configuration with logic-based formalisms. Traditional CSPs can be equivalently formulated as sentences of the existential fragment $\exists FO_{\wedge,+}$ of $FO_{\wedge,+}$, which is the fragment of first-order logic of formulae with arbitrary quantifications and conjunctions, but with no negations or disjunctions. Apparently, one well-suited candidate for extending a standard CSP is to add the logical implication as a connective, besides conjunction [2]. The resulting

language is the fragment $\exists FO_{\rightarrow,\wedge,+}$ of FO. This language is - even without further extensions - sufficiently strong for modelling a wide range of configuration constraints and is one of the main starting points for our own formalism. On the other hand, this formalism still makes the underlying assumption that all variables range over an initially given finite set of values (fixed finite domain) and all constraints must be coded in extension in the constraint database.

3 A new logic-based modelling formalism

To cope with the aforementioned challenges, we will introduce a new logic-based framework for modelling practical configuration problems. The basic idea is to describe a configuration problem (the problem domain) by a set of logical sentences. The task of finding a configuration is then reduced to the problem of finding a model for the logical sentences.

The logic is based on classical many-sorted logic with equality interpreted as identity. For restricting the number of potential connections between components we use existential counting quantifiers. For example, we might have a formula $\exists_l^u x\phi(x)$ enforcing that the number of different x such that $\phi(x)$ holds is restricted to be within the range $[l, u]$, with l and u natural numbers such that $l \leq u$.

Sorts: We stipulate that for every component type there is a sort Id, the component's identifier. Other sorts may include the natural numbers or sorts for specific component attributes. We stipulate that there are *standard names* for the elements in the domain of each sort, with any two different names interpreted as being 'not equal'.

Components: The different components are modelled as n-ary predicates $Component(id, \boldsymbol{x})$, with id the component's identifier, and $\boldsymbol{x}$ a vector of component attributes. Components come in two major variants, they are either *input* or *generated* components. For a component C_1 of the input variant we will make a closure assumption on the domain of the components identifiers $\Delta_{C_1}^{id}$ — none but the component identifiers mentioned in the input exists. We also stipulate that a configuration problem includes at least one component of the input variant. In order to transform the model into SAT or OPL, we need to enforce finiteness of the model by computing upper bounds on the generated components, to be explained below.

Connections: Configuration is about connecting components: For every set $\{C_1, C_2\}$ of potentially connected components we introduce one of the binary predicate symbols $C_1 2 C_2$ and $C_2 2 C_1$ - it does not matter which. We allow connections from a component type to itself, i.e., $C2C$. We stipulate that there be no isolated component types that do not partake in any connection. Based on the different component types we can distinguish three different connection types: input2input, input2generated, and generated2generated. For every connection predicate $C_1 2 C_2$ for connections between component types C_1 and C_2 two formulas are included:

$$\forall(id_1, \boldsymbol{x})\exists_{l_1}^{u_1}(id_2, \boldsymbol{y})C_1(id_1, \boldsymbol{x}) \Rightarrow \quad (1)$$
$$C_1 2 C_2(id_1, id_2) \wedge C_2(id_2, \boldsymbol{y}) \wedge \phi(id_1, id_2, \boldsymbol{x}, \boldsymbol{y})$$

$$\forall(id_2, \boldsymbol{x})\exists_{l_2}^{u_2}(id_1, \boldsymbol{y})C_2(id_2, \boldsymbol{x}) \Rightarrow \quad (2)$$
$$C_1 2 C_2(id_1, id_2) \wedge C_1(id_1, \boldsymbol{y}) \wedge \psi(id_1, id_2, \boldsymbol{x}, \boldsymbol{y})$$

The first formula says how many components of type C_2 can be connected to any given component of type C_1, with the subformula $\phi(id_1, id_2, \boldsymbol{x}, \boldsymbol{y})$ expressing constraints that must hold additionally, e.g. an aggregate function like $\sum n \leq \mathit{Capacity}$ with n being a component attribute. The second formula is for the other direction. If the connection is from a component type to itself only one of the formulas is included.
Next to binary connections, we also support 1-to-many connections. These rules are used to express the relationship between a component and a set of connected components. It is mandatory in this setup to define a formula expressing the cardinalities ranging from the single component to the set of components (3). Notice that the single component on the left-hand side can't be part of the set on the right-hand side. A rule of this type can coexist next to binary connections between the single component and components in the set.
Optionally, we can also state cardinalities in the direction from the set of components to the single component, i.e. a many-to-one connection (4). Since this expresses the bounds from every component in the set to the single component, there is no need of additionally stating binary components. In fact, it is allowed to either define a many-to-one rule or a set of binary connections for this direction, but not both simultaneously. This restriction is needed to ensure consistency in the propagation of bounds, to be explained in more detail below.

$$\forall(id_1, \boldsymbol{x})\exists_{l}^{u}(id_i, \boldsymbol{y})C_1(id_1, \boldsymbol{x}) \Rightarrow [\bigvee_i C_1 2 C_i(id_1, id_i) \wedge C_i(id_i, \boldsymbol{y})] \quad (3)$$

$$\forall(id_i, \boldsymbol{x})\exists_{l}^{u}(id_1, \boldsymbol{y})[\bigvee_i C_i(id_i, \boldsymbol{x})] \Rightarrow C_1 2 C_i(id_1, id_i) \wedge C_1(id_1, \boldsymbol{y}) \quad (4)$$

Calculating upper and lower bounds: For computing the possible domain sizes of generated components, we extract Diophantine inequalities from the connection formulas. This builds up on the work by Falkner et al. about semantics of UML class diagrams and cardinalities applied to the configuration domain [1]. Knowing these domain sizes is very important for the planned translation to SAT or OPL models.
Referring back to the binary connection defined by formulas (1) and (2), assuming C_1 is an input and C_2 is a generated component, we are able to calculate upper and lower bounds for component C_2. After some simple combinatorics omitted due to space reasons, we get the following formula for the bounds of C_2:

$$\left\lceil \frac{l_1 * |C_1|}{u_2} \right\rceil \leq |C_2| \leq \left\lfloor \frac{u_1 * |C_1|}{l_2} \right\rfloor \quad (5)$$

In order to define a lower resp. an upper bound for C_2, we need the bounds l_2 resp. u_2 in the direction of C_1. It is necessary to at least have upper bounds on all generated components in order to ensure finiteness of the model. For this reason there needs to be a directed path to an input component without lower bounds of zero. We generally allow the definition of lower bounds with value zero on connection cardinalities, as long as the boundedness is secured by another connection. A propagation algorithm makes sure that bounds are propagated throughout the whole connection graph.

4 Example: Modified Bin-Packing

We want to explain our approach by means of a simple Bin-Packing example containing the basic characteristics of configuration problems, i.e. connecting components and dynamically generating new components. We distinguish between two types of *Things* A and B with all *Things* having a certain size. The *Bins* have an upper bound on the total size for *Things* of each type that can be put into them. *Things* are input components while the *Bins* are generated components with the goal of their number being minimized. The problem can be described by the following formulas:

$$\forall(id_{\mathrm{TA}}, size)\exists_1^1(id_{Bin})C_{\mathrm{TA}}(id_{\mathrm{TA}}, size) \Rightarrow \\ C_{\mathrm{TA}}2C_{Bin}(id_{\mathrm{TA}}, id_{Bin}) \wedge C_{Bin}(id_{Bin}) \tag{6}$$

$$\forall(id_{Bin})\exists_0^5(id_{\mathrm{TA}}, size)C_{Bin}(id_{Bin}) \Rightarrow \\ C_{\mathrm{TA}}2C_{Bin}(id_{\mathrm{TA}}, id_{Bin}) \wedge C_{\mathrm{TA}}(id_{\mathrm{TA}}, size) \wedge \sum size \leq 5 \tag{7}$$

$$\forall(id_{\mathrm{TB}}, size)\exists_1^1(id_{Bin})C_{\mathrm{TB}}(id_{\mathrm{TB}}, size) \Rightarrow \\ C_{\mathrm{TB}}2C_{Bin}(id_{\mathrm{TB}}, id_{Bin}) \wedge C_{Bin}(id_{Bin}) \tag{8}$$

$$\forall(id_{Bin})\exists_0^2(id_{\mathrm{TB}})C_{Bin}(id_{Bin}) \Rightarrow \\ C_{\mathrm{TB}}2C_{Bin}(id_{\mathrm{TB}}, id_{Bin}) \wedge C_{\mathrm{TB}}(id_{\mathrm{TB}}, size) \wedge \sum size \leq 2 \tag{9}$$

$$\forall(id_{Bin})\exists_1(id_T, \boldsymbol{y})C_{Bin}(id_{Bin}) \Rightarrow \\ (C_{\mathrm{TA}}2C_{Bin}(id_T, id_{Bin}) \wedge C_{\mathrm{TA}}(id_T, \boldsymbol{y}))\vee \\ (C_{\mathrm{TB}}2C_{Bin}(id_T, id_{Bin}) \wedge C_{\mathrm{TB}}(id_T, \boldsymbol{y})) \tag{10}$$

Formula (6) states that every *ThingA* has to be put into exactly one *Bin*. The subformula at the end of formula (7) determines that the total size of *ThingA* components placed in a *Bin* must not exceed 5. Up to 5 *ThingA* components can be put into a *Bin* in case all of them having minimum size 1 (hence the cardinality upper bound is 5). Formulas (8) and (9) analogously define the binary connection for *ThingB*.

Assume having an instance with 20 *Things* of each type, connection *ThingA-Bin* gives a lower bound of 4 and connection *ThingB-Bin* gives a lower bound of 10 for component *Bin* using the computation defined in (3). We take the maximum of all available values, hence the lower bound for *Bin* is 10. Notice that in (7) and (9) the cardinality lower bounds of the connections are defined as zero to express the situation that a *Bin* could contain only one type of *Thing* without the other. This results in the fact that we can't compute an upper bound for *Bin* using the binary connections and this violates the finite model requirement. In order to express that for a *Bin* to exist it needs to have at least one *Thing* in it, we can define a one-to-many connection between *Bin* and the set of *Things* like in rule (10). It is sufficient to only define a lower bound for this connection and in conjunction with the binary connections we can now compute an upper bound of 40 for a *Bin*, which would occur in a situation where every *Thing* would be put in a separate *Bin*.

5 Conclusion and future work

We presented the basic framework of a logic-based formalism for modelling configuration problems. The main focus lies on the knowledge representation level and not on competing with optimized algorithms for specific problems. As a next step we plan to define a precise language for constraints involving arithmetic expressions and aggregate functions. The automatic transformation of our formalism into OPL is also one of our next milestones. We also plan to integrate taxonomies in the form of component ontologies and to specify the needed domain closure axioms. Further perspectives are to analye the expressive power and the complexity of decision and computational problems related to this formalism in detail.

References

1. Falkner, A., Feinerer, I., Salzer, G., Schenner, G.: Computing product configurations via UML and integer linear programming. International Journal of Mass Customisation 3(4), 351–367 (2010)
2. Gottlob, G., Greco, G., Mancini, T.: Conditional constraint satisfaction: Logical foundations and complexity. In: IJCAI. pp. 88–93 (2007)
3. Junker, U.: Configuration. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, pp. 837 – 874. Elsevier (2006)
4. Mittal, S., Falkenhainer, B.: Dynamic constraint satisfaction problems. In: Proc. AAAI-90. pp. 25 – 32. MIT Press (1990)
5. Mittal, S., Frayman, F.: Towards a generic model of configuraton tasks. In: IJCAI. pp. 1395–1401 (1989)
6. Sabin, D., Freuder, E.C.: Configuration as composite constraint satisfaction. In: Proc. of the AIMRP Workshop (1996)
7. Sabin, D., Weigel, R.: Product configuration frameworks - a survey. IEEE Intelligent Systems 13(4), 42–49 (1998)
8. Thorstensen, E.: Capturing configuration. In: Proceedings of Doctoral Program at CP 2010. St. Andrews, Scotland (2010)

A CLP-based System For Custom Product Manufacturing

Dario Campagna* and Andrea Formisano**

Dipartimento di Matematica e Informatica, Università di Perugia, Italy
(dario.campagna|formis)@dmi.unipg.it

Abstract. In this paper we present the first results of an ongoing work on product and production process modeling and configuration. First, we describe a graphical modeling framework that allows one to model both a product and its production process. Then, we outline a possible CLP-based implementation of such product/process configuration system.

1 Introduction

During the past years, the interest of companies deploying mass customization strategies toward product configuration systems has grown, since such software can support them in the management of configuration processes. Many research studies have been conducted so far on this topic (see, e.g., [9]), and different software product configurators have been proposed (see, e.g., [6,4,10]).

Process modeling tools, differently from product configuration systems, allow one effectively to deal with (business) process management. In general, they allow the user to define a description of a process, and guide her through the process execution. Also within this field it is possible to find tools and scientific works (see, e.g, [11,8]).

Mass customization needs to cover the management of the whole customizable product cycle, from product configuration to product production. Current product configuration systems and researches on product configuration focus only on product modeling and on techniques for configuration process support. They do not cover product production process issues, despite the advantages that coupling of product with process modeling and configuration could give.

Inspired by the works of Aldanondo et al. (see, e.g., [1]), we started implementing a CLP-based configuration system on top of a graphical framework for modeling configurable products (whose producible variants can be represented as trees), and their production processes. The main intent of our framework, called PRODPROC, is to natively support features for modeling production process aspects otherwise difficult to describe. Such features may lead to the definition of reacher configurable product models, that allow the propagation of consequences of product configuration decision toward the planning of its production process, and the propagation of consequences of process planning decision toward the product configuration. Moreover, we want our graphical framework to allow one to easily define mixed product/process models, without the need to know how the configuration systems works "under the hood".

* Student
** Supervisor

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 7–12, Perugia, Italy, 12 September 2011.

2 The PRODPROC Modeling Framework

In this section, we present the main modeling features of PRODPROC. Also, we outline a brief description of PRODPROC semantics in terms of model instances. [1]

Modeling Products We are interested in modeling configurable products whose corresponding (producible) variants can be represented as trees. Nodes of these trees correspond to physical components, whose characteristics are all determined. The tree structure describes how the single components taken together define a configured product. Hence, we model a configurable product as a multi-graph, called *product model graph*, and a set of constraints. A product model represents a configurable product. Its configuration can lead to the definition of different (producible) product variants.

Nodes of the product model graph represent well-defined components of a product (e.g., the frame of a bicycle). A node is characterized by a name, a set of variables representing configurable features of the component (e.g., the material of which the bicycle frame is made of), and a set of constraints involving variables of the node as well as variables of its ancestors in the graph. Each variable is endowed with a finite domain (typically, a finite set of integers or strings), i.e., the set of its possible values. In the description of a configured product, physical components are represented as instances of nodes in the graph. The graphical representation of a node (cf. Fig. 1) consists of a box with three sections, each containing one of the elements constituting a node.

The edges model *has-part*/*is-part-of* relations between product components (e.g., between the frame and the handlebar of a bicycle). An edge is defined by: a name, two node names indicating the parent and the child nodes in the *has-part* relation, the cardinality of such relation (expressed as either an integer number or a variable), and a set of constraints. Such constraints may involve the cardinality variable (if any) as well as the variables of the parent node and of any of its ancestors. An instance of an edge connecting two nodes is an edge connecting two instances of such nodes. An edge is graphically represented by an arrow connecting the parent node to the child node (cf. Fig. 1). Such an arrow is labeled with the edge name and cardinality, and may have attached an ellipse containing cardinality constraints. We require the presence of a node without entering edges in the product model graph. We call this node *root node*. It will have only one instance, such instance will be the root of the configured product tree.

As mentioned, a product description consists of a product model graph together with a set of global constraints. Such constraints, called *model constraints*, involve variables of nodes not necessary related by *has-part* relations (*node model constraints*) as well as cardinalities of different edges exiting from a node (*cardinality model constraints*). Also, node model constraints can be defined using arithmetic constraints, propositional constraints, and global constraints like `alldifferent` and aggregation constraints.

Modeling Processes In general, a process can be represented in terms of activities and temporal relations between them. We characterize a process by: a set of activities, a set

[1] Further examples and details on PRODPROC can be found in `www.dmi.unipg.it/formis/papers/report2011_04.ps.gz`.

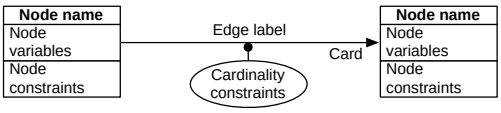


Fig. 1: Graphical representation of nodes and edges.

of variables (as before, endowed with a finite domain of strings or of integers) representing process characteristics and involved resources; a set of temporal constraints between activities; a set of resource constraints; a set of constraints on activity durations. A process model does not represent a single production process. Instead, it represents a configurable production process, whose configuration can lead to the definition of different executable processes.

PRODPROC defines three kinds of activity: *atomic activities*, *composite activities*, and *multiple instance activities*. An *atomic* activity A is an event that happens in a time interval. It has associated a name, two integer decision variables denoting the start time and end time of the activity, a decision variable denoting the duration of the activity, and a flag indicating whether or not the activity is executed. A composite activity is an event described in terms of a process. Hence, it has associated four variables analogously to an atomic activity. Moreover, it is associated with a model of the process it represents. A *multiple instance* (atomic or composite) activity is an event that may occur multiple times. Together with the four variables (and possibly the process model), a multiple instance activity has associated a decision variables representing the number of times the activity can be executed. Figures 2a, 2b, and 2c, show the graphical representation of atomic activities, composite activities, and multiple instance activities, respectively.

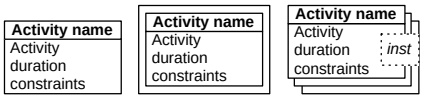
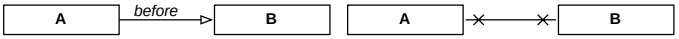


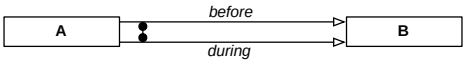
Fig. 2: Graphical representation of activities.

Temporal constraints between activities are inductively defined starting from *atomic temporal constraints*, using logical operators like conjunction and disjunction. We consider as atomic temporal constraints all the thirteen binary relations which capture all the possible ways in which two time intervals might overlap or not (as introduced by Allen in [2]), and some constraints inspired by the constraint templates of the language ConDec [8]. Fig. 3 shows the graphical representation of some temporal constraints.

PRODPROC allows one to specify constraints on resource amounts [7] and activity durations. A *resource constraint* is a quadruple $\langle A, R, q, TE \rangle$, where A is an activity; R is a variable endowed with a finite integer domain; q is an integer or a variable endowed with a finite integer domain, defining the quantity of resource R consumed (if $q < 0$) or produced (if $q > 0$) by executing A; TE is a time extent that defines the time interval where the availability of resource R is affected by the execution of ac-



(a) A is executed before B. (b) Either A or B is executed.



(c) A is executed before B *or* A is executed during the execution of B.

Fig. 3: Graphical representation of temporal constraints.

tivity A. The possibilities for TE are: $FromStartToEnd$, $AfterStart$, $AfterEnd$, $BeforeStart$, $BeforeEnd$, $Always$, with their obvious meaning. Another form of resource constraint defines initial level constraints, i.e., expressions determining the quantity of a resource available at the time origin of a process. The basic form is $initialLevel(R, iv)$, where R is a resource and $iv \in \mathbb{N}$. Fig. 4 shows the graphical representation for a resource R and the constraints $\langle A, R, q_A, TE_A \rangle$, $\langle B, R, q_B, TE_B \rangle$, and $initialLevel(R, iv)$, where $q_A > 0$ and $q_B < 0$.

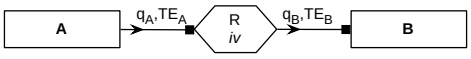


Fig. 4: Graphical representation of resource constraints.

An *activity duration constraint* for the activity A is a constraint involving the duration of A, process variables, and resource quantity variables related to A.

PRODPROC also allows one to couple elements for modeling a process and elements for modeling a product through constraints involving process variables and product variables. In general, constraints involving both product and process variables may help to detect/avoid planning impossibilities due to product configuration, and configuration impossibilities due to production planning, during the configuration of a product. For example, they can allow one to detect the impossibility to produce a product due to the lack of a certain resource needed by an activity.

PRODPROC Models and Instances A PRODPROC *model* consists of a product description, a process description, and a set of constraints coupling the two. It represents a collection of single (producible) variants of a configurable product, and describes the processes needed to produce them. A PRODPROC *instance* represents one of such variants and its production process. To define the notion of instance we need to introduce first the auxiliary notion of *candidate instance*. A PRODPROC candidate instance consists of the following components: a tree, called *instance tree*, defined by instances of nodes and edges in the product model graph; a set of assignments for node instance variables; a set of activity instances, i.e., activities that will be executed; a set of assignments for all process model variables and activities parameters. A PRODPROC instance is a candidate instance such that the assignments to product and process variables and

parameters satisfy all the conditions specified through constraints in the PRODPROC model (i.e., node constraints, temporal constraints, etc.). Intuitively, such constraints, being expressed at the PRODPROC model level (i.e., on model variables), have to be reflected onto the PRODPROC instance (i.e., on instance variables). This is obtained by applying an appropriate *instantiation mechanism* that, given a model and a (partial) candidate instance (a candidate instance is partial when there are variables with no value assigned to), generates a set of constraints defined on variables of the candidate instance from each constraint in the model.

3 CLP-based Product/Process Configuration

On top of the PRODPROC framework it is possible to implement a full-fledged configuration system. In particular, the graphical language presented in Sect. 2 can allow a user to easily define product/process models, and Constraint Logic Programming (CLP) [5] can be exploited to implement an interactive algorithm that, given a PRODPROC model, guide a user through the configuration process to obtain a PRODPROC instance.

From a PRODPROC model and a user defined (partial) candidate instance, it is possible to obtain a Constraint Satisfaction Problem (CSP) $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ where: $\mathcal{V}$ is the set of all the variables appearing in the (partial) candidate instance; $\mathcal{D}$ is the set of domains for variables in $\mathcal{V}$; $\mathcal{C}$ is the set of constraints in the PRODPROC model instantiated on variables of the (partial) candidate instance. Given such a CSP, a finite domain solver can be used to reduce domains associated with variables, preserving satisfiability, or to detect the inconsistency of the encoded CSP (due to user's assignments that violate the set of constraints or to inconsistencies of the original product model). Moreover, it can be used to detect that changes are needed in the topology of the instance tree.

We are using SWI-Prolog to develop a CLP-based configuration system that exploit the close relation between configuration problems and CSPs.[2] In particular, we are using the SWI-Prolog `pce` library to implement the system graphical user interface, and the `clpfd` library for constraint propagation and labeling purposes. The current version of the system is limited to product modeling. It allows a user to define a product description using the PRODPROC graphical language, to check model syntactic correctness, and to automatically generate product instances to check model validity. Our system will support a configuration process as follows. First, the user initializes the system (1) selecting the model to be configured. After such an initialization phase, the user starts to make her choices by using the system interface (2). The interface communicates to the system engine, i.e., the piece of software that maintains a representation of the product/process under configuration and checks the validity and consistency of user's choices, each data variation specified by the user (3). Whenever an update of the (partial) configuration takes place, the user, through the system interface, can activate the engine inference process (4). The engine instantiates PRODPROC constraints on the current (partial) candidate instance defined by user choices, and encodes the product/process configuration problem into a CSP. Then, the engine uses the SWI-Prolog finite domain solver to propagate CSP constraints, i.e., to compute the logical effects of

[2] We chose CLP instead of Constraint Programming for the advantages the former gives in terms of rapid software prototyping.

user's choices (5). Once the inference process ends (6), the engine returns to the interface the results of its computation (7). In its turns, the system interface communicates to the user the consequences of her choices on the (partial) configuration (8). The user can then make new choices, or modify previous ones if an inconsistency has been reported.

4 Conclusions

In this paper, we presented the first results of an ongoing work on product and process modeling and configuration, and pointed out the the lack of a tool covering both physical and production aspects of configurable products. To cope with this absence, we presented a framework called PRODPROC. Furthermore, we outlined how it is possible to build a full-fledged CLP-based configuration system on top of it.

The PRODPROC framework presents features that, to the best of our knowledge, are not present in other existing product or process modeling languages. These are, for example, product model graphs, model constraints, resource variables and resource constraints, activity duration constraints. Moreover, all these innovative features belong to a single framework that allows one to model products, their production processes, and to couple products with processes using constraints. With respect to the modeling language presented in [1], PRODPROC is far more complex and expressive.

We already implemented a first prototype of a CLP-based configuration system that uses PRODPROC. It covers only product modeling and configuration, but we are working to add to it process modeling and (automatic) configuration capabilities. We also plan to experiment our configuration system on different real-world application domains, and to compare it with commercial products, e.g., [3].

References

1. M. Aldanondo and E. Vareilles. Configuration for mass customization: how to extend product configuration towards requirements and process configuration. *J. of Intelligent Manufacturing*, 19(5):521–535, 2008.
2. J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26:832–843, 1983.
3. U. Blumöhr, M. Münch, and M. Ukalovic. *Variant Configuration with SAP*. SAP Press, 2009.
4. Configit A/S. Configit Product Modeler. `http://www.configit.com`.
5. J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994.
6. U. Junker. The Logic of ILOG (J)Configurator: Combining Constraint Programming with a Description Logic. In *Proc. of the IJCAI'03 Workshop on Configuration*, pages 13–20. 2003.
7. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artif. Intell.*, 143:151–188, 2003.
8. M. Pesic, H. Schonenberg, and W. van der Aalst. DECLARE: Full support for loosely-structured processes. In *Proc. of EDOC'07*, pages 287–287, 2007.
9. D. Sabin and R. Weigel. Product configuration frameworks-a survey. *IEEE Intelligent Systems*, 13:42–49, July 1998.
10. Tacton Systems AB. Tacton Configuration Studio. `http://www.tacton.com`.
11. A. H. M. ter Hofstede, W. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation - YAWL and its Support Environment*. Springer, 2010.

Explaining ßow-based propagation

Nicholas Downing, Thibaut Feydy, and Peter J. Stuckey

National ICT Australia and the University of Melbourne, Victoria, Australia
{ndowning@students.,tfeydy@,pjs@}csse.unimelb.edu.au

Abstract. Lazy clause generation is a powerful approach to reducing search in constraint programming. For use in a lazy clause generation solver, global constraints must be extended to explain themselves. Alternatively they can be decomposed into simpler constraints which already have explanation capability. In this paper we present a new generic flow-based propagator with explanation capability, and show how it can be applied to *gcc* and *sequence* constraints. We compare different *gcc* and *sequence* implementations on several real-world problems to determine how explanation changes the trade-offs for propagation and search.

1 Introduction

Lazy clause generation [9] is a hybrid approach to constraint solving that combines features of finite domain propagation and Boolean satisfiability. By creating an implication graph recording the reasons for propagation we can reduce search by deriving and propagating *nogoods* to avoid repeating failed searches.

gcc and *sequence* are two of the most important global constraints. They occur frequently in scheduling and rostering problems. Earlier work has shown flow-based propagation to be useful for these constraints [10, 3]. Explanations have been described for *gcc* [6], but not *sequence*. We generalize this work to other constraints, e.g. *sequence* and the problem-specific *gsc* [11]. Our experiments verify that our methods are useful, and also provide some interesting comparisons to see how explanations affect propagation methods and search strategies.

2 Lazy clause generation

We give a brief description of propagation-based solving and lazy clause generation, for more details see [9]. We consider constraint satisfaction problems, consisting of constraints over integer variables $x_1, \ldots, x_n$, each with a given finite domain $D_{\text{orig}}(x_i)$. A feasible solution is a valuation to the variables such that each x_i is within its allowable domain and all constraints are satisfied.

A propagation solver maintains a domain restriction $D(x_i) \quad D_{\text{orig}}(x_i)$ for each variable and considers only solutions that lie within $D(x_1) \times \quad \times D(x_n)$. Solving interleaves propagation, which repeatedly applies propagators to remove unsupported values, and search which splits the domain of some variable and considers the resulting sub-problems. This continues until all variables are fixed (success) or failure is detected (backtrack and try another subproblem).

Lazy clause generation is implemented in the above framework by defining an alternative model for the domains $D(x_i)$, which is maintained simultaneously.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 13–18, Perugia, Italy, 12 September 2011.

Specifically, Boolean variables are introduced for each potential value of a variable, named $[\![x_i = j]\!]$ and $[\![x_i \geq j]\!]$. Negating them gives the opposite, $[\![x_i \neq j]\!]$ and $[\![x_i \leq j-1]\!]$. Fixing such a *literal* modifies D to make the corresponding fact true in $D(x_i)$ and vice versa. Hence these literals give an alternate Boolean representation of the domain, which can support SAT reasoning.

In a lazy clause generation solver, the actions of propagators (and search) to change domains are recorded in an *implication graph* over the literals. Whenever a propagator changes a domain it must *explain* how the change occurred in terms of literals, that is, each literal l that is made true must be explained by a clause $L \rightarrow l$ where L is a (set or) conjunction of literals. When the propagator causes failure it must explain the failure as a *nogood*, $L \rightarrow \mathit{false}$, with L a conjunction of literals which cannot hold simultaneously. Conflict analysis reduces L to a form suitable to use as a clausal propagator to avoid repeating the same search [8].

3 Flow networks

A *flow network* is a graph (V, E) which models a system where flow is conserved, with vertices/nodes V and edges $E = \{(u, v) : \text{there is a directed arc } u \rightarrow v\}$. Flow in the graph is represented by a vector $\mathbf{f}$ with bounds $\mathbf{l}, \mathbf{u}$ such that $l_{uv} \leq f_{uv} \leq u_{uv}$. If the flow network is a *circulation* then flow is strictly conserved, that is *outflows* $-$ *inflows* $= 0$ at each node, but for convenience we allow a constant vector $\mathbf{s}$ of supplies (or demands, if negative) per node, giving

$$\forall n \in V, \sum_{v \in V:(n,v)\in E} f_{nv} - \sum_{u \in V:(u,n)\in E} f_{un} = s_n. \tag{1}$$

Example 1. Figure 1 shows a simple flow network with nodes representing nurses (x = Xavier, y = Yasmin), shifts (d = day, n = night), and a sink t. A feasible (integer) assignment to $\mathbf{f}$ gives a solution to the problem of rostering the nurses with 1 or 2 nurses on day shift and 0 or 1 nurses on night shift, where $f_{ij} = 1$ if nurse i works shift j, 0 otherwise. A nurse i works only one of the shifts due to flow conservation at his/her node, $f_{i\mathsf{d}} + f_{i\mathsf{n}} = 1$. The number of nurses on shift j is $f_{j\mathsf{t}}$ due to flow conservation at its node, $f_{\mathsf{x}j} + f_{\mathsf{y}j} = f_{j\mathsf{t}}$, with the staffing requirement expressed as the bounds on $f_{j\mathsf{t}}$. This illustrates Régin's [10] encoding of the constraint $\mathit{gcc}([x, y], [1..2, 0..1])$, with $x, y = 1$ (day) or 2 (night), and flows e.g. f_{xd} being integer views e.g. $\mathit{bool2int}([\![x = 1]\!])$ of domain literals.

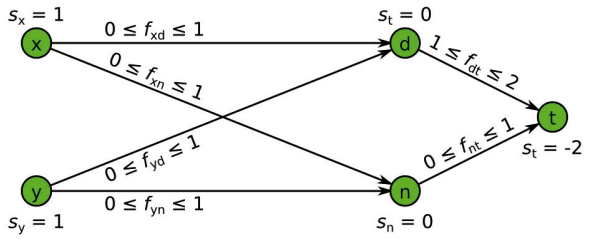


Fig. 1: Example flow network encoding a *gcc* constraint

4 Circulation propagator

We define a new constraint $\mathit{circulation}(V, E, \mathbf{s}, \mathbf{f})$ which enforces the flow conservation constraints (1) on $\mathbf{f}$ according to the graph (V, E) and supplies $\mathbf{s}$, where $l_{uv}, u_{uv} = \min, \max D(f_{uv})$. To propagate, we first solve for a feasible flow, using artificial supplies or demands, which can be cancelled by augmenting paths [4].

Suppose there is no feasible solution. Let C, the 'cut', be the set of nodes searched. It must contain node(s) in excess but none in deficit. Then according to the current flow bounds, more flow enters C than can leave it, taking into account the arcs crossing C and the net supply/demand of C. Summing the flow conservation equations (1) over $n \in C$ gives flow conservation for the cut,

$$\sum_{(u,v)\in(C\times V\setminus C)\cap E} f_{uv} - \sum_{(u,v)\in(V\setminus C\times C)\cap E} f_{uv} = \sum_{n\in C} s_n. \tag{2}$$

Given C that proves infeasibility, we explain (2) as a bounds-consistent *linear* constraint [9], i.e. even if outflows are at maximum for outgoing arcs and inflows are at minimum for incoming arcs, maximizing the RHS of (2), the RHS is still $< \sum_{n\in C} s_n$. The failure nogood is the conjunction of literals $[\![f_{uv} \geq l_{uv}]\!]$ for inflows and $[\![f_{uv} \leq u_{uv}]\!]$ for outflows, substituting the current $\mathbf{l}, \mathbf{u}$.

Example 2. Continuing Example 1, suppose search sets $f_{\mathsf{xd}} = f_{\mathsf{yd}} = 0$, equivalently $x, y \neq 1$, so that insufficient nurses are available for day shift. Figure 2 shows the residual graph of a partial solution with flows in range but not conserved. Attempting to resolve the excess, BFS explores nodes $C = \{\mathsf{x}, \mathsf{d}, \mathsf{y}\}$. Cut-conservation gives $\mathit{bool2int}([\![x = 1]\!]) + \mathit{bool2int}([\![y = 1]\!]) + f_{\mathsf{nt}} = 2$, which is unachievable since both literals are *false* and $f_{\mathsf{nt}} \leq 1$. Hence the *circulation* propagator fails with nogood $[\![x \neq 1]\!] \wedge [\![y \neq 1]\!] \wedge [\![f_{\mathsf{nt}} \leq 1]\!] \rightarrow \mathit{false}$.

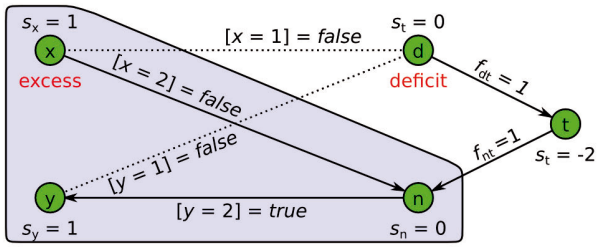


Fig. 2: Example residual graph showing infeasibility of the *gcc* constraint

In case of feasibility, pruning is attempted, by a generalization of Régin's *gcc* algorithm [10], based on Strongly Connected Components (SCCs), to arbitrary networks. We use SCC-splitting [5], and we generate explanations lazily.

Example 3. Continuing Example 1, suppose search sets $y = 2$, then the network is feasible with $x = 1$. There are no augmenting cycles so each node is a separate SCC. Arc (x, n) spans SCCs, so $C =$ the target SCC $\{\mathsf{n}\}$ is used as a cut, giving $f_{\mathsf{nt}} - \mathit{bool2int}([\![x = 2]\!]) - \mathit{bool2int}([\![y = 2]\!] = 0$, unachievable with both literals *true*, as $f_{\mathsf{nt}} \leq 1$. Hence $[\![f_{\mathsf{nt}} \leq 1]\!] \wedge [\![y = 2]\!] \rightarrow [\![x \neq 2]\!]$, pruning arc (x, n).

For *gcc* the explanations are the same as Katsirelos's [6] except that we can deduce and propagate equalities (rather than just disequalities) which is more succinct in certain cases e.g. the *alldifferent* explanations $[\![x = v]\!] \rightarrow [\![y \neq v]\!]$.

5 New *sequence* and *gsc* encodings

The *sequence* constraint takes the form $sequence(l, u, w, [y_1, \ldots, y_n])$ and says that every consecutive w-window must sum to $l..u$. We give a new flow-based encoding for *sequence*, as a flow network, similar to [7] but simpler and using fewer arcs. Referring to Figure 3, a flow f_i along the spine corresponds to a sum of y_i over some w-window, which we may show by a series of cuts, e.g. the cut illustrated shows by cut-conservation that $f_3 = y_3 + y_4 + y_5$. Constraining the f_i-flows to $l \leq f_i \leq u$ enforces *sequence*. Our *circulation* propagator ensures domain-consistency on y_i provided they are 0..1 valued (the common case).

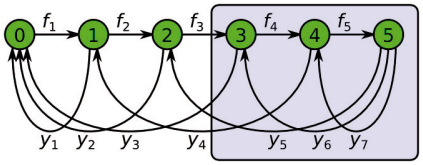


Fig. 3: Flow network encoding a $w = 3, n = 7$ *sequence* constraint

Régin and Puget's $gsc(l, u, w, [x_1, \ldots, x_n], [(v_1, c_1), \ldots, (v_m, c_m)])$ says that $x_i \in \{v_1, \ldots, v_m\}$ occurs $l..u$ times per w-window and that $x_i = v_j$ occurs c_j times overall [11]. They reduce it to *gcc*, we give an equivalent, but simpler, direct encoding that requires no side constraints. Referring to Figure 4, nodes x_i, v_j represent variables and values as in a standard *gcc* network. Nodes w_k ensure that $x_k, \ldots, x_{k+w-1}$ meet the $l..u$ constraint by setting the flow from the overall source s to those variable nodes. As the windows w_k do not overlap, there are w different window alignments, hence w *circulation* propagator instances.

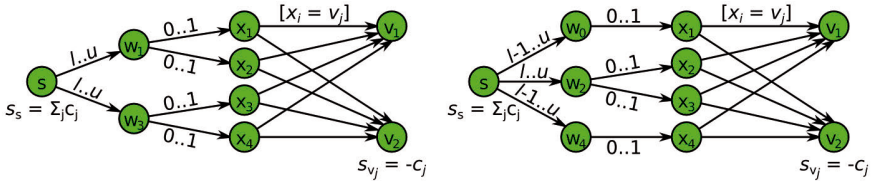


Fig. 4: The w flow networks encoding a $w = 2, n = 4, m = 2$ *gsc* constraint

6 Experiments

We implemented the *circulation* constraint in *Chuffed*, the state-of-the-art lazy clause generation solver. We tried it on CS: car sequencing (prob001 in CSPLib), instances from main 79-instance set, and NR: nurse rostering, models 1 and 2 of [3], instances based on the first 50 instances from the N30 data set in NSPLib. Hardware was a cluster of Dell PowerEdge 1950 with 2 × 2.00GHz Intel Quad Core Xeon E5405, 2×6MB Cache, and 16 GB RAM. Timeouts were 1800s. Data files are available from `http://www.csse.unimelb.edu.au/~pjs/flow`.

Both problems use *gcc* and *sequence* as the only constraints, except that CS has *table* constraints to decode the set of options required for each car class, and NR has clausal constraints that force a nurse to take 2 consecutive days on any given shift, and lexical symmetry breaking between the nurses' rosters. Both use multiple *sequence* constraints; CS has a single *gcc* to set the production per car type, whereas NR has a *gcc* per day for the staffing levels. For CS we also tried *gsc*, which is applied once per option and subsumes all other constraints.

$gcc([x_1, \ldots, x_n], [c_1, \ldots, c_m])$ is implemented as LD: decomposition into *linear* constraints $\sum_{i=1}^{n} bool2int([\![x_i = j]\!]) = c_j \quad j \quad 1..m$, or DF: Régin's domain-consistent flow-based encoding using our new *circulation* propagator.

$sequence(l, u, w, [y_1, \ldots, y_n])$ is implemented as CD: 'cumulative' decomposition into partial sums $s_i = \sum_{j=1}^{i} y_j \quad i \quad 0..n$ and differences $l \leq s_{i+w} - s_i \leq u \quad i \quad 0..n - w$ (both implemented as *linear* constraints), RD: *regular* decomposition into *table* constraints over allowable state change tuples (q_{i-1}, y_i, q_i) and thence to SAT, or DF: the new domain-consistent flow-based encoding.

The search strategy is IO: input order, an appropriate static search which for CS is based on the ideas of [12] and for NR rosters each nurse before moving forward one day, DWD: dom/wdeg [2], ACT: activity-based (VSIDS) search [8].

Table 1 reports the geometric mean of runtimes (using the timeout for timed-out instances), with the number of timeouts appearing as a superscript. The heading line shows how many solved instances were unsatisfiable or satisfiable and how many were indeterminate as not solved by any solver. These latter instances aren't included in the rest of the table. In each block the solver with the fewest timeouts is highlighted, with ties being broken by the runtimes.

		CS: car scheduling, unsat=0 sat=73 ?=6				NR: nurse rostering, unsat=59 sat=37 ?=4			
		NOLEARN		LEARN		NOLEARN		LEARN	
		gcc=LD	DF	*gcc*=LD	DF	*gcc*=LD	DF	*gcc*=LD	DF
IO	*seq*=CD	149.71s^{36}	151.60s^{36}	77.03s^{31}	92.38s^{34}	1297.53s^{88}	1256.51s^{88}	32.57s^{19}	79.98s^{20}
	RD	10.81s^{27}	11.95s^{27}	6.52s^{25}	11.88s^{29}	1090.06s^{84}	1049.24s^{83}	14.69s^{18}	39.37s^{17}
	DF	7.01s^{24}	8.14s^{26}	10.41s^{22}	20.76s^{28}	**890.48s**^{82}	918.25s^{82}	**2.38s**^{15}	5.69s^{15}
	GSC	**0.52s**^{6}		**0.47s**^{5}					
DWD	*seq*=CD	149.50s^{35}	153.37s^{36}	11.67s^{4}	14.89s^{4}	1524.06s^{92}	1525.60s^{91}	**14.15s**^{12}	28.79s^{17}
	RD	12.59s^{28}	13.82s^{28}	0.14s^{3}	0.15s^{3}	1384.28s^{86}	1013.71s^{78}	70.91s^{25}	158.05s^{40}
	DF	8.84s^{27}	10.61s^{28}	0.61s^{5}	0.89s^{4}	11.01s^{46}	**11.86s**^{45}	0.80s^{10}	1.34s^{13}
	GSC	**0.43s**^{4}		**0.45s**					
ACT	*seq*=CD			**19.36s**^{5}	22.89s^{5}			18.87s^{6}	30.89s^{24}
	RD			591.42s^{45}	774.01s^{56}			12.46s^{9}	13.14s^{20}
	DF			1490.41s^{67}	1725.10s^{67}			1.48s^{4}	**1.80s**^{1}
	GSC			1028.57s^{46}					

Table 1: Our methods versus standard *gcc* and *sequence* implementations

7 Conclusions and further work

Learning changes the tradeoffs for propagation. Strategy DWD allows the best comparison. For the *sequence* in CS, the strong propagator DF was better than RD without learning but the decomposition was better with learning. Similarly, for the *gcc* in NR, the strong propagator DF was better without learning but the decomposition LD was better with learning. This suggests that learning can often recover some of the global knowledge lost through decomposition.

Learning also changes the tradeoffs for search. Dynamic search DWD was slightly worse than static search IO without learning but usually much better with learning, suggesting that DWD drives the search towards nogood reuse. NR with ACT is even better, with domain-consistent *gcc* becoming useful again, suggesting enhanced local reuse of the longer and more specific nogoods produced under domain consistency. For CS, ACT is the wrong strategy as the *sequence* constraints are tight, so any gaps in the schedule are impossible to resolve later on, whereas DWD tends to propagate the search linearly within the schedule.

For NR we can close more instances than previous standard methods in less time, showing the value of adding explanations to the standard flow-based propagators. For CS, given that we cannot use ACT, our *gcc* and *sequence* constraints were not as good as the standard decompositions LD and RD respectively, but flow-based explanations are still applicable, since the problem-specific *gsc* with learning closes more instances than standard methods, and shows the usefulness of explanations plus genericity for rapid implementation of new constraints.

For future work on soft constraints we also have a similar propagator based on Dual Network Simplex, essentially an explained version of Steiger's [13], with the explanations being derived from Benders' infeasibility and objective cuts [1].

References

1. Benders, J.F.: Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik 4, 238–252 (1962)
2. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting Systematic Search by Weighting Constraints. In: Procs. of ECAI04. pp. 146–150 (2004)
3. Brand, S., Narodytska, N., Quimper, C.G., Stuckey, P., Walsh, T.: Encodings of the SEQUENCE constraint. In: CP. Volume 4741 of LNCS. pp. 210–224 (2007)
4. Ford, L., Fulkerson, D.: Maximal flow through a network. Canad. J. Math. 8, 399–404 (1956)
5. Gent, I., Miguel, I., Nightingale, P.: Generalised arc consistency for the AllDifferent constraint: An empirical survey. AI 172(18), 1973 – 2000 (2008)
6. Katsirelos, G.: Nogood processing in CSPs. Ph.D. thesis, University of Toronto, Canada (2008)
7. Maher, M., Narodytska, N., Quimper, C.G., Walsh, T.: Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In: Procs. of CP08. pp. 159–174 (2008)
8. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Procs. of DAC01. pp. 530–535 (2001)
9. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints 14, 357–391 (2009)
10. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: Procs. of AAAI96. pp. 209–215 (1996)
11. Régin, J.C., Puget, J.F.: A filtering algorithm for global sequencing constraints. In: Smolka, G. (ed.) Procs. of CP97, vol. 1330, pp. 32–46 (1997)
12. Smith, B.: Succeed-first or Fail-first: A Case Study in Variable and Value Ordering. In: Procs. of PACT97. pp. 321–330 (1997)
13. Steiger, R., van Hoeve, W.J., Szymanek, R.: An efficient generic network flow constraint. In: Procs. of SAC11. pp. 893–900 (2011)

Mapping Out the Tractability of 3-Variable Forbidden Patterns *

Martin C. Cooper, supervisor Guillaume Escamocher, PhD student

Abstract

Identifying the exact frontier between tractable and intractable classes of Constraint Satisfaction Problems (CSP) is a fundamental problem in complexity theory. Going beyond tractable classes defined by restrictions either on the language of constraint relations or on the structure defined by the set of constraint scopes, we study classes of CSP instances defined by forbidding subproblems. This approach has already led to the identification of properties defining new tractable classes, such as the broken-triangle property (BTP) [1], which generalises tree-structured instances, and the crisp version of the joint-winner property (JWP) [2] which generalises a set of non-overlapping AllDiff constraints. As a first step in the search for new tractable classes defined by forbidden subproblems, we have begun a systematic study of the tractability of classes of binary CSP instances defined by forbidding 3-variable patterns. We present a set of possible supports for tractable patterns, which can be viewed as a necessary condition for a pattern to be tractable, and which greatly reduces the number of open cases left to study. We also give a characterization of the complexity of all (but two) patterns which satisfy some given properties on their size.

Introduction

A pattern is a partially-specified subproblem. We say that a pattern A is *tractable* if there exists a polynomial-time algorithm to solve the class of CSP instances CSP($\overline{A}$) in which A does not occur; A is *intractable* if CSP($\overline{A}$) is NP-complete. If a pattern is not known to be tractable or intractable, then we say that it is *open*. We consider 3-variable patterns in binary CSPs, since this is the first case which goes beyond classes defined by properties of individual constraints and since interesting 3-variable tractable patterns have already been discovered [1, 2]. Since binary CSP with size-3 domains is NP-complete by reduction from 3-SAT, we only need consider forbidding patterns with at most three distinct values in each domain. We denote by V the set of all variables in the instance. For all $v \in V$, we denote by D_v the domain of v. A *point* is a variable-value assignment. Where there is no possible ambiguity we will also use D_v to represent the set of points involving variable v. For simplicity of presentation, we assume that there is a unique binary constraint between every pair of variables. In a CSP instance, two points a, b are compatible (incompatible) if making this pair of assignments satisfies (does not satisfy) the corresponding binary constraint. We can consider a pattern as a generic CSP instance in which certain compatibilities are left unspecified, and a CSP instance can be viewed as a pattern in which all compatibilities are specified.

By convention, in a pattern all variables are assumed to be distinct. However, two points $a, b \in D_v$ may actually represent the same variable-value assignment (unless one is compatible and the other incompatible with some assignment c to another variable v').

Since the patterns contain at most three variables, and that there are at most three possible assignments for each of these variables, a pattern can contain at most 27 edges. Since an edge can be of three possible kinds (compatible edge, incompatible edge, no edge), the number of patterns to study is 3^{27}.

*supported by ANR Project ANR-10-BLAN-0210.

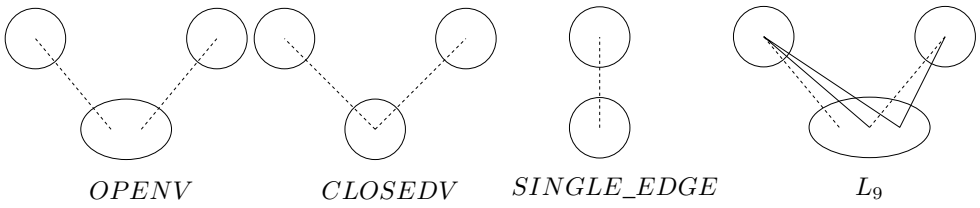
Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 19–24, Perugia, Italy, 12 September 2011.

We start in Section 1 by defining the graphical representation used to describe patterns. Then in Section 2 we introduce a new tool we call reduction, which will subsequently allow us to restrict the number of patterns to study to a reasonable number. Combining this new tool and already known properties, we present our results in the two following sections. We build in Section 3 a set of patterns which form the support of all possible tractable patterns. We then give in Section 4 an exhaustive characterisation of complexity (with two exceptions) of patterns with three or less compatible edges. We finally summarize our results in the conclusion.

1 Graphical Representation

The domains of the variables are represented by circles. Possible assignments for a given variable v are represented by points inside the circle representing D_v. If two assignments are known to be compatible, then we draw a continuous line between them. If two assignments are known to be incompatible, then we draw a dashed line between them. If we don't have any information on the compatibility of two assignments, then we don't draw any line between them.

Examples:



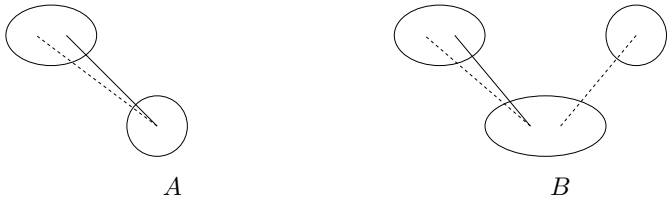
OPENV, CLOSEDV and SINGLE_EDGE are subpatterns which occur in many tractable patterns. The pattern L_9 is one of the patterns whose tractability we demonstrate in section 4. The class it defines is a generalization of the already known tractable class zero-one-all [6].

2 Reduction

In a pattern A, a point a which is linked by a single compatible edge to the rest of A is known as a *dangling point*. If an arc consistent instance I does not contain the pattern A then it does not contain the pattern A' which is equivalent to A in which the dangling point a and the corresponding compatibility edge have been deleted. Thus, since arc consistency is a polynomial-time operation which cannot introduce a forbidden pattern, to decide tractability we only need consider patterns without dangling points.

Definition 1. *We say that a pattern A can be reduced to a pattern B, and that B is a reduction of A, if one of the following is true:*

1. *A is embedded in B. Example:*



2. *Fusioning two points in A transforms A into B. Example:*

A B

3. *Adding a dangling point and its corresponding compatibility edge to B transforms B into A. Example:*

A B

4. *A can be reduced to a pattern C, such that B is a reduction of C.*

Definition 2. *Let $k \geq 0$ and P be a pattern with k or less compatible edges. We say that P is a k-final tractable pattern if P is tractable and cannot be reduced to a different tractable pattern with k or less compatible edges.*

Lemma 1. *Let A and B be two patterns, such that A can be reduced to B. Let I be a CSP instance satisfying arc consistency. If B occurs in I, then A also occurs in I.*

Proof. We suppose B occurs in I. If A is embedded in B, then the result is immediate. If fusioning two points a and b in A transforms it into B, then A actually covers two different patterns: the one where a and b are different points, and the one where a and b are the same point. The latter pattern is B. So the set of instances containing B is a subset of the set of instances containing A and we have the result. If adding a dangling point and its corresponding compatibility edge to B transforms it into A, then since I satisfies arc consistency A also occurs in I. If A can be reduced to a pattern C such that B is a reduction of C, then by induction C and A also occur in I. □

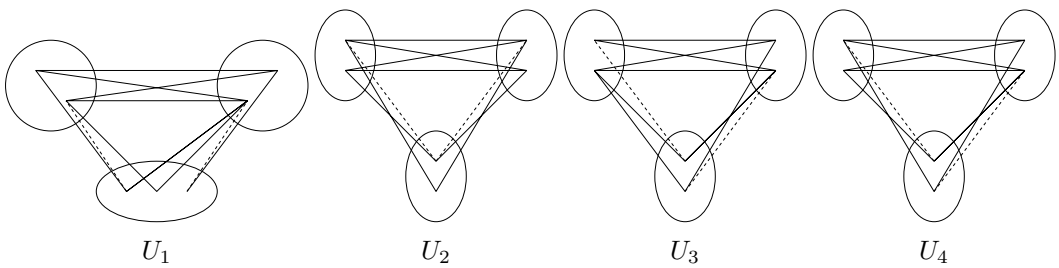
Corollary 1. *Let A and B be two patterns, such that A can be reduced to B. Then*

- *If B is tractable, then A is tractable.*
- *If A is NP-Complete, then B is NP-Complete.*

Therefore, we can use the reduction tool to study only patterns which cannot be reduced to a known tractable pattern, or which are not the reduction of a known NP-Complete pattern.

3 Necessary Conditions for Tractability

Let $\mathcal{U} = \{U_1, \ldots, U_4\}$



U_1 U_2 U_3 U_4

A pattern on 3 variables with three or more incompatible edges is NP-Complete [3]. A pattern with two or more distinct incompatible edges between the same couple of domains is also NP-Complete [3]. Consequently, if a 3-variable pattern is tractable, then the subpattern formed by its incompatible edges can be reduced to either CLOSEDV or OPENV. So the only unknown part in a tractable pattern are the compatibility edges. We have the following Lemma:

Lemma 2. *Let P be a tractable pattern. Then P can be reduced to a pattern belonging to U.*

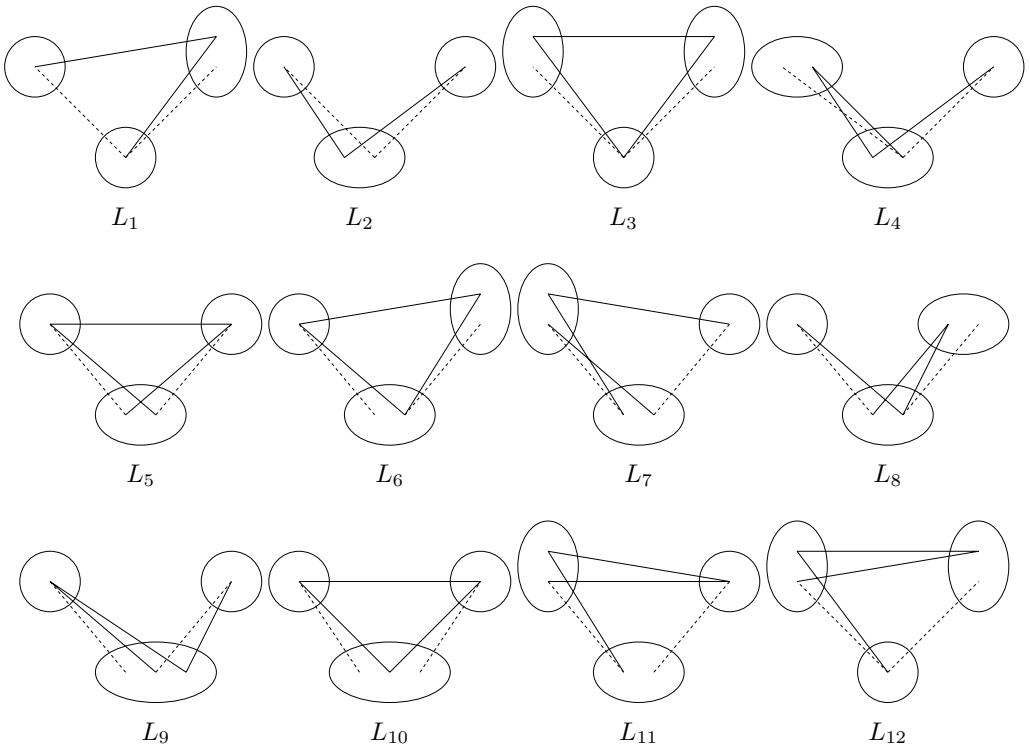
Proof. For space reasons, we cannot include the entire proof. The proof consists of showing first that any pattern which is not embedded in one of the patterns in $\mathcal{U}$ contains one of four given patterns. Then we show these four patterns are all NP-Complete. Since a pattern containing a NP-Complete subpattern is NP-Complete, we have the result. □

This result shows that the only tractable patterns are the ones which are reducible to a pattern in $\mathcal{U}$. By Lemma 1, we only need to study the subpatterns of the patterns in $\mathcal{U}$. There are 4 patterns in $\mathcal{U}$, each one having either 10 or 11 edges. So the number of patterns to study is bounded by $4 \times 2^{11} = 2^{13}$. However, $\forall 1 \leq i \leq 4$, $\forall e$ where e is an incompatible edge in U_i, it is easy to show that $U_i \backslash e$ is reducible by arc consistency and fusion to the trivial tractable pattern SINGLE_EDGE. So the only non-trivial subpatterns to study are the ones which have both incompatible edges. So the number of patterns to study is actually bounded by $4 \times 2^9 = 2^{11}$. In fact, by symmetry arguments it is not difficult to show that the number of distinct open patterns on three variables is actually less than 1000.

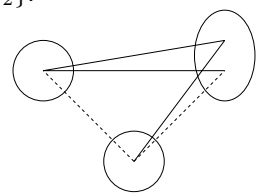
4 Complexity of Patterns with Three or Less Compatible Edges

We now give the complete complexity map of all patterns with three or less compatible edges. In order to do so, we need to give the set $\mathcal{L}$ which is, apart from only two open patterns, the set of all 3-final tractable patterns. We also give the only two open patterns O_1 and O_2 with three or less compatibles edges.

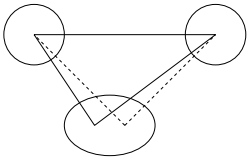
Let $\mathcal{L} = \{L_1, \ldots, L_{12}\}$



and $\mathcal{O} = \{O_1, O_2\}$.



O_1



O_2

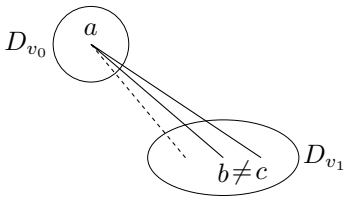
Lemma 3. *Let P be a pattern with three or less compatible edges such that $P \neq O_1, O_2$. Then the two following properties are equivalent:*

1. *P can be reduced to a pattern belonging to $\mathcal{L}$.*
2. *P is tractable.*

Proof. The proof consists of two parts. The first one shows that any pattern $P \neq O_1, O_2$ with three or less compatible edges not reducible to a pattern in $\mathcal{L}$ is NP-Complete. The second one shows that all the patterns in $\mathcal{L}$ are tractable. For space reasons, we cannot include the entire proof. We however give one proof of tractability for one of the patterns in $\mathcal{L}$.

We are going to show that the pattern L_9 is tractable. We suppose we have an instance in which we forbid the pattern L_9.

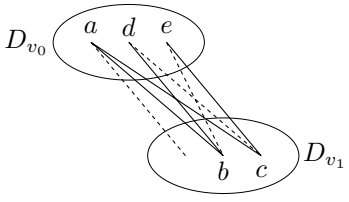
Let G be the following gadget: two variables v_0 and v_1 such that we have a in D_{v_0}, b and c in D_{v_1}, with $b \neq c$, a compatible with both b and c, and a incompatible with a point in D_{v_1}.



Let f be a point in D_{v_2}, with $v_2 \neq v_0, v_1$. If f is compatible with b but not with c, then we have the forbidden pattern L_9. Likewise, if f is compatible with c but not with b, then we have the forbidden pattern L_9. So all the points of the instance not in D_{v_0} or D_{v_1} have the same compatibility towards b and c.

If all points in D_{v_0} have the same compatibility towards b and c, then b and c have the same compatibility towards all the points in the instance. So by neighborhood substitution [4, 5] we can remove c. Removing points cannot introduce a forbidden pattern. If all the points in D_{v_0} which have not the same compatibility towards b and c are compatible with b but not with c, then all the points in the instance compatible with c are also compatible with b. So by neighborhood substitution [4, 5] we can remove c. Similarly, if all the points in D_{v_0} which have not the same compatibility towards b and c are compatible with c but not with b, then all the points in the instance compatible with b are also compatible with c and by neighborhood substitution [4, 5] we can remove b.

Thus we can assume there are d and e in D_{v_0} such that d is compatible with b but not with c, and e is compatible with c and not with b.



Edges (b, e), (b, a) and (b, d) form the gadget G. So a and d have the same compatibility towards all the points in the instance outside of D_{v_0} and D_{v_1}. Similarly, edges (c, d), (c, a) and (c, e) form the gadget G. So a and e have the same compatibility towards all the points in the

instance outside of D_{v_0} and D_{v_1}. So d and e have the same compatibility towards all the points in the instance outside of D_{v_0} and D_{v_1}.

Let S be a solution containing c. Let c' be the point of S in v_0. If c' is compatible with b, then we can replace c by b in S while maintaining the correctness of the solution, since b and c have the same compatibility towards all the points in the instance outside of D_{v_0} and D_{v_1}. If c' is not compatible with b, then we can replace c by b and c' by d in S while maintaining the correctness of the solution, since b and c have the same compatibility towards all the points in the instance outside of D_{v_0} and D_{v_1} and c' and d have the same compatibility towards all the points in the instance outside of D_{v_0} and D_{v_1} (by the same argument as above with e replaced by c'). So if a solution contains c, then there is another solution containing b. So we can remove c.

So each time the gadget G is present, we can remove a point. The gadget G is a known tractable pattern since forbidding G is equivalent to saying that all constraints are zero-one-all [6]. So if it is not present, then the instance is tractable.

Hence we have shown that the pattern L_9 is tractable. □

We can remark at this point that all patterns with two or less compatible edges and whose subpattern of incompatible edges is a subpattern of either OPENV or CLOSEDV can be embedded in one of the patterns in $\mathcal{L}$. Hence all such patterns are tractable.

Conclusion

We have introduced a reduction operation on patterns, which implies a complexity hierarchy between patterns. We then used this tool to show that all tractable 3-variable patterns reduce to just one of four patterns. This leaves only subpatterns of the aforementioned four patterns as open cases. This effectively reduces the number of open patterns from 3^{27} to 2^{11}. We also characterised the tractability of all patterns with three or less compatible edges, except for two patterns whose tractability remains open. In particular, we have shown that all patterns with two or less compatible edges, providing they satisfy some simple properties on their incompatible edges, are tractable. We now have a set $\mathcal{U}$ of only four patterns to which all tractable 3-variable patterns are reducible, and a set $\mathcal{L}$ of patterns which summarises the tractable 3-variable patterns with at most three compatible edges (with two exceptions which are still open). Our future work will focus on narrowing the gap between these upper and lower boundaries, by showing the NP-Completeness of a subpattern of some pattern in $\mathcal{U}$, or by proving the tractability of an extension of some pattern in L.

References

[1] Martin C. Cooper, Peter G. Jeavons, András Z. Salamon, *Generalizing constraint satisfaction on trees: Hybrid tractability and variable elimination*, Artificial Intelligence 174 (9–10) (2010) 570–584.

[2] Martin C. Cooper, Standa Živný, *Hybrid tractability of valued constraint problems*, Artificial Intelligence 175 (9-10) (2011) 1555–1569.

[3] David A. Cohen, Martin C. Cooper, Páidí Creed, András Z. Salamon, *The tractability of CSP classes defined by forbidden patterns*, arXiv:1103.1542.

[4] Eugene C. Freuder, *Eliminating interchangeable values in constraint satisfaction problems*, in: Proceedings AAAI-91, Anaheim, CA (1991) 227-233.

[5] Martin C. Cooper, *Fundamental properties of neighbourhood substitution in constraint satisfaction problems*, Artificial Intelligence 90 (1997) l-24.

[6] Martin C. Cooper, David A. Cohen and Peter G. Jeavons, *Characterising tractable constraints*, Artificial Intelligence 65 (2), 1994, 347-361.

Bucket and Mini-bucket Schemes for M Best Solutions over Graphical Models

Natalia Flerova[1] (student), Emma Rollon[2] and Rina Dechter[1] (supervisor)

[1]University of California Irvine, [2]Universitat Politecnica de Catalunya

Abstract. The paper focuses on finding the m best solutions of a combinatorial optimization problem defined over a graphical model (e.g., Weighted CSP). We describe *elim-m-opt*, a new bucket elimination algorithm for solving the m-best task and analyze its worst-case performance. An extension to the mini-bucket framework that yields a collection of bounds for each of the m-best solutions is discussed and empirically evaluated.

1 Introduction

Given an optimization problem, the objective typically is to find an optimal solution, i.e., a solution that provides the best value of the objective function. However, in many applications it is desirable to obtain not just a single optimal solution but a set (of a given size m) of the best possible solutions.

In the paper we focus on graphical models and show how the well-known Bucket Elimination framework can be extended to compute the m-best solutions by a relatively simple modification of its underlying combination and marginalization operators [1] yielding algorithm *elim-m-opt*.

We analyze the complexity of *elim-m-opt* and discuss extensions to Mini-Bucket Elimination to compute bounds on each of the m-best solutions, yielding algorithm *mbe-m-opt*. We also provide empirical analysis for *mbe-m-opt* demonstrating its effectiveness both as an exact scheme as well as for approximation. Most proofs are omitted for lack of space.

2 Background

We consider problems expressed as graphical models, e.g., Markov and Bayesian networks [3], constraint networks and influence diagrams.

Definition 1 (graphical model).
A ***graphical model*** *is a tuple* $\mathcal{M} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes)$*, where:* $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ *is a set of variables;* $\boldsymbol{D} = \{D_1, \ldots, D_n\}$ *is the set of their finite domains of values;* $\boldsymbol{F} = \{f_1, \ldots, f_r\}$ *is a set of discrete functions, defined on subsets of variables of* $\boldsymbol{X}$*, called scopes and their range is a set* $\boldsymbol{A}$ *whose elements are called* valuations *and* $\bigotimes$ *is the combination operator over functions (typically sum, product, or join). The graphical model* $\mathcal{M}$ *represents the function* $C(\boldsymbol{X}) = \bigotimes_{f \in \boldsymbol{F}} f$*.*

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 25–30, Perugia, Italy, 12 September 2011.

Definition 2 (reasoning task, m-best task).
*A **reasoning task** is a tuple $\mathcal{P} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$, where $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes)$ is a graphical model and $\Downarrow$ is a marginalization operator (i.e. min, max, sum). The reasoning task is to compute $\Downarrow_{\boldsymbol{X}} C(\boldsymbol{X})$. **The m-best task** over a graphical model $\mathcal{M}$ is to find m complete assignments $\boldsymbol{T} = \{t_1, \ldots, t_m\}$, such that $\forall t' \notin \boldsymbol{T}\ \forall t \in \boldsymbol{T}, C(t') \leq C(t)$. The solution is the set of valuations $\{C(t_1), \ldots, C(t_m)\}$, called m-best solutions.*

It was shown in several works that, when the operators of a graphical model satisfy certain axioms, inference algorithms, such as variable elimination and join tree schemes, are sound and complete for the reasoning task. These axioms can be stated directly over the operators $\bigotimes$ and $\Downarrow$ over functions, as was done by Shenoy and Shafer [4]. In this paper we assume graphical models and reasoning tasks obeying the Shenoy-Shafer axioms.

Definition 3 (bucket elimination). *Bucket elimination (*BE*) is a dynamic programming framework used for many reasoning tasks [1]. The input of BE is a reasoning task $\mathcal{P}$ and an elimination ordering $o = (X_1, X_2, \ldots, X_n)$. Each $f_i \in \boldsymbol{F}$ is placed in the bucket of its latest variable in o.* BE *processes the buckets from X_n to X_1, computing for each $Bucket_{X_i}$, noted $\boldsymbol{B}_i$, $\Downarrow_{X_i} \bigotimes_{j=1}^{n} \lambda_j$, where λ_j are the function in the $\boldsymbol{B}_i$, some of which are original $f_i's$ and some are earlier computed messages. The result of the computation, a new function also called* message*, is placed in the bucket of its latest variable in o. The time and space complexity of the algorithm is exponential in a structural parameter called* induced width*, which is the largest scope of all the functions computed.*

3 Algorithm *elim-m-opt*

Let $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes)$ be a graphical model, over which we want to solve the m-best task. Let $2^{\mathbf{A}}$ be the set of subsets of $\mathbf{A}$, and let us call a function with range in $2^{\mathbf{A}}$ a vector function.

Definition 4 (combination and addition over sets). *Let $S, T \in 2^{\boldsymbol{A}}$. Their combination, noted $S \otimes T$, is the set $\{a \otimes b \mid a \in S, b \in T\}$. Their addition, noted $sort^m\{S, T\}$, is the set of the m-best elements in the set $S \cup T$.*

Definition 5 (combination and marginalization over vector functions).

*Let $f : \boldsymbol{D}_f \to 2^{\boldsymbol{A}}$ and $g : \boldsymbol{D}_g \to 2^{\boldsymbol{A}}$ be two vector functions. Their **combination**, noted $f \overline{\bigotimes} g$, is a new vector function defined on scope $var(f) \cup var(g)$ s.t. $\forall t \in D_{var(f) \cup var(g)}$, $f \overline{\bigotimes} g(t) = f(t_{[var(f)]}) \otimes g(t_{[var(g)]})$ The **marginalization** of f over $X_i \in var(f)$, noted $\underset{X_i}{sort^m} f$, is a function over scope $var(f) - \{X_i\}$ such that $\forall t \in D_{var(f)-X_i}$, $(\underset{X_i}{sort^m} f)(t) = sort^m\{\bigcup_{x \in D_{X_i}} f(t \cdot x)\}$*

The bucket-elimination algorithm *elim-m-opt* is described in Algorithm 1 using the two new combination and marginalization operators of $\overline{\bigotimes}$ and $sort^m$.

The algorithm processes the buckets from last to first as usual. The message-function associates each tuple in its domain with the m-best *costs-to-go* restricted to the subproblem below the bucket variable in the bucket tree. It is easy to show that the combination and marginalization operators defined obey Shenoy-Shafer's axioms, from which the correctness of the algorithm follows.

Complexity of *elim-m-opt*: Given n buckets, one for each variable X_i, $\mathbf{B}_i$ containing deg_i (i.e., the degree of the respective node in the bucket-tree) functions and at most w^* different variables, the total time complexity of *elim-m-opt* is $O(nmk^{w^*} \log m)$ and the total space complexity is $O(mnk^{w^*})$.

Algorithm 1 elim-m-opt algorithm

Input: An m-best reasoning task $\tilde{\mathcal{P}}(m) = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, sort^m)$; An ordering of variables $o = \{X_1, \ldots, X_n\}$;
Output: A zero-arity function $\lambda_1 : \emptyset \rightarrow 2^{\mathbf{A}}$ containing the solution.
1: **Initialize:** Generate an ordered partition of vector functions h in buckets $\mathbf{B}_1, \ldots, \mathbf{B}_n$, where $\mathbf{B}_i$ contains all the functions whose highest variable in their scope is X_i.
2: **for** $i \leftarrow n$ down to 1 (Processing bucket B_i) **do**
3: Generate $\lambda_i = sort^m_{X_i}(\overline{\bigotimes}_{f \in B_i} f)$ and assignment $\overline{x_i} = argsort^m_{X_i}(\overline{\bigotimes}_{f \in B_i})$, concatenate with relevant elements of the previously generated assignment messages.
4: Place λ_i and $\overline{x_i}$ in the bucket of the largest-index variable in $var(\lambda_i)$
5: **end for**
6: **Return:** λ_1

3.1 The Mini-Bucket for the m-best

Mini-bucket Elimination *(MBE)* [2] is an approximation designed to avoid the space and time complexity of *BE*. Consider a bucket $\mathbf{B}_i$ and an integer bounding parameter z. *MBE* creates a z-partition $Q = \{Q_1, ..., Q_p\}$ of $\mathbf{B}_i$, where each set $Q_j \in Q$, called *mini-bucket*, includes no more than z variables. Then, each mini-bucket is processed separately, thus computing a set of messages $\{\lambda_{ij}\}_{j=1}^{p}$, where $\lambda_{ij} = \Downarrow_{X_i} (\bigotimes_{f \in Q_j} f)$. In general, greater values of z increase the quality of the bound.

Definition 6 (m-best bound). *Let $S = \{a_1, \ldots, a_j\}$ and $T = \{b_1, \ldots, b_k\}$ be two sets (i.e., $S, T \in 2^{\mathbf{A}}$). S is a m-best bound of T iff $\forall 1 \leq i \leq |T|,\ b_i \leq a_i$.*

Algorithm *m-best MBE* (*mbe-m-opt*) (given in Figure 2) is a straightforward extension of MBE to m-best reasoning task, where the combination and marginalization operators are the ones defined over vector functions. Algorithm *mbe-m-opt* solves m-best reasoning task $\tilde{\mathcal{P}}(m)$ and its output is a *m-best bound* on the m-best solutions.

Theorem 1 (*mbe-m-opt* bound and complexity) *Given an m-best reasoning task $\tilde{\mathcal{P}}(m)$,* mbe-m-opt *computes an m-best bound on $\tilde{\mathcal{P}}(m)$. Given an integer control parameter z, the time and space complexity of* mbe-m-opt *is $O(mnk^z \log(m))$ and $O(mnk^z)$, respectively, where k is the maximum domain size and n is the number of variables.*

Algorithm 2 MBE-m-opt algorithm

Input: An m-best reasoning task $\tilde{\mathcal{P}}(m) = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, sort^m)$; An ordering of variables $o = \{X_1, \ldots, X_n\}$; parameter z.
Output: bounds on each of the m-best solution costs and the corresponding assignments for the expanded set of variables (i.e., node duplication).
1: **Initialize:** Generate an ordered partition of functions $\overline{f}(t) = \{f(t)\}$ into buckets $\mathbf{B}_1, \ldots, \mathbf{B}_n$, where where $\mathbf{B}_i$ contains all the functions whose highest variable in their scope is X_i.
2: **for** $i \leftarrow n$ down to 1 (Processing bucket B_i) **do**
3: Partition functions in bucket B_i into $\{Q_{i_1}, ..., Q_{i_l}\}$, where each Q_{i_j} has no more than z variables.
4: Generate cost messages $\lambda_{i_j} = sort^m_{\mathbf{X}_i}(\overline{\bigotimes}_{f \in Q_{i_j}} f)$ and place each in the largest index variable in $var(Q_{i_j})$
5: **end for**
6: **Return:** The set of all buckets, and the vector of m-best costs bounds in the first bucket.

3.2 Using the m-best bound to tighten the first-best bound

It is easy to observe that upper or lower bounds are generated by solving a relaxed version of a problem, the relaxed problem's solution set contains all the solutions to the original problem.

Proposition 1. *Given the m-best solutions costs generated by* mbe-m-opt *(for clarity we consider MPE problem, namely* $\bigotimes = \prod$, $\Downarrow = \max$ *and* $\boldsymbol{A} \in \{0, 1\}$*. The results can be extended for other reasoning tasks)* $\tilde{C} = \{\tilde{p}_1 \geq \tilde{p}_2 \geq, ..., \geq \tilde{p}_m\}$*, let* p^{opt} *be the probability of the most probable explanation and let* j_0 *be the first index such that* $\tilde{p}_j = p^{opt}$*, or else we assign* $j_0 = m + 1$*. Then, if* $j_0 > m$*,* $\tilde{p}_m$ *is an upper bound on* p^{opt}*, which is as tight or tighter than all other* $\tilde{p}_1, ...\tilde{p}_{m-1}$*. In particular* $\tilde{p}_m$ *is tighter than the bound* $\tilde{p}_1$*.*

If $j \leq m$, we already have optimal value, otherwise we can use $\tilde{p}_m$ as our tighter upper bound, useful during search algorithms such as A*. It is possible to decide efficiently (in $O(nm)$ steps) if a bound coincides with the exact optimal cost. The *mbe-m-opt* provides for each bound a corresponding tuple, where assignments to duplicated variables are maintained. The first assignment from these m-best bounds (going from largest to smallest) that corresponds to a tuple whose duplicate variables are assigned identical value, is optimal.

4 Empirical demonstrations

All experiments assume solving m-best MPE task. We evaluated empirically algorithm *mbe-m-opt* with $m = \{1, 5, 10, 20, 50, 100, 200\}$ and with z-bound 10 on two sets of instances from the UAI 2008 evaluation[1]. The first set contained grid instances with 100-2500 variables and tree-width 12-50, the second - pedigree instances with several hundred variables and tree width 15-30. For clarity and space reasons we present only a subset of instances illustrating typical behaviour.

Figures 1 and 3 present the dependence of the run-time on m for a few selected instances. Figure 2 shows as a function of j the change in accuracy

[1] http://graphmod.ics.uci.edu/uai08/Evaluation/Report/Benchmarks

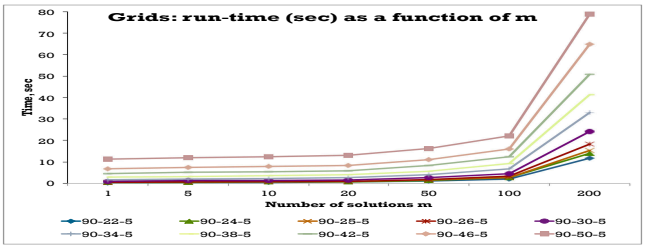


Fig. 1: Run time (sec) as a function of required number of solutions m for the grid instances

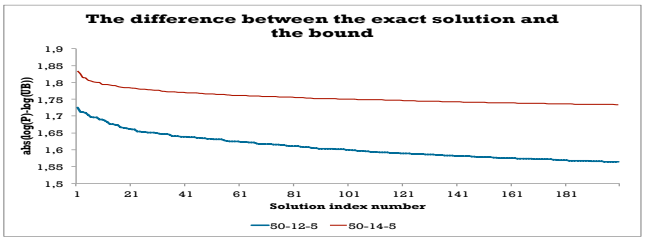


Fig. 2: The absolute difference between the first exact solution and the m^{th} upper bound as a function of m for two grid instances

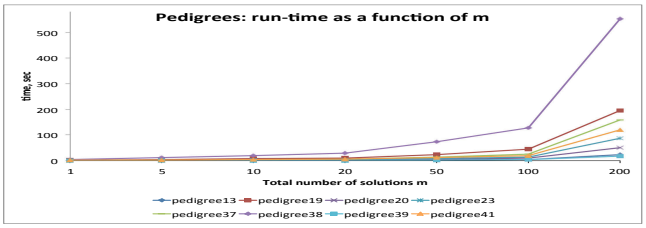


Fig. 3: Run time (sec) as a function of required number of solutions m for the pedigree instances

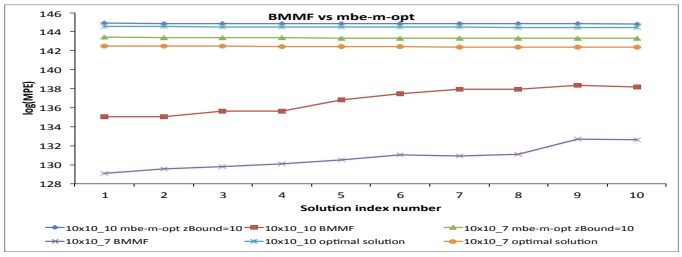


Fig. 4: Comparison of *mbe-m-opt* with z-bound 10 and BMMF to the exact solution for the two instances of random 10 by 10 grids

defined as the absolute difference between the optimal solution and the bound on the j^{th} solution. For these grid instances as j increases, the bound on the cost of the j^{th} solution slowly approaches the exact best solution demonstrating a potential improving of the bound on the optimal assignment using the m-best bounds as discussed in Section 3.2.

We carried some comparison with BMMF, an approximate message-passing algorithm by [5], on randomly generated 10 by 10 grids. The run times of the algorithms are not comparable since our algorithm is implemented in C and BMMF in Matlab, which is inherently slower. The algorithms also differ in the nature of the outputs: BMMF provides approximate solutions with no guarantees while *mbe-m-opt* generates bounds on all the m-best solutions. Still some information can be learned. In Figure 4 we see that the algorithm with the z-bound=10 yields an upper bound and that BMMF outputs significantly less accurate results than *mbe-m-opt* with even a low z-bound. Admittedly, these experiments are quite preliminary and not conclusive.

5 Conclusions

We presented a new bucket-elimination algorithm for solving the m-best task over a graphical model and analyzed its performance. The significance of the proposed algorithm is primarily in providing a unifying inference framework for the m-best task that can both suggest approximation schemes and yield heuristic advice. The promise of the elim-m-opt inference algorithm is in its potential to yield viable bounds for the m-best solutions via the mini-bucket algorithm.

Furthermore, it could also lead to loopy propagation message-passing schemes. Moreover, all such approximation extensions would be applicable to the broad range of graphical models captured by the unifying framework of Shenoy and Shafer. Future work will focus on such extensions and on empirical evaluations of the emerging schemes.

The empirical analysis we provided is only preliminary. Yet it shows that *mbe-m-opt* scales even better than worst-case predict as a function of m. Comparison with other exact and approximation algorithms is left for future work.

References

1. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85, 1999.
2. R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.
3. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.
4. P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. *Uncertainty in Artificial Intelligence*, 4:169–198, 1990.
5. C. Yanover and Y. Weiss. Finding the M Most Probable Configurations Using Loopy Belief Propagation. In *Advances in Neural Information Processing Systems 16*. The MIT Press, 2004.

Consistency of Constraint Networks Induced by Automaton-Based Constraint Specifications

María Andreína Francisco Rodríguez, Pierre Flener, and Justin Pearson

Uppsala University, Department of Information Technology, Uppsala, Sweden
MariaAndreina.Francisco_Rodriguez.3450@student.uu.se,
Pierre.Flener@it.uu.se, Justin.Pearson@it.uu.se

Abstract. We discuss the consistency of constraints for which the set of solutions can be recognised by an automaton. Such an automaton induces a decomposition of the constraint into a conjunction of constraints. The so far most general result is that if the constraint hypergraph of such a decomposition is Berge-acyclic, then the decomposition provides hyper-arc consistency. We focus on constraint networks that have α-acyclic or centred-cyclic hypergraph representations and show the necessary conditions to achieve hyper-arc consistency in these cases.

Keywords: Acyclicity, hypergraph, automaton.

1 Introduction

Global constraints are an important component in many modern constraint solvers. A global constraint does two things: from the modelling perspective, it allows a modeller to express commonly occurring combinatorial structures; from the solving perspective, it comes with a filtering algorithm that removes impossible domain values during search.

In [3], a framework is given where a global constraint can be specified in a relatively simple and high-level way by a (deterministic or non-deterministic) finite automaton. The idea is to describe *what* it means for the constraint to be satisfied in terms of the accepting paths of the automaton. Based on the automaton, the framework decomposes the specified new global constraint into a conjunction of already implemented (global) constraints. These constraints give the semantics of the specified global constraint and provide the filtering.

It is so far known [2, 3] that if the constraint graph of a decomposition (induced by an automaton) is Berge-acyclic [5], then the decomposition automatically provides hyper-arc consistency, that is the decomposition achieves all the filtering that is possible. Beside Berge-acyclity, another ten patterns of constraint hypergraph structure are identified in the current on-line version of the *Global Constraint Catalogue* [4],[1] but little is known about the filtering strength of the (automaton-induced) decompositions that satisfy these structures.

[1] See http://www.emn.fr/z-info/sdemasse/gccat/sec3.6.5.html

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 31–36, Perugia, Italy, 12 September 2011.

In this paper, we show how an α-acyclic constraint hypergraph (see Section 3) can be modified to provide hyper-arc consistency. Moreover, we show that by adding implied constraints the so-called centred-cyclic networks of [4] can also be modified to provide hyper-arc consistency (see Section 4). This covers five of the ten open hypergraph patterns (namely α-acyclic(2), α-acyclic(3), centred-cyclic(1), centred-cyclic(2), and centred-cyclic(3)) and almost doubles the number of automaton-induced decompositions in the on-line version of the *Global Constraint Catalogue* [4] that are known to provide hyper-arc consistency.

It was observed in [2] that an α-acyclic constraint hypergraph can be made hyper-arc consistent by making the constraints pairwise consistent, but no algorithm was given. Here we show the connection (Theorem 1) between maintaining pairwise consistency and doing a reachability analysis on an automaton.

There is also a large body of related work (e.g., [6–8, 13]) on decomposing global constraints to achieve hyper-arc consistency. This work can be seen as a more systematic approach to provide hyper-arc consistency via decompositions.

2 Background: The *automaton* Constraint

The $automaton(A, V)$ constraint [3] holds if the constraint described by the automaton A holds for the sequence of decision variables V, that is if A accepts the sequence of values of V. We define the *automaton* constraint in three stages: first its particular case that is also known as the *regular* constraint [12], and then two orthogonal extensions, namely predicate automata and counter automata.

2.1 Modelling Constraints with Automata

The *automaton* constraint can be implemented either via a specialised propagator [12], or via decomposition into a conjunction of constraints [3]. For a given automaton, define a constraint $T(\rho, \rho', \sigma)$ extensionally by the following set:

$$\{\langle \rho, \rho', \sigma \rangle \mid \rho \xrightarrow{\sigma} \rho'\} \tag{1}$$

That is, $T(\rho, \rho', \sigma)$ is satisfied whenever there is a transition from state ρ to state ρ' that consumes symbol σ. An *automaton* constraint on a sequence of n decision variables, $v_1, \ldots, v_n$, is then decomposed into the following conjunction of $n + 2$ constraints, called the *transition constraints*:

$$q_0 \in S \wedge T(q_0, q_1, v_1) \wedge \cdots \wedge T(q_{n-1}, q_n, v_n) \wedge q_n \in F \tag{2}$$

where $q_0, q_1, \ldots, q_{n-1}, q_n$ are new decision variables, called the *state variables*, with domain Q. For contrast, we call $v_1, \ldots, v_n$ the *problem variables*.

The implementation of [3] actually works unchanged for non-deterministic finite-state automata, but we have elected to restrict our focus to deterministic ones, in order to ease the notation.

2.2 Modelling Constraints with Predicate Automata

The automata in [3] are more powerful than those in [12]: the labels can be predicates, and all predicates must be satisfied on an accepting path. Let $\mathbf{Pred}_k$ be a set of k-ary predicates in some suitable language. That is, a predicate takes a vector, $\mathcal{V}$, of k values and it is either true or false.

A *k-ary-predicate DFA* is a tuple $\langle Q, \Sigma, \phi, S, F, \delta \rangle$, where Q, Σ, S, F, and δ are exactly as for a DFA, and ϕ is a function from Σ to $\mathbf{Pred}_k$. Given a predicate automaton $\langle Q, \Sigma, \phi, S, F, \delta \rangle$, the automaton $\langle Q, \Sigma, S, F, \delta \rangle$ is referred to as the *underlying automaton* of the predicate automaton.

In [3], constraints defined by predicate automata are implemented with the help of reification. The constraint T defined in (1) is used for the following transition constraints:

$$q_0 \in S \wedge T(q_0, q_1, s_1) \wedge \cdots \wedge T(q_{n-1}, q_n, s_n) \wedge q_n \in F \tag{3}$$

These transition constraints are like (2), but are expressed for *new* decision variables $s_1, \ldots, s_n$, which are connected as follows to the problem variables via the predicates and reification: given an n-length sequence $\mathcal{V}_1, \ldots, \mathcal{V}_n$ of k-ary vectors of problem variables, we add the following constraints, called the *signature constraints*:

$$\bigwedge_{\sigma \in \Sigma} (s_i = \sigma \Leftrightarrow \phi(\sigma)(\mathcal{V}_i)) \tag{4}$$

for all $1 \leq i \leq n$, where the s_i are called the *signature variables*, with domain Σ. Hence $\mathbf{Pred}_k$ contains whatever can be implemented as reified constraints in the underlying constraint solver.

2.3 Modelling Constraints with Counter Automata

While the class of constraints that can be described by (predicate) automata is very large (currently, 63 of the 354 constraints of the on-line version of the *Global Constraint Catalogue* [4] are described that way), it is often the case that (predicate) automata are very large or specific to a problem instance. The second extension in [3] is the use of counters that are initialised at the start and evolve through counter-updating operations coupled to the transitions of the automaton. Such counter automata allow the capture of non-regular languages and yield (even for regular languages) automata that are much smaller if not instance-independent (and currently enable another 57 constraints of the catalogue to be described succinctly or generically). The two extensions are orthogonal and can be composed, so we define this second extension in isolation.

Unlike the counter automata in theoretical computer science (see, e.g., [11]), the counter automata here do not have access to the values of the counters during a run, but the values of the counters are updated by the transitions.

In [3], counter automata are decomposed into transition constraints that are slightly extended to include information about the values of the counters. Define a new constraint $T(\rho, \rho', \mathcal{C}, \mathcal{C}', \sigma)$ extensionally by the following set:

$$\{\langle \rho, \rho', \mathcal{C}, \mathcal{C}', \sigma \rangle \mid (\rho, \mathcal{C}) \xrightarrow{\sigma} (\rho', \mathcal{C}')\}$$

An *automaton* constraint on a sequence of n problem variables, $v_1, \ldots, v_n$, and a vector of ℓ counters, $\mathcal{C}$, is then decomposed into the following conjunction of $n+3$ transition constraints:

$$q_0 \in S \wedge T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1) \wedge \cdots \wedge T(q_{n-1}, q_n, \mathcal{C}_{n-1}, \mathcal{C}_n, v_n) \wedge q_n \in F \wedge \mathcal{C} = \mathcal{C}_n \quad (5)$$

where $q_0, q_1, \ldots, q_{n-1}, q_n$ are state variables, with domain Q, while $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{n-1}, \mathcal{C}_n$ are vectors of integer decision variables, called *counter variables*, and $\mathcal{C}_0$ has the initial values of the counters.

3 α-Acyclic Automata

It is well known that a sufficient method for achieving hyper-arc consistency for a constraint set with a Berge-acyclic constraint hypergraph is to ensure each constraint of the hypergraph is hyper-arc consistent [9,10]. Nevertheless, in general, hyper-arc consistency of the individual constraints of a problem is not sufficient to ensure hyper-arc consistency of the whole problem.

Without loss of generality, in the rest of the paper, we only consider constraint satisfaction problems whose constraint hypergraph is connected and reduced.

Let $\{R_1(S_1), \ldots, R_m(S_m)\}$ be a set of constraints with an α-acyclic constraint hypergraph that are pairwise consistent, and let $R(S)$ be the constraint representing the set of solutions of the conjunction of these constraints, where S is a tuple containing all the variables in $S_1, \ldots, S_m$. It can be shown [1] that for all i the projection $\text{proj}_{S_i} R(S)$ is equal to $R_i(S_i)$. Thus, all tuples in the constraints $R_i(S_i)$ participate in a solution. In particular, if all the constraints in the set are hyper-arc consistent, their conjunction is hyper-arc consistent.

Theorem 1. *Consider the transition constraints (5) from a counter-automaton-induced constraint decomposition:*

$$q_0 \in S \wedge T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1) \wedge \cdots \wedge T(q_{n-1}, q_n, \mathcal{C}_{n-1}, \mathcal{C}_n, v_n) \wedge q_n \in F \wedge \mathcal{C} = \mathcal{C}_n$$

Maintaining these transition constraints pairwise consistent is equivalent to doing reachability analysis on the corresponding automaton.

Proof. The transition constraint $T(q_{i-1}, q_i, \mathcal{C}_{i-1}, \mathcal{C}_i, v_i)$ codes the transition relation of the automaton. Achieving pairwise consistency on the pair $q_0 \in S$ and $T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1)$ forces the domain of q_1 and $\mathcal{C}_1$ to include only states and counter values that are reachable after consuming one alphabet symbol of the automaton. Thus $T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1)$ can be replaced by a constraint T_0, which is a sub-constraint of T that only contains the tuples corresponding to transitions that can happen in one step. Assume that hyper-arc consistency and pairwise consistency have been achieved up to step i by the constraints:

$$q_0 \in S \wedge T(q_0, q_1, \mathcal{C}_0, \mathcal{C}_1, v_1) \wedge \cdots \wedge T(q_{i-1}, q_i, \mathcal{C}_{i-1}, \mathcal{C}_i, v_i)$$

By induction, the domains of q_i and $\mathcal{C}_i$ are all states and counter values that are reachable after i transitions in the automaton. Thus, achieving pairwise

consistency on the pair $T_i(q_{i-1}, q_i, \mathcal{C}_{i-1}, \mathcal{C}_i, v_i)$ and $T(q_i, q_{i+1}, \mathcal{C}_i, \mathcal{C}_{i+1}, v_{i+1})$ gives a new constraint T_{i+1} that contains only the subset of the tuples of T_i that can take part in a chain of transitions of length $i + 1$.

4 Centred-Cyclic Automata

We now analyse another type of constraint hypergraph that appears in the online version of the *Global Constraint Catalogue* [4], namely centred-cyclic hypergraphs. We show that it is possible to ensure hyper-arc consistency for this kind of constraint hypergraph. Since a centred-cyclic hypergraph is not α-acyclic, in order to achieve hyper-arc consistency, new methods have to be used.

A centred-cyclic constraint decomposition has two groups of constraints: the transition constraints and the signature constraints. Considered alone, the transition constraints are berge-acylic, and the signature constraints are α-acyclic. So, it is possible to achieve hyper-arc consistency on each group separately, but this is not enough to achieve hyper-arc consistency of the whole decomposition. The transition constraints and the signature constraints only overlap in the sequence of signature variables $s_1, \ldots, s_n$. In order to ensure hyper-arc consistency for the whole hypergraph, the projection of both subproblems onto their common variables must be the same: that is, the set of possible values for the sequence of signature variables $s_1, \ldots, s_n$ must be a solution to both the signature constraints and the transition constraints. Thus it is sufficient to add to the decomposition an implied constraint $\mathcal{I}(\langle s_1, \ldots, s_n \rangle)$ that is satisfied by a sequence of signature values $\langle \sigma_1, \ldots, \sigma_n \rangle$ if and only if the corresponding transition sequence is allowed by the automaton and the signature constraints.

Theorem 2. *Given an automaton-induced decomposition, the constraint* $\mathcal{I}(\langle s_1, \ldots, s_n \rangle)$ *can be computed directly from the underlying automaton.*

Proof. Assuming that each predicate of the automaton is satisfiable for some assignment of the problem variables, the signature constraints (4) allow all possible values for each signature variable s_i. The only restrictions on the values of the signature variables come from the transition constraints (3).

The transition constraints, together with the condition that q_0 and q_n respectively belong to the start and accepting state sets, restrict the intermediate values to correspond to a chain of transitions from a start state to an accepting state. Thus the constraint $\mathcal{I}$ must restrict the values of the signature variables to correspond to the edge labels of accepting chains of the underlying automaton.

5 Conclusion

We have used the notions and results of hypergraphs and relational databases to derive properties of constraints given their hypergraph. In particular, we have shown that a constraint satisfaction problem whose hypergraph is α-acyclic can be modified in order to achieve hyper-arc consistency. Also, we have shown a

way to decompose centred-cyclic constraints and evaluate their consistency. We will now investigate the remaining five hypergraph constraint patterns currently identified in the on-line version of the *Global Constraint Catalogue* [4]. Moreover, we will study how these results can be applied in practice.

Acknowledgements. The second and third authors are supported by grant 2007-6445 of the Swedish Research Council (VR).

References

1. Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, July 1983.
2. Nicolas Beldiceanu, Mats Carlsson, Romuald Debruyne, and Thierry Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4):339–362, 2005.
3. Nicolas Beldiceanu, Mats Carlsson, and Thierry Petit. Deriving filtering algorithms from constraint checkers. In Mark Wallace, editor, *Proceedings of CP'04*, volume 3258 of *LNCS*, pages 107–122. Springer-Verlag, 2004.
4. Nicolas Beldiceanu, Mats Carlsson, and Jean-Xavier Rampon. Global constraint catalogue: Past, present, and future. *Constraints*, 12(1):21–62, March 2007. The catalogue is at `http://www.emn.fr/z-info/sdemasse/gccat`.
5. Claude Berge. *Graphes et Hypergraphes.* Dunod, Paris, France, 1970.
6. Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, Claude-Guy Quimper, and Toby Walsh. Reformulating global constraints: The *slide* and *regular* constraints. In *Proceedings of SARA'07*, volume 4612 of *LNAI*, pages 80–92. Springer-Verlag, 2007.
7. Christian Bessière, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Decomposition of the NVALUE constraint. In David Cohen, editor, *Proceedings of CP'10*, volume 6308 of *LNCS*, pages 114–128. Springer-Verlag, 2010.
8. Christian Bessière, George Katsirelos, Nina Narodytska, and Toby Walsh. Circuit complexity and decompositions of global constraints. In Craig Boutilier, editor, *Proceedings of IJCAI'09*, pages 412–418, 2009.
9. Philippe Janssen and Marie-Catherine Vilarem. Problèmes de satisfaction de contraintes : Techniques de résolution et application à la synthèse de peptides. Technical Report 54, Centre de Recherche en Informatique de Montpellier, France, 1988.
10. Philippe Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes : Algorithmes de propagation et de résolution - Propagation de contraintes dans les réseaux dynamiques.* PhD thesis, Université Montpellier II, France, 1991.
11. Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *Proceedings of ATVA'05, the 3rd International Symposium on Automated Technology for Verification and Analysis*, volume 3707 of *LNCS*, pages 489–503. Springer-Verlag, 2005.
12. Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Mark Wallace, editor, *Proceedings of CP'04*, volume 3258 of *LNCS*, pages 482–495. Springer-Verlag, 2004.
13. Claude-Guy Quimper and Toby Walsh. Decomposing global grammar constraints. In Christian Bessière, editor, *Proceedings of CP'07*, volume 4741 of *LNCS*, pages 590–604. Springer-Verlag, 2007.

Towards a better understanding of hardness

Matthew Gwynne and Oliver Kullmann

Computer Science Department
Swansea University, UK
csmg@Swansea.ac.uk, O.Kullmann@Swansea.ac.uk
http://cs.swan.ac.uk/~csmg, http://cs.swan.ac.uk/~csoliver

Abstract. Of fundamental importance for SAT solving is the translation to CNF. One of the basic tasks is to find metrics for determining what are "good" translations, i.e., what makes the resulting SAT problem easy to solve. We introduce a new measure $\mathrm{hd}(F)$, the "hardness", for formulas F in conjunctive normal form (i.e., clause-sets). $\mathrm{hd}(F)$ for unsatisfiable clause-sets has been studied in [6,8]. However, now we treat satisfiable clause-sets differently. We consider the Advanced Encryption Standard (AES) and Data Encryption Standard (DES) as examples. The key discovery for these ciphers present examples of hard problems, and we investigate translations of these problems geared towards minimising $\mathrm{hd}(F)$. We present the *SAT representation hypothesis*: The task of solving a SAT problem efficiently is captured by constructing a representation of the underlying boolean function which is of low hardness.

1 Introduction

Over the past decade, the use of SAT solvers for solving industrial and crafted problems has greatly increased. In particular, the use of SAT to solve cryptographic problems has become popular, and there are several translations of the Data Encryption Standard and Advanced Encryption Standard ciphers into SAT. The question arises with any translation: what makes a good translation for some problem P? We investigate this question using AES and DES as examples.

We assume that P is already decomposed, somehow, into "constraints". In our example, a constraint C is a "box" within the cipher; another typical example for C would be a cardinality constraint. By a "constraint" we mean that we "know" (at least) all satisfying and falsifying assignments. We consider only SAT translations which preserve the decomposition of P and so each C gets translated into a clause-set F_C, and all the F_C together yield the translation of P. Despite translating to SAT, we still want to consider $F = F_C$ as some form of constraint; we want to "know" its satisfying and falsifying assignments and we want to let the SAT solver know! A common approach here is to adapt the notion of "hyper-arc consistency" by the use of UCP (unit-clause propagation), that is, in the weak sense, for every partial assignment φ such that $\varphi * F$ is unsatisfiable, UCP on $\varphi * F$ yields the empty clause, or, in the stronger sense, additionally every forced assignment in $\varphi * F$ (for satisfiable $\varphi * F$) is also derived by UCP. We strengthen/generalise this approach in three ways:

- we generalise UCP, allowing arbitrary polynomial time, in some standardised sense, to get to "know" the satisfying and falsifying assignments;
- we consider *all* variables in F_C, not just those in C — that is, we should know *all* satisfying assignments of F;

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 37–42, Perugia, Italy, 12 September 2011.

– instead of ad-hoc methods to construct F (for example those for cardinality constraints) we develop systematic methods.

The Data Encryption Standard and the Advanced Encryption Standard have both received significant attention from the cryptography community, and have been attacked in [1,5,10] using SAT solvers. In both cases, the ciphers encrypt a plaintext P to a ciphertext C using a key K. We consider the key discovery problems, i.e., finding K from P and C. The DES can be considered as computing the ciphertext via iterated application, using 16 rounds, of various 10-bit S-boxes (6-bit to 4-bit boolean functions, 8 per round) and addition (XOR) of input and key bits. The AES can be considered in a similar way, using 10 rounds and 16-bit S-boxes (8-bit to 8-bit permutations, 16 per round); the AES also has a linear MixColumns operation in each round, using $4 \cdot 16$ 8-bit to 8-bit multiplication operations, and also a non-trivial key-schedule (unlike the DES), involving further S-boxes and additions. It is these S-boxes, multiplications and additions that we consider as our constraints. That is, we translate the AES and DES via a decomposition based on the definition, considering the S-boxes, additions and multiplications as the constraints to translate directly to SAT. This differs from the approaches taken in [2,5,10], which apply global algebraic methods and completely "rewrite" the DES/AES constraint system.

In order to obtain solvable problems, we consider the "small-scale AES" generalisation as introduced in [2], denoted by $\mathrm{aes}(m, r, c, e)$, and parameterised by the number m of rounds, number r of rows, number c of columns, and the bit-size e of individual elements of the AES "block". Key, plaintext and ciphertext have the same number of bits $r \cdot c \cdot e$. The standard 128-bit AES is $\mathrm{aes}(10, 4, 4, 8)$, that is, 10 rounds and the 128-bits are considered as 4×4 matrix of 8-bit elements. All "boxes" (i.e., permutations) have e input bits and e output bits, and thus yield $2e$-bit boolean functions. The MixColumns operation, itself an operation using r^2 many such (multiplication) permutation boxes, has r-vectors of e-bit elements as inputs and outputs, and there are c such operations per round.

In Section 2, we investigate a new notion of hardness of CNF formulas where, in general, a low hardness should mean a good translation. Then in Section 3, we present experimental results on using representations with low hardness. All work presented is available within the `OKlibrary` (see [9]), a research platform for hard problems, and also in [4], a forthcoming technical report.

2 Notions of hardness

In [6,8], a measure of hardness $\mathrm{hd}_{U,S} : \mathcal{CLS} \to \mathbb{N}_0$ of clause-sets[1)] was introduced. For unsatisfiable clause-sets, this notion is closely related to the complexity of tree-resolution. For satisfiable instances it measures the complexity of unsatisfiable sub-instances one has to solve to arrive at a satisfying assignment, using only forced assignments (i.e., the negated assignment yields an unsatisfiable sub-instance) plus guessed assignments. $\mathrm{hd}_{U,S}$ is parameterised by oracles U, S for unsatisfiability/satisfiability detection. For this paper, we restrict our considerations to the simplest cases $U_0 := \{F \in \mathcal{CLS} : \bot \in F\}$ ($\bot$ is the empty clause) and $S_0 := \{\top\}$ ($\top$ is the empty-clause-set). A clause-set is considered trivially unsatisfiable or satisfiable if it is in U_0 or S_0 respectively. In [6,8] a hierarchy of clause-set classes $G_k(U, S)$ is defined. Checking whether $F \in G_k(U, S)$ can be done in time $O(n^{2k})$ using breadth-first search.

[1)] A clause-set is set of set of literals which is then interpreted as either a conjunctive or disjunctive normal form formula. If not stated, we consider CNF clause-sets.

Definition 1. *The* ***hardness*** $\mathbf{hd_{U,S}(F)} \in \mathbb{N}_0$ *of a clause-set* $F \in \mathcal{CLS}$ *is the minimum* $k \in \mathbb{N}_0$ *with* $F \in G_k(U, S)$.

We have:

1. $F \in G_0(U_0, S_0)$ iff $\bot \in F$ or $F = \top$.
2. $F \in G_1(U_0, S_0)$ iff after unit-clause propagation either $\bot \in F$ or $F = \top$, where for the latter we can guess one assignment.
3. $F \in G_2(U_0, S_0)$ iff after failed-literal reduction either $\bot \in F$ or $F = \top$, where for the latter we can guess two assignments.

Only using the aspect of forced assignment, without guessing assignments, we obtain the k-level reductions $r_k : \mathcal{CLS} \to \mathcal{CLS}$. r_1 is unit-clause propagation, r_2 is failed-literal reduction, and so on. For unsatisfiable F we have that $\mathrm{hd}_{U_0,S_0}(F) = \mathrm{hd}_{U_0}(F)$ is the smallest $k \in \mathbb{N}_0$ such that $\bot \in r_k(F)$.

2.1 A new notion of hardness

We introduce a new notion of hardness, based on hd_{U_0}. For satisfiable clause-sets, now instead of measuring the resources for finding some satisfying assignment, we measure the level of r_k-reductions needed to derive *all* possible conclusions.

Definition 2. *The* ***hardness*** $\mathbf{hd(F)} \in \mathbb{N}_0$ *for* $F \in \mathcal{CLS}$ *is the minimal* $k \in \mathbb{N}_0$ *such that for all clauses* C *with* $F \models C$ *(i.e.,* F *implies* C*) we have* $\bot \in r_k(\varphi_C * F)$. *The partial assignment* φ_C *sets all literals of* C *to* 0*, while "*$*$*" denotes application of partial assignments. If* $\mathrm{hd}(F) \leq k$ *we call* F ***k-soft****, and if* $\mathrm{hd}(F) \geq k$ *we call* F ***k-hard****.*

For unsatisfiable clause-sets F, we have $\mathrm{hd}_{U_0}(F) = \mathrm{hd}(F)$. However, for satisfiable clause-sets, the two notions differ. For $F \in \mathcal{SAT}$, $\mathrm{hd}(F)$ characterises the level of look-ahead needed to determine that the solver has entered a unsatisfiable sub-branch, during any depth-first backtracking search. Some polynomial time SAT decision classes have low hardness. For instance:

- If F is (renamable) Horn formula, then $\mathrm{hd}(F) \leq 1$.
- If all clauses of F have length at most 2, then $\mathrm{hd}(F) \leq 2$.

A satisfying assignment for F with $\mathrm{hd}(F) = k$ can be found in polynomial time by checking whether $\bot \in r_k(F)$; if $\bot \in r_k(F)$ then F is unsatisfiable; otherwise we find a variable v and assignment $b \in \{0, 1\}$ such that $\bot \notin r_k(\langle v \to b\rangle * F)$ and continue recursively on $\langle v \to b\rangle * F$ which must itself be satisfiable. Now consider a boolean function f with n variables, such that we wish to *represent* f as a CNF clause-set.

Definition 3. *A* ***representation*** $F \in \mathcal{CLS}$ *of a boolean function* f *is a clause-set* F *with* $\mathrm{var}(f) \subseteq \mathrm{var}(F)$ *such that restricting the satisfying assignments of* F *to* $\mathrm{var}(f)$ *we get exactly the satisfying assignments of* f.

The two most extreme representations (without new variables) of f with minimum and maximum hardness are the set of all prime implicates and the canonical CNF.

Definition 4. *Let* $\mathbf{prc_0(f)}$ *be the set of all clauses* C *which as CNF are* ***prime implicates*** *of* f*, that is,* $f \models \{C\}$ *while* $f \not\models \{C'\}$ *for every* $C' \subset C$. *For clause-sets, we overload this notion, that is, for* $F \in \mathcal{CLS}$*, we have* $\mathbf{prc_0(F)} = \mathrm{prc}_0(f)$ *for the boolean function* f *given by* F.

Let $\mathbf{CNF}(f)$ *be the canonical CNF clause-set for* f*, that is, the CNF clause-set* F *with* $\mathrm{var}(F) = \mathrm{var}(f)$ *and with* $C \in F$ *for all clauses of length* $n(f) := |\mathrm{var}(f)|$ *with* $F \models C$*. For the canonical DNF clause-set, we use* $\mathbf{DNF}(f)$*. In general, we call a clause-set* F ***full*** *if every clause* $C \in F$ *has* $|C| = n = |\mathrm{var}(F)|$*.*

The prime implicates for f have that $\mathrm{hd}(\mathrm{prc}_0(f)) = 0$; for all unsatisfiable full CNFs F, we have that $\mathrm{hd}(F) = n$ and for arbitrary full F we have that $\mathrm{hd}(F) = n(F) - \min_{C \in \mathrm{prc}_0(F)} |C|$.

2.2 Representations with low hardness

With $\mathrm{hd}_{U,S}(F)$, there is a polynomial time algorithm for checking "$\mathrm{hd}_{U,S}(F) = k$?". For $\mathrm{hd}(F)$, the core idea is that we do not measure $\mathrm{hd}(F)$ but construct a clause-set F representing some boolean function f so as to minimise $\mathrm{hd}(F)$. Therefore, we consider 1-soft representations of boolean functions, which we will use to translate small sub-functions within the AES and DES in Section 3.

The use of new variables is a powerful method for reducing the complexity when solving problems from propositional logic. We introduce a translation for a boolean function f using new variables based on the canonical DNF, introducing a new variable $\mathrm{vct}_f(C) \notin \mathrm{var}(f)$ for each clause $C \in \mathrm{DNF}(f)$ (recall, these clauses represent the total satisfying assignments of f).

Definition 5. *Consider a boolean function* $f : \{0,1\}^V \to \{0,1\}$*. Let the* ***canonical translation*** $\mathbf{ct}(f) \in \mathcal{CLS}$ *be defined as*

$$\mathrm{ct}(f) := \left\{ \{\mathrm{vct}_f(C)\}_{C \in \mathrm{DNF}(f)} \right\} \cup \bigcup_{C \in \mathrm{DNF}(f)} \mathrm{prc}_0\left(\mathrm{vct}_f(C) \leftrightarrow \bigwedge_{x \in C} x \right).$$

Lemma 6. *For all* $f : \{0,1\}^V \to \{0,1\}$ *the canonical translation* $\mathrm{ct}(f)$ *is* 1*-soft.*

To construct k-soft representations without new variables, we present a search-based approach. We want then to find representations which are minimal in a sense, and so we introduce the notion of a "base".

Definition 7. *A* k***-base****,* $k \in \mathbb{N}_0$*, for a boolean function* f *is a clause-set* F *fulfilling the following conditions:*

1. F *is a representation of* f*;*
2. F *is* k*-soft;*
3. F *is minimal w.r.t. elimination of clauses and literal occurrences for 1 & 2.*

F *is a* k***-base*** *(for itself) if it is a* k*-base for the underlying boolean function.*

The basic idea to construct a k-base F is to start with the set $\mathrm{prc}_0(f)$ of all prime implicates, and iteratively remove clauses from F while checking that all $C \in \mathrm{prc}_0(f)$ still follow from F by r_k. This simple scheme takes a very long time, and so we have developed heuristics for generating small 1-bases. We remark that computing small*est* 1-bases seems not feasible for the examples we considered.

2.3 SAT representation hypothesis

Based on our new notion of hardness, we present the *SAT representation hypothesis*. This states that the task of finding a "good" representation of a boolean

function f, for the purpose of SAT solving in polynomial time, is fully captured by constructing a k-soft representation F for f for some appropriate k (depending on the problem). The smaller k, the lower the exponent for the polynomial in the run-time estimation, but the larger F is, so a *balance* is to be sought. If f is only some part of a bigger function (for example the S-box in AES), then f should be made as large as possible.

3 Experimental results

The AES and DES ciphers were translated using translations provided in the `OKlibrary`. Additions are translated directly as their prime implicates, while each of the S-boxes, and multiplications were translated using:

- "minimum" CNF representations, minimising the number of clauses;
- the canonical representation (see Section 2.2);
- 1-base translations (see Section 2.2);

In terms of hardness, the canonical and 1-base translations are both 1-soft, while the minimum translations are k-hard for $2 \leq k \leq 4$ (computed). We would expect to see the fact that the canonical and 1-base translations are "easier" reflected in run times of the solvers, and in particular in the number of conflicts (or nodes) the solver uses.

We ran SAT solvers `minisat-2.2.0` ([3]) and the `OKsolver_2002` ([7]) on key discovery instances of the DES over m rounds, des(m), and small scale AES variants with m rounds, c columns, r rows and field size e, aes(m, r, c, e). In general, as each of the parameters increases, the instances become harder. Results for DES and AES are presented in Figure 1 and Figure 2; the full results are available at `http://cs.swan.ac.uk/~csmg/Hardness/`.

Variant	$\|K\|$	`minisat-2.2.0`			`OKsolver_2002`		
		can	1-base	min	can	1-base	min
des(3)	64	3,717.85	**1,146.9**	3,714.85	**72.75**	13,186.7	395,749.8
des(4)	64	1,557,147	**1,259,590**	1,271,120	-	**4,205,769**	4,818,590
des(5)	64	**40,018,619**	183,335,114	258,451,462	-	-	-
aes(10, 2, 1, 8)	16	135,919	**131,517**	159,292,117	-	**17,633**	-
aes(3, 2, 4, 4)	32	26,578,774	**9,731,073**	43,754,923	-	-	-
aes(2, 2, 2, 8)	32	**3,908,438**	19,744,428	181,241,122	-	-	-

Fig. 1. Conflicts/nodes used to solve DES/AES variants

4 Conclusion

Based on the results so far, the hardness of small sub-functions in translations to SAT seem to offer a good indicator of the performance of SAT solvers on the whole problem. Translations using sub-function representations with low hardness were sometimes an order of magnitude better than small or full translations.

In some instances, the results were not as expected, with the minimum translation performing better than a 1-soft translation. However, there are still further

Variant	$\|K\|$	minisat-2.2.0			OKsolver_2002		
		can	1-base	min	can	1-base	min
des(3)	64	0.20	**0.016**	0.027	**4.22**	7.54	129.04
des(4)	64	459.18	48.19	**31.12**	> 10800	2,039.62	**1,140.81**
des(5)	64	17,221	**13,363**	14,291	> 18000	> 18000	> 18000
aes(10, 2, 1, 8)	16	298.18	**229.37**	9,617.52	> 18000	**3,224.61**	> 18000
aes(3, 2, 4, 4)	32	4,067.81	**690.87**	1,771.68	> 18000	> 18000	> 18000
aes(2, 2, 2, 8)	32	13,600.25	**4,679.6**	9,429.73	> 18000	> 18000	> 18000

Fig. 2. Times (in seconds) used to solve DES/AES variants

questions to be answered including, for instance, how the decomposition of the translation into small boolean functions affects hardness. Answering such questions should shed light on the reasons for the performance of solvers in such instances.

The next step is to investigate different decompositions of the DES and AES into small boolean functions, considering also translations given in [1,5,10], and to consider how such decompositions affect the hardness of the final translation. Such investigations must also continue at a theoretical level, providing a framework for constructing "easy" representations of boolean functions.

References

1. Gregory V. Bard and Nicholas T. Courtois. Algebraic cryptanalysis of the Data Encryption Standard. In Steven D. Galbraith, editor, *Proceedings of the 11th IMA international conference on Cryptography and coding*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007.
2. Carlos Cid, Sean Murphy, and Matthew Robshaw. *Algebraic Aspects of the Advanced Encryption Standard*. Springer, 2006. ISBN-10 0-387-24363-1.
3. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing 2003*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518, Berlin, 2004. Springer. ISBN 3-540-20851-8.
4. Matthew Gwynne and Oliver Kullmann. Attacking DES + AES via SAT: Constraints and boolean functions. Technical Report arXiv:??? [cs.DM], arXiv, April 2011.
5. Philipp Jovanovic and Martin Kreuzer. Algebraic attacks using SAT-solvers. *Groups-Complexity-Cryptology*, 2(2):247–259, December 2010.
6. Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical Report TR99-041, Electronic Colloquium on Computational Complexity (ECCC), October 1999.
7. Oliver Kullmann. Investigating the behaviour of a SAT solver on random formulas. Technical Report CSR 23-2002, Swansea University, Computer Science Report Series (available from `http://www-compsci.swan.ac.uk/reports/2002.html`), October 2002. 119 pages.
8. Oliver Kullmann. Upper and lower bounds on the complexity of generalised resolution and generalised constraint satisfaction problems. *Annals of Mathematics and Artificial Intelligence*, 40(3-4):303–352, March 2004.
9. Oliver Kullmann. The `OKlibrary`: Introducing a "holistic" research platform for (generalised) SAT solving. *Studies in Logic*, 2(1):20–53, 2009.
10. Fabio Massacci and Laura Marraro. Logical cryptoanalysis as a SAT problem. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000 Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*, pages 343–375. IOS Press, Amsterdam, 2000. ISBN 1 58603 061 2.

The Min Average Latency Steiner Multigraph Problem: Budget-Constrained Wildlife Corridor Design for Multiple Species

Katherine J. Lai (student) and Carla P. Gomes (supervisor)

Computer Science Department, Cornell University
Ithaca, NY 14853
klai,gomes@cs.cornell.edu

Abstract. We introduce the Min Average Latency Steiner Multigraph Problem, an extension of our recent work concerning wildlife conservation efforts to combat the negative effects of habitat loss and fragmentation affecting endangered species. We model this problem as a network design problem where the goal is to select a set of land parcels (nodes in a graph) which will form a so-called wildlife corridor that connects core habitat areas (terminals) while minimizing the average difficulty (latency) for animals to travel between pairs of locations, subject to a budget constraint. This is an extension of the work on the Steiner Multigraph Problem, a formulation we introduced to find the min-cost wildlife corridor for multiple species with different landscape requirements. In this new extension, we are given a budget constraint and want to find the network that maximizes a connectivity measure based on the Least-Cost Paths model, a popular measure used in landscape ecology. While this is motivated by the wildlife conservation application, this problem can be stated generally and can model applications in other network design applications.

1 Introduction

The study of network design optimization problems has many practical applications. In recent years, techniques in this area have been used to address wildlife conservation problems in computational sustainability [5]. For many endangered species, human development activities have diminished and fragmented their existing habitat. In addition to setting aside parks and reserves, maintaining connectivity between such areas is important to mitigate the negative demographic and genetic consequences that may result from isolated populations [4]. This connectivity problem can be modeled as a network design problem where conservation planners design so-called wildlife corridors, or swaths of preserved land that connect important patches of habitat for the endangered species. These wildlife corridors have been used successfully by wildlife, though the design choices must be made carefully to increase the likelihood of use [8]. Another important computational issue arises given that funds are usually quite limited for these conservation efforts. It is also important to simultaneously consider the various needs of different species for biodiversity and economic reasons [1].

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 43–48, Perugia, Italy, 12 September 2011.

Conrad et al. [2] have modeled the wildlife corridor design problem for a particular species as a *Connection Subgraph problem* which requires connectivity between a set of *terminals* which represent the animal's core habitat areas (terminals). The land parcels under consideration are modeled as graph nodes and are characterized by two nonnegative values: habitat suitability and cost. Edges are drawn between any two land parcels that share a border. A feasible corridor is a subgraph of land parcels that connect all the given terminals. The goal of the problem is then to maximize the total utility of the purchased nodes subject to a budget constraint. Maximizing the utility allows the model to prefer wildlife corridors that contain highly suitable habitat.

Our more recent work [9] introduces the *Steiner Multigraph Problem* to accommodate multiple species which may have different landscape requirements. In particular, each species under consideration may only be able to travel over some particular subset of the landscape based on landscape features such as slope and ground cover. Thus the Steiner Multigraph problem takes as input a set of terminals and a subgraph representing the permeable landscape for each of the species. The objective is then to find the min-cost corridor satisfying these connectivity constraints. As such, it does not address what decisions should be made when additional funds are available under a budget constraint.

In this work, we extend the Steiner Multigraph Problem to find the multi-species wildlife corridor with the best connectivity possible under a budget constraint. There are various connectivity measures that can be considered for optimization, most of which are based on considering the *resistance* value of a node rather than the habitat suitability value. This value quantifies the difficulty or unlikelihood for a particular species to travel across that piece of land. This value can be quite different from habitat suitability; for example, while a particular animal may not live in an open field or in a river, it may still be quite willing to travel across these land features. A popular model for evaluating landscape connectivity in landscape ecology is the Least-Cost Path model, where two locations are considered to have high connectivity if the length of the shortest resistance-weighted path (latency) between them is small [10]. Thus our objective will be to minimize the average latency between pairs of terminals in the solution's subgraph. We could also have chosen to extend the Steiner Multigraph Problem by adopting the approach of the Connection Subgraph model where the sum of the habitat suitability values is maximized. However, given that the goal of conserving a wildlife corridor is to connect locations in a fragmented landscape, using the additional funds to further improve connectivity is perhaps more appropriate.

In this work, we propose the *Min Average Latency Steiner Multigraph Problem*, a budget-constrained extension of the Steiner Multigraph Problem which finds a wildlife corridor that connects habitat areas (terminals) for multiple species while respecting their respective landscape requirements. Given a budget value, the goal is to find a wildlife corridor which also maximizes connectivity between the species' terminals by minimizing the average shortest resistance paths (latency) between pairs of terminals in the corridor.

2 The Problem

2.1 Problem Definition

We will focus on the node-weighted version of the problem because of our motivating application in wildlife corridor design, though it can easily be formulated with weights on the edges instead. Another consequence of the wildlife application is that there is oftentimes a physical limit to how far an animal will travel to explore and find new habitat. We will therefore take as input a set of pairs of terminals for each species to represent pairs of locations that represent feasible destinations for a single animal. This approach is also used in our recent work on Upgrading Shortest Paths [3]. We formally define the problem as follows.

The Min Average Latency Steiner Multigraph Problem (MALSMP)

Given: an undirected graph $G = (V, E)$, an index set I, node sets $V_i \subseteq V$ for all $i \in I$, sets of terminals $T_i \subseteq V_i$ for all $i \in I$, a set of terminal pairs $\mathcal{T}_i \subseteq T_i \times T_i$, resistance values $r_v^i \in \mathbb{R}^+$ for all $i \in I$ and $v \in V_i$, costs c_v for all $v \in V$, and a budget value B.

Find: a node set $W \subseteq V$ such that $\sum_{v \in W} c_v \leq B$, the induced subgraph $G(W \cap V_i)$ connects T_i for each $i \in I$, and a path of nodes $P_{st} \subseteq W \cap V_i$ can be chosen for each terminal pair $(s, t) \in \mathcal{T}_i$ and $i \in I$ such that the average length of the effective shortest resistance path, or *latency*,

$$\frac{\sum_{i \in I} \sum_{(s,t) \in \mathcal{T}_i} \sum_{v \in P_{st}} r(v)}{\sum_{i \in I} |\mathcal{T}_i|}$$

is minimized.

We assume that connecting all the terminal pairs in each $\mathcal{T}_i$ subsequently connects all terminals T_i. If this is not the case, we partition the terminals in T_i based on the connected components implied by connecting all pairs in $\mathcal{T}_i$ and add another index value to I for each additional set of terminal pairs. We also note that as stated, we are averaging all latencies equally across all terminal pairs regardless of their index in I. However, we can easily imagine that each index may represent a species that has a different level of importance compared to the others, or it may have far fewer terminal pairs in comparison to another species that is more widely-distributed. We may therefore want to consider a multi-objective version of the problem where we have an objective for each index $i \in I$ equal to the average latency over all the pairs in $\mathcal{T}_i$. We can address this multi-objective problem by weighting the different objectives differently. This is a trivial modification to the mixed integer programming formulation we will describe, so we will simply assume equal weights for each index $i \in I$ for clarity.

2.2 A Mixed Integer Programming (MIP) Formulation

We can encode MALSMP by formulating each shortest path for a terminal pair $p = (s, t) \in \mathcal{T}_i$ as a single unit of flow traversing the graph $G(V_i)$ from s to t

that incurs a latency equal to the total resistance values included in the path's nodes. Combining the constraints for each terminal pair gives a min-cost multi-commodity flow mixed integer program, and we restrict the use of nodes in the graph for these flows via a common set of variables x_v that indicate which nodes have been conserved. The total cost of conserving the nodes cannot exceed the budget value B. As mentioned previously, we assume that connecting all terminal pairs in a set $\mathcal{T}_i$ implies that all terminals in T_i are subsequently connected. For each pair of terminals $p(s,t)$ where $p \in \mathcal{T}_i$, we have a set of node variables f_v^{ip} for all $v \in V_i$ and two arc variables f_{uv}^{ip} and f_{vu}^{ip} for all edges (u,v) in $G(V_i)$. Using $\Gamma_i(v)$ to indicate the neighboring nodes of v in V_i, we define the following expressions for incoming and outgoing flow at each node in the graph:

$$in_v^{ip} = \sum\nolimits_{u \in \Gamma_i(v)} f_{uv}^{ip} \tag{1}$$

$$out_v^{ip} = \sum\nolimits_{u \in \Gamma_i(v)} f_{vu}^{ip} \tag{2}$$

The complete MIP is shown in Figure 1. Constraints (4)-(5) force one of the terminals s in a terminal pair $p = (s,t) \in \mathcal{T}_i$ to be the source of 1 unit of flow for the other terminal t. The corresponding sink constraints are captured by (7)-(8). Constraints (6) and (9) force the endpoints s and t of each flow to be in the solution. Constraints (10) and (11) capture flow conservation for all other nodes while preventing flow through a node v if f_v^{ip} is set to 0. By (12), the node flow variable f_v^{ip} can only be positive if v has been chosen for the solution, i.e. x_v has been set to 1. The last two constraints (15) and (16) force the indicator variables to be integer and the flow to be nonnegative. An optimal solution to this MIP is a Steiner Multigraph solution where the average shortest path lengths over all the terminal pairs is minimized as given by the objective function (3). Each set of terminals must be connected by the set of nodes $V' = \{v \in V : x_v = 1\}$ since there must be a feasible flow from s to t, and any such flow must use some path in V_i that can only travel via nodes that have nonzero capacity, i.e. nodes for which $x_v = 1$.

3 Solution Methods

Given the MIP encoding of the MALSMP, a straightforward method for solving the problem is to use a MIP solver such as CPLEX. In the original Connection Subgraph problem, it was shown that a hybrid approach that first solves the min-cost Steiner tree problem before using the result as an initial feasible solution for the MIP solver makes a large impact on the running time [6]. We expect that such an approach will also be helpful here. However, it has been shown that solving the min-cost Steiner Multigraph problem is NP-hard even for planar instances with a constant number of terminals, so it may be faster in practice to only find an approximate solution as a first step [9]. Other solution approaches to be explored and tested include local search algorithms such as simulated annealing. We have not yet implemented and tested these solution methods.

$$\min \quad \frac{1}{\sum_{i \in I} |\mathcal{T}_i|} \sum_{i \in I} \sum_{p \in \mathcal{T}_i} path_{ip} \tag{3}$$

$$\text{s.t.} \quad in_s^{ip} = 0 \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i \tag{4}$$

$$out_s^{ip} = 1 \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i \tag{5}$$

$$f_s^{ip} = 1 \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i \tag{6}$$

$$in_t^{ip} = 1 \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i \tag{7}$$

$$out_t^{ip} = 0 \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i \tag{8}$$

$$f_t^{ip} = 1 \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i \tag{9}$$

$$in_v^{ip} = f_v^{ip} \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i, v \notin \{s,t\} \tag{10}$$

$$f_v^{ip} = out_v^{ip} \qquad \forall i \in I, p = (s,t) \in \mathcal{T}_i, v \notin \{s,t\} \tag{11}$$

$$f_v^{ip} \leq x_v \qquad \forall i \in I, p \in \mathcal{T}_i, v \in V_i \tag{12}$$

$$path_{ip} = \sum_{v \in V_i} r_v^i f_v^{ip} \qquad \forall i \in I, p \in \mathcal{T}_i \tag{13}$$

$$\sum\nolimits_{v \in V} c_v x_v \leq B \tag{14}$$

$$x_v \in \{0,1\} \qquad \forall v \in V \tag{15}$$

$$f_v^{ip}, f_{uv}^{ip} \geq 0 \qquad \forall i \in I, \forall p \in \mathcal{T}_i, \forall v \in V_i, \forall u \in \Gamma_i(v) \tag{16}$$

Fig. 1. Using the expressions from (1) and (2), this min-cost multicommodity flow MIP exactly captures MALSMP. For each species i, a unique commodity is defined for each terminal pair $p = (s,t) \in \mathcal{T}_i$, and exactly 1 unit of flow of this type must travel from the source s to the sink t. Each flow incurs a resistance value or latency equal to the sum of the resistances of the path's nodes. Nodes can only carry flow if they have been bought, and the total cost of the bought nodes must not exceed the budget B.

4 Discussion

In this work, we propose the Min Average Latency Steiner Multigraph Problem (MALSMP) to capture the network design problem of finding a well-connected wildlife corridor for multiple species under a budget constraint. While motivated by wildlife applications, this problem can also be used to model network design problems in other applications such as the design of multicast networks. For example, researchers working on multicast networks have studied a very similar problem where instead of optimizing the average latency, the goal is to find the min-cost Steiner tree for which the diameter, or equivalently the maximum latency, is constrained by an upper bound [7].

In MALSMP, we optimize the average connectivity measure under the Least-Cost Paths model. We could also consider other objective functions such as minimizing the maximum latency or using a different connectivity model that reflects the benefit of having redundancy in the graph, i.e. a measure of robustness against node failures. As future work, it would be interesting to find a good way to define and optimize over such a connectivity measure.

References

1. Beier, P., Majka, D.R., Spencer, W.D.: Forks in the road: Choices in procedures for designing wildland linkages. Conservation Biology 22(4), 836–851 (2008)
2. Conrad, J., Gomes, C.P., van Hoeve, W.J., Sabharwal, A., Suter, J.: Connections in Networks : Hardness of Feasibility versus Optimality. In: CPAIOR. pp. 16–28 (2007)
3. Dilkina, B., Lai, K.J., Gomes, C.P.: Upgrading shortest paths in networks. In: CPAIOR. pp. 76–91 (2011)
4. Gilpin, M., Soule, M.: Minimum viable populations: processes of extinction. In: Soule, M. (ed.) Conservation Biology: The Science of Scarcity and Diversity, pp. 19–34. Sinauer Associates (1986)
5. Gomes, C.: Computational sustainability: Computational methods for a sustainable environment, economy, and society. The Bridge 39(4) (2009)
6. Gomes, C.P., van Hoeve, W.J., Sabharwal, A.: Connections in networks: A hybrid approach. In: CPAIOR. pp. 303–307 (2008)
7. Gouveia, L., Magnanti, T.L.: Network flow models for designing diameter-constrained minimum-spanning and Steiner trees. Networks 41(3), 159–173 (May 2003)
8. Haddad, N.M., Bowne, D.R., Cunningham, A., Danielson, B.J., Levey, D.J., Sargent, S., Spira, T.: Corridor Use By Diverse Taxa. Ecology 84(3), 609–615 (Mar 2003)
9. Lai, K.J., Gomes, C.P., Schwartz, M.K., McKelvey, K.S., Calkin, D.E., Montgomery, C.S.: The Steiner Multigraph Problem: Wildlife Corridor Design for Multiple Species. In: AAAI, Special Track on Computational Sustainability and AI (2011)
10. Singleton, P.H., Gaines, W.L., Lehmkuhl, J.F.: Landscape permeability for large carnivores in washington: a geographic information system weighted-distance and least-cost corridor assessment. Res. Pap. PNW-RP-549: U.S. Dept. of Agric., Forest Service, Pacific Northwest Research Station (2002)

cumulatives_trajectories: a Constraint for Modelling Preemptive Reassignable Tasks with Momentarily Resource Consumption *

Arnaud Letort (*student*) and Nicolas Beldiceanu (*supervisor*)

TASC team, (EMN-INRIA,LINA) Mines de Nantes, France
{arnaud.letort,nicolas.beldiceanu}@mines-nantes.fr

Abstract. This paper presents a constraint for modelling cumulative problems with preemptive reassignable tasks and temporary resource consumption. We design this constraint to tackle problems where a task can be dynamically reassigned (i.e. can migrate) to some alternative resource during its execution. While migrating, tasks consume additional resources such as CPU, RAM, I/O.

Keywords: cumulative, preemption, planning, migration, data centre

1 Motivation

Data centres involve a number of servers (called *resources* in the following) running virtual machines (i.e., *tasks*) such as web services or scientific computations. In order to adapt themselves to variations of the workload during time, a crucial operation consists of dynamically migrating tasks between resources [1], [6] (e.g., if we want to adapt the number of active resources w.r.t. the current workload). In order to be feasible, a migration requires temporarily additional resources such as CPU, memory or network which depend on the nature of the task. Because of the dynamic character of migrations which are not known a priori, current approaches [8], [10] that try to optimise the number of active servers, solve a sequence of optimisation problems, each of them being related with successive variations on the overall workload [9]. Consequently they are limited in the way they can anticipate future workload variations and consider temporary consumptions associated with migrations when solving the optimisation problem at a given step, i.e., they may be too greedy. Motivated by this problem, the contribution of this paper is a new constraint, *cumulatives_trajectories* (called *ct* in the following), which can directly model such a situation with a single constraint. The advantage of *ct* are twofold:

– First, *ct* allows to directly model tasks that can potentially migrate from one resource to an other, i.e., it allows encoding a planning problem [2] rather than a scheduling problem. On the one hand, the word *trajectories*

* This research was founded in part by the SelfXL project of the French National Research Agency (ANR).

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 49–54, Perugia, Italy, 12 September 2011.

stems from the fact fact some task can move from one source resource to an other target resource several times, i.e., the trajectory of a task consists of the sequence of moves of the task. On the other hand, the word *cumulatives* indicates that each resource aspect attached to a resource has its own limited capacity.
- Second, *ct* also allows to directly model detailed operational rules on how a task uses additional resources (i.e., *when* and *how much*) during a migration.

Section 2 provides an informal description of the constraint and introduces a running example that will be used throughout this paper. Section 3 defines the arguments of the constraint and Section 4 describes future work.

2 Informal Description of *ct* and Running Example

For each task we describe its resource consumption profiles on the different resource aspects it actually uses. The trajectory of a task is defined by a maximum number of slices, where a migration or a suspension separate two consecutive slices.[1] The different resource aspects associated with a resource must satisfy a *cumulative* constraint w.r.t. all tasks slices assigned to that resource. Depending on the migration type, additional resource aspects may be temporarily used on the source/target resources during the migration of the task.

Example 1. A data centre is a centralised repository, composed of interconnected servers, for the storage and management of data. Here we consider three different resource aspects, (i) RAM (i.e. Random-Access Memory), (ii) CPU (i.e. Central Processing Unit) and (iii) link capacity. A *server* is a physical machine with a fixed amount of RAM and CPU, connected to others via a network. A *virtual machine* is a process which requires a certain amount of CPU and RAM that may evolve over time to run. Servers and network links correspond to resources in our context, whereas virtual machines correspond to tasks. Figure 1 presents a running example with two servers connected by a network link and a task that migrates from server one to server two. □

3 Defining the *ct* Constraint

The constraint *ct* has the following form :

$$ct(Resources, Tasks, Network, Migrations, Ratios)$$

We follow the style use in the Constraint Catalog [4, p. 6-9] for describing the constraint arguments. A $collection(A_1, A_2, \ldots, A_n)$ corresponds to a collection of ordered items, where each item consists of $n > 0$ attributes. Each attribute is an expression of the form a-T, where a is the name of the attribute and T its type. The basic data types *int* and *dvar* respectively denote an *integer* and a *domain variable*.

[1] For space reason we do not speak about suspension in the rest of this paper.

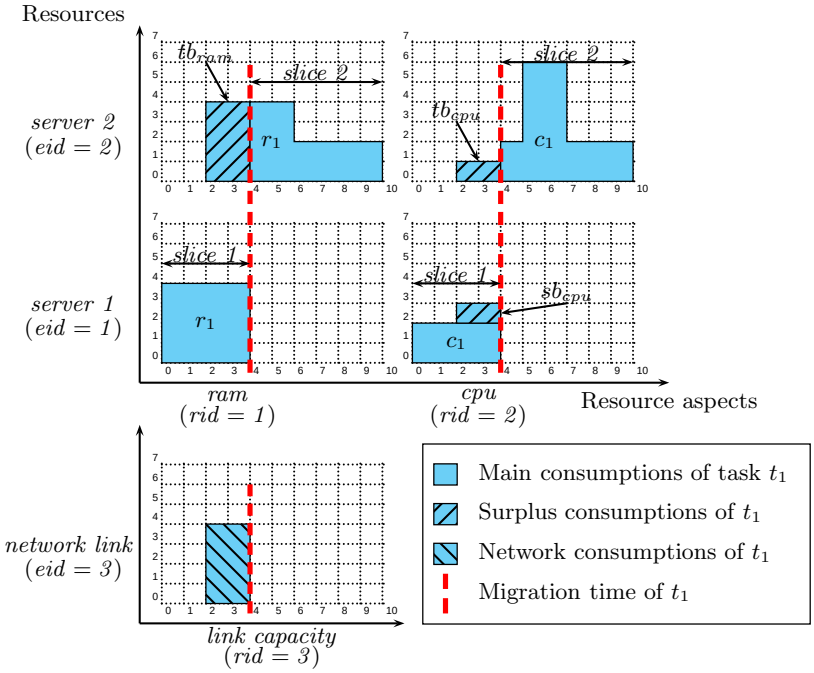


Fig. 1. (running example) During the four first time units, task t_1 is assigned to resource one (slice one of t_1) where it uses some ram and cpu (in grey), then it migrates to resource two (slice two of t_1). This migration provokes three extra consumptions (i.e. two for the cpu and one for the ram) and a network link consumption. These additional consumptions (shown in dashed) are caused by the copy of the ram.

3.1 Resources

Resources is a collection where each resource is identified by a unique identifier, its different resource aspects and corresponding capacities. Each resource aspect is also identified by a unique integer.

Resources : $collection(eid, rcapa)$

- eid: int. Identifier of the resource.
- $rcapa$: $collection(rid\text{-int}, capa\text{-int})$. Set of different fixed constant capacities, one for each resource aspect that the resource has (i.e. $capa > 0$).

Example 2. (declaration of resources of the running example) A server is declared as a resource with two different resource aspects : (i) RAM and (ii) CPU. By convention, we identify the RAM by $rid = 1$ and the CPU by $rid = 2$. A network link between two servers is a resource with only (iii) link capacity.

$\langle (eid-1, rcapa-\langle (rid-1, capa-5), (rid-2, capa-4)\rangle)$ %server 1
$(eid-2, rcapa-\langle (rid-1, capa-5), (rid-2, capa-7)\rangle)$ %server 2
$(eid-3, rcapa-\langle (rid-3, capa-4)\rangle)\rangle$ % link between server 1 and 2 □

3.2 Tasks

The *Tasks* argument corresponds to a collection of tasks where each task consumes resources between its start and end times, one or more resource aspects according to its resource consumption profiles. Throughout its execution, a task must be assigned to a resource with corresponding resource aspects. It can migrate from a resource to another.

Tasks : *collection*(*tid*, *start*, *effdur*, *rundur*, *end*, *cons*, *sl*, *nbslices*, *preemptive*, *mid*)

- *tid*: int. Identifier of the task.
- *start*, *end*: dvar. Start/End time of the task.
- *effdur*: dvar. Effective duration i.e. pauses included.
- *rundur*: dvar. Running duration, i.e. pauses excluded.
- *cons* : *collection*(*rid*-int,*dur*-int,*height*-int). This collection describes all resource consumption. Since the resource consumption of a task on a resource aspect *rid* may vary over time it is given as a profile (i.e. a sequence of consecutive intervals where each interval is defined by its duration *dur* and height *height*). The sum of attributes *dur* of each profile is equal to *rundur*.
- *sl* : *collection*(*eid*-dvar, *len*-dvar, *pause*-dvar). This collection gives the trajectory of the task over time (i.e. a sequence of slices), relatively to its start date. The task is assigned to the resource *eid* during *len* time unit, followed by a suspension of *pause* time unit.
- *nbslices*: dvar. Only the *nbslices* first elements of *sl* are relevant.
- *preemptive*: int. Indicates whether the task can be suspended (0) or not (1).
- *mid*: int. Identifier of the migration mode used by the task.

Example 3. (tasks of the running example) A virtual machine is considered as a task which consumes an amount of CPU and RAM evolving over time. The following task consumes resources over ten time units and migrates one time, from resource one to resource two.

$[tid-1, start-0, effdur-10, rundur-10, end-10,$
$\quad cons-\langle[rid-1, dur-6, height-4],$ % RAM consumption profile
$\qquad\quad [rid-1, dur-4, height-2],$ % see r1 on Figure 1
$\qquad\quad [rid-2, dur-5, height-2],$ % CPU consumption profile
$\qquad\quad [rid-2, dur-2, height-6]),$% see c1 on Figure 1
$\qquad\quad [rid-2, dur-3, height-2]\rangle,$
$\quad sl-\langle[eid-1, len-4, pause-0],$ % see *slice 1* on Figure 1
$\qquad [eid-2, len-6, pause-0]\rangle,$ % see *slice 2* on FIgure 1
$\quad nbslices-2, preemptive-0, mid-1]$ □

3.3 Execution Modes and Momentary Consumptions

We first introduce some terminology about various aspects of migration.

Definition 1. *A* surplus consumption *is a temporary additional consumption induced by the migration of a task. For each migration, there are for each resource*

aspects four potential surplus consumption, configurable both in duration and height. Two of them are located on the source resource, while the two other are situated on the target. Duration and height of the four potential surplus consumption are given by the attribute mres *of the argument* Migrations.

From now on, the term "main consumption of a task" stands only for its consumptions described by attribute *cons*, i.e. we exclude its surplus consumption introduced by Definition 1.

Definition 2. *The* migration time *is the first instant when the task stops consuming resource on the source resource. Let* $sl[i]$ *and* $sl[i+1]$ *be two consecutive slices of a task, standing for a migration. The* migration time *is given relatively to the begin of the task by* $\sum_{k=0}^{k\leq i}(sl[k].len + sl[k].pause)$.

Definition 3. *The* migratory offset *is the size of the time interval during which while migrating, the task does not consume any resource in terms of main consumption. The* migratory offset *is the maximum between the corresponding attribute* noffset *(of the argument* Network*) and the result of the computation given by the attribute* moffset *(of the argument* Migrations*).*

Definition 4. *The* migration interval *is the time interval during which the task consumes resources on additional resources associated with the migration from the source to the target. This time interval is computed with the attribute* minterv *of the argument* Migrations.

Ratios Migrating a task from a source resource to a target resource usually requires additional resources depending on the current consumption of the task. The argument *Ratios* permits to express duration and height of these surplus consumption w.r.t. the consumption on a given resource aspect of the task.

Ratios is a collection where each item is defined by an identifier *ratid*-int, a numerator *num*-int, a denominator *den*-int, a constant *cst*-int, and a resource aspect reference r.

The calculation of durations and heights has the following form : $\lceil \frac{num*h(r)}{den} \rceil + cst$, where $h(r)$ stands for the height of resource aspect r consumed by the task at the migration time.

Network The argument *Network* is defined by a set of source/target resources for which we specify the corresponding links and traversal time.

Network : *collection*(*eids*, *eidt*, l, *noffset*) with :

- *eids*: int. Identifier of the source resource
- *eidt*: int. Identifier of the target resource.
- l : *collection*(*eid*-int). Resources used by a task during its migration from *eids* to *eidt*. They will all be used during the migration interval given by the migration mode of task t (i.e. attribute *mid*).
- *noffset*: int. The time required for a task to go from *eids* to *eidt* using resources given in l.

Migrations Migrating a task may introduce both extra delay and/or extra resource consumptions on the source and target resource or on other resources (network links). For space limitation reason we do not detail this argument in the paper.

4 Conclusion

We introduce a constraint, *cumulatives_trajectories*, for modelling problems with preemptive reassignable tasks. It allows to model the workload distribution problem in data centres and others like the Airport Parking Assignment [13], [7]. In the future we will focus on light filtering algorithms for scalability reasons.

Acknowledgements We thank Jean-Marc Menaud and Fabien Hermenier for introducing us to the topic of data centres.

References

1. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Proceedings of 19th ACM. pp. 164–177. SOSP '03, New York, NY, USA (2003)
2. Barták, R., Toropila, D.: Enhancing constraint models for planning problems. In: Lane, H.C., Guesgen, H.W. (eds.) Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference, May 19-21, 2009, Sanibel Island, Florida, USA. AAAI Press (2009)
3. Beldiceanu, N., Carlsson, M.: A New Multi-Resource *cumulatives* Constraint with Negative Heights. In: Van Hentenryck, P. (ed.) Principles and Practice of Constraint Programming (CP'2002). LNCS, vol. 2470, pp. 63–79. Springer-Verlag (2002), preprint available as SICS Tech Report T2001-11
4. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog (2010)
5. Celesti, A., Tusa, F., Villari, M., Puliafito, A.: Improving virtual machine migration in federated cloud environments. Evolving Internet, International Conference on pp. 61–67 (2010)
6. Clark, C., Fraser, K., Hand, S., Hansen, J., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: In Proceedings of the 2nd ACM/USENIX NSDI. pp. 273–286 (2005)
7. Dincbas, M., Simonis, H.: APACHE - A Constraint Based, Automated Stand Allocation System. In: ASTAIR'91. pp. 267–282. Royal Aeronautical Society, London
8. Hermenier, F.: Gestion dynamique des tâches dans les grappes, une approche à base de machines virtuelles. Ph.D. thesis, Université de Nantes (11 2009)
9. Hermenier, F., Demassey, S., Lorca, X.: The bin-repacking scheduling problem in virtualized datacenters. In: CP'11. Lecture Notes in Computer Science, Springer-Verlag, Perrugia, Italy (2011)
10. Hermenier, F., Lawall, J., Menaud, J.M., Muller, G.: Dynamic Consolidation of Highly Available Web Applications. Research Report RR-7545, INRIA (02 2011)
11. Poder, E., Beldiceanu, N.: Filtering for a Continuous Multi-Resources cumulative Constraint with Resource Consumption and Production. In: ICAPS'08. Sidney
12. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Computer Communications Review 33, 2003 (2002)
13. Simonis, H.: Models for global constraint applications. Constraints 12, 63–92 (2007)

Reinforced Adaptive Large Neighborhood Search

Jean-Baptiste Mairy (UCLouvain, Belgium)
jean-baptiste.mairy@uclouvain.be

Directors: Yves Deville (UCLouvain), Pascal Van Hentenryck (Brown University)

1 Introduction

CP-based Large Neighborhood Search (LNS) combines Local Search (LS) and Constraint Programming (CP); at each step of LS, a fragment (subset) of the variables is relaxed and CP is used to search the neighborhood in order to find a new solution. LNS is known to be effective on numerous problems at quickly finding good solutions [1–5]. When designing a LNS model, one has to specify the size of the fragments and how the fragments are chosen. They can be chosen specifically for the problem, or a simple random strategy can be used. The specific approach has the advantage of being quite effective. However, it requires some knowledge on the problem. Even for well-known problems, setting the right values for the parameters is a difficult task. Random parameters have the advantage of being totally generic but can be less performant.

The objective of this research is to develop generic heuristics for an adaptive selection of the fragments in LNS. This work uses Reinforcement Learning to adapt the heuristics during the search.

CP-Based Large Neighborhood Search This LNS approach takes advantage of the expressiveness of Constraint Programming (CP) and the speed of Local Search (LS). Its working principle is to maintain a *candidate solution* through the search that is *not* violating any constraint but that may not be optimal (or not known to be). The successive solutions are obtained by repeating the two following operations until a stopping criterion is met.

1. *neighborhood definition*: this step consists of choosing the set of variables (fragment) that will be relaxed to their original domains while fixing the other variables to their values in the current best solution. The domains of the relaxed variables define the neighborhood of the current solution that will be explored with CP at the next step.
2. *neighborhood exploration*: this step consists of using CP to explore the restricted problem defined by the relaxation of the fragment. When an improving solution is found, the current best solution is replaced. It is replaced only if CP provides a better solution. A limit in time or in number of failures is specified to avoid spending too much time in exploring the neighborhood.

Although the LNS framework is really simple, its parameters (the size of the fragments, the fragment selection procedure and the limit on the CP exploration

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 55–60,
Perugia, Italy, 12 September 2011.

step) must be carefully chosen. No general principle has emerged yet on how to choose the fragments during the search. Practitioners often prefer to relax random variables to favor diversification.

Reinforcement Learning Framework In this paper, Reinforcement Learning is used to develop adaptive strategies to select the fragments in LNS. Reinforcement Learning [6] is all about learning while doing. For this work, it means that good fragments are learned while selecting the variables to relax at each LNS iteration.

The general picture of Reinforcement Learning uses the notion of *agent*. The task of the *agent* is to select *actions*. For each *action* chosen, it gets a *reward*. Its goal is to maximize those *rewards*. The best *action* to choose vary given the *state* of the *external world*. Each *action* results in a *reward* but also in a new *state* for the *external world*. In order to choose the *actions*, the agent maintains *internal quantities*. Those quantities usually reflect the quality of the states and state-action pairs. They are updated each time a new *reward* is obtained. The name Reinforcement Learning comes from the fact that those quantities are used to select *actions* while being updated with respect to the result of the selected *actions*.

All the methods described in this paper use Reinforcement Learning to select the fragments. The fragment selector selects fragments based on the internal quantities that it maintains. It updates those quantities based on the rewards. The rewards are defined with respect to the output of the CP search. The notion of *state* is not used to define the heuristics.

The next sections present different applications of Reinforcement Learning to LNS. Section 2 presents an Inclusion Quality Based fragment selection. Section 3 describes a Variable Quality Based fragment selection. Those two first techniques are targeted at selecting variables that are important for reoptimizing the objective function of the problem. Section 4 presents a Relation Based fragment selection. This last technique is trying to select fragments composed of related variables.

The techniques are all described in terms of the internal quantities of the fragment selector, the computation of the rewards, the update of the internal quantities and the selection of the fragments.

2 Inclusion Quality Based Selection

This technique performs an action for each variable independently. For each variable, the two possible actions are inclusion in the fragment and non-inclusion. Choosing one action for every variable results in the selection of a fragment. The goal is to detect and include variables that are important for optimizing the objective function.

Internal Quantities: for each variable x, this selector maintains:

$Q[x, 0]$: quality of non-selection of x

$Q[x, 1]$: quality of selection of x

Those quantities are positive during the entire search.

Rewards: the rewards are computed with respect to the output of the CP search:

$$reward = \begin{cases} 50 + 0.01 * nbIt & \text{if objective improved} \\ -1 - 0.01 * nbIt & \text{else} \end{cases} \quad (1)$$

where $nbIt$ represents the number of LNS iterations. In case of non-improvement, the reward is a penalty. Adding a fraction of the number of LNS iterations gives more importance to the rewards at the end of the search. Improving the objective at the end of the search is more difficult than at the beginning.

Updates: for each variable x of the previously selected fragment, the quantities are updated with the following rules: On improvement, increase the quality of selection:

$$Q[x, 1] = Q[x, 1] + \alpha * (reward - Q[x, 1]) \quad (2)$$

where α is a constant parameter of the technique. This increase is only applied to variables that have changed value. In case of non-improvement, increase the quality of non-selection:

$$Q[x, 0] = Q[x, 0] + \alpha * (-reward - Q[x, 0]) \quad (3)$$

The intuition behind those update rules is that they gather the rewards as quality of selection and the penalties as quality of non-selection. The penalties are made positives to keep the quality of non-selection positive. As the parameter α is constant, those updates perform exponential smoothing on the rewards gathered. This allows to reduce more or less (depending on the value of α) the influence of early rewards. Reducing the influence of the early rewards allows those heuristics to take the non-stationarity of the problem into account. The selection of fragment is a non-stationary problem because the good fragments to relax do not converge to a fixed fragment.

Fragment Selection: the fragment selection uses the following probability of selection (versus non-selection) for each variable x:

$$P[x] = \frac{Q[x, 1]}{Q[x, 0] + Q[x, 1]} \quad (4)$$

For each variable x independently, it is included in the fragment with probability $P[x]$. For this technique, the size of the fragment is automatically adjusted. If too large fragments are selected, leading to non-improvement of the objective, the probability of selection of all its variable decreases. This reduces the expected future fragment size.

3 Variable Quality Based Fragment Selection

This section defines variable quality based selectors. For this selector, a quantity (the quality of the variable) is maintained for each variable. It tries to capture the importance of the variable for the objective function. The goal is to select fragments composed of variables that will allow a reoptimization of the objective function. Two selectors are presented. They differ in the utilization of the internal quantities.

Internal Quantities: for each variable x, this selector maintains:

$$V[x]: \text{ the quality of the variable } x \tag{5}$$

Those quantities are positive during the entire search.

Rewards: the rewards depend on the output of the CP search:

$$reward = \begin{cases} 50 & \text{if objective improved} \\ 0 & \text{else} \end{cases} \tag{6}$$

There is no need to add a fraction of the number of iterations here. The quality of a variable is relative to the quality of the other variables.

Updates: both when improvement and non-improvement, the update rule for each variable x of the previous fragment is the following.

$$V[x] = V[x] + \alpha * (reward - V[x]) \tag{7}$$

where α is a constant parameter of the technique. The intuition behind this update rule is again the collection of the rewards. As α is constant, the updates perform exponential smoothing. This reduces the influence of early rewards. When improvement, the reward is only granted to variables that have changed value.

Fragment Selection: Two different fragment selectors are defined. They Both select variables until a given fragment size is attained. They can be described as:

1. Iteratively select a variable x among available ones with probability of being selected proportional to $V[x]$
2. Include in the fragment variables with the highest quality V

4 Relation Quality Based Fragment Selection

This section presents various relation quality based selectors. For each pair of variables, they maintain a quantity representing the quality of the relation between each pair of variables. The goal is to be able to select fragments composed of closely related variables. All the selectors of this section share the same internal quantities. Although, two different reward systems and update rules are defined and there are five different utilizations of the quantities for each reward-update.

Internal Quantities: for each pair of variables (x, y), this selector maintains

$$Q[x, y]: \text{ the quality of the relation between } x \text{ and } y \tag{8}$$

Those quantities try to capture the importance of a variable for another in a fragment. For instance, if two variables are linked by a strong constraint, it is important to relax them simultaneously to allow them to change value. The quality of the relation between the two variables should then be high. The matrix Q is kept symmetric and all its entries are positive during the search.

Rewards: two different reward systems are defined:

$$reward_1 = \begin{cases} 50 & \text{if objective improved} \\ 0 & \text{else} \end{cases} \tag{9}$$

and

$$reward_2 = \begin{cases} 50 & \text{if objective improved} \\ -10 & \text{else} \end{cases} \tag{10}$$

In the second reward system, the reward corresponds to a penalty when no improvement has been made.

Updates: two update rules are defined. The first uses exponential smoothing with the first reward system. The second uses the second reward system without exponential smoothing. They are detailed hereafter. They are applied for each pair of variables (x, y) of the previously selected fragment.

With exponential smoothing:

$$Q[x, y] = Q[x, y] + \alpha * (reward_1 - Q[x, y]) \tag{11}$$

where α is a constant parameter of the technique. Without exponential smoothing:

$$Q[x, y] = Q[x, y] + reward_2 \tag{12}$$

For this last update rule, the positivity of the elements of Q is restored after each update.

The intuition behind those updates is again the aggregation of the rewards. In both cases, when an improvement is made, the reward is only granted between pairs of variables that have changed value.

Fragment Selection: five fragment selection techniques are defined. Each of them can be used with any update rule. They all start by picking a random variable and iteratively select one variable at a time until a given fragment size is attained. Those fragment selections are detailed hereafter:

1. Select variable x among available ones with probability of being selected proportional to

$$\sum_{y \in fragm} Q[y, x] \tag{13}$$

2. Select variable x among available ones with maximum

$$\sum_{y \in fragm} Q[y, x] \tag{14}$$

3. Select variable randomly among the 10 variables maximizing the quantity

$$S[x] = \sum_{y \in fragm} Q[y, x] \tag{15}$$

4. Let $lastVar$ be the last variable included in the fragment. Select variable randomly among the 10 variables maximizing $S[x] = Q[lastVar, x]$
5. Let $lastVar$ be the last variable included in the fragment. Select variable among the 10 variables maximizing $S[x] = Q[lastVar, x]$ with probability of being selected for x proportional to

$$\sum_{y \in fragm} Q[y, x] \tag{16}$$

All those selectors are targeted at selecting fragments with related variables. Such fragments allow large search spaces for the CP search.

5 Conclusion

This paper presents preliminary results defining various heuristics to select the fragments in Large Neighborhood Search. These heuristics use the Reinforcement Learning framework. We are now experimentally comparing these heuristics on numerous problems to assess their performance. They will also be compared to state of the art specific and generic heuristics. We also intend to use learning techniques to dynamically choose the size of the fragment and the limit on the CP exploration step. The remaining parameters of the techniques (such as α and the values of the rewards) will also be studied.

References

1. Laurent Perron, Paul Shaw and Vincent Furnon. Propagation Guided Large Neighborhood Search. CP 2004, LNCS 3258 (2004) 468–481.
2. Daniel Godard, Philippe Laborie and Wim Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. AAAI 2005.
3. Guy Desaulniers, Eric Prescott-Gagnon and Louis-Martin Rousseau. A Large Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows. MIC 2007.
4. Emilie Danna and Laurent Perron. Structured vs. Unstructured Large Neighborhood Search : A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. CP 2003, LNCS 2833 (2003) 817–821.
5. Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems CP 1998, LNCS 1520 (1998) 417–431.
6. Richard S. Sutton, Andrew G. Barto. Reinforcement learning : an introduction Cambridge, MIT Press, 1998

A Relaxation-Guided Approach for Vehicle Routing Problems with Black Box Feasibility

PhD student: Florence Massen
Advisors: Yves Deville, Pascal Van Hentenryck

ICTEAM, Université catholique de Louvain, Belgium

Keywords: Local Search, Set-Covering, Vehicle Routing, Black Box Feasibility

Abstract. This paper proposes an abstraction of emerging vehicle routing problems, the Vehicle Routing Problem with Black Box Feasibility. In this problem the routes of a basic VRP need to satisfy an unknown set of constraints. A black box algorithm, considered of non-linear complexity, is provided to test the feasibility of a route. The complexity of the problem under consideration lies in the unknown problem structure and the expensive feasibility check. Practical examples of such problems are combinations of VRP with Loading problems or VRP with Scheduling problems. We propose a column generation-based approach to locally optimize this problem. Columns are heuristically generated by so-called collector ants, executing a construction heuristic and guided by pheromones. The pheromones are not deposited by the ants themselves, but by an oracle. This oracle judges which edges should be preferred by the ants based on the solution of the relaxed problem.

1 Introduction

Vehicle Routing Problems have received a great deal of attention since as early as the 1960's. While initially only basic variants have been considered, during the following decades research has focused on more complex variants, such as problems with time windows or with pick-up and delivery. In recent years so-called Rich Vehicle Routing Problems have been tackled. They strive to give a more realistic representation of problems encountered in real-world industry. Such problems often require handling the combination of different complicating constraints, the majority of which are typically considered individually in literature. This paper introduces a generalized VRP variant, the VRP with Black Box Feasibility (VRPBB). The problem is an extension of basic VRPs. Besides respecting the VRP constraints (capacity, time windows, ...) each route needs to verify an unknown set of constraints C. The feasibility of a route with respect to C is verified by a black box algorithm. This verification is considered of non-linear complexity. The VRPBB thus considers a VRP with unknown hard intra-route constraints. This generalization allows to accommodate emerging routing problems with hard side constraints for which no efficient feasibility check is available. Typical applications would be combinations of routing with loading (3L-CVRP [5]) or routing with scheduling (VRPTW with Driver Regulations [7]). The challenge in this problem stems obviously from the unknown problem structure and the cost of the feasibility checks. We propose to reformulate the VRPBB as a set-covering problem. This allows to maximize the utility of the expensive feasibility checks and allows to move freely in a possibly sparse neighborhood. We

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 61–66, Perugia, Italy, 12 September 2011.

propose a (non-exact) column generation-based approach to address the problem. Agents called collector ants heuristically generate and collect columns (routes) of negatived reduced cost (with potential to improve the objective of the relaxed problem). The collector ants are guided by a measure of attractiveness of each edge called pheromone deposits stemming from an external oracle. This oracle computes pheromone deposits in function of the current relaxed solution.

2 The Vehicle Routing Problem with Black Box Feasibility

2.1 Capacitated Vehicle Routing Problem

The Capacitated Vehicle Routing Problem is the most basic vehicle routing problem and underlying to most VRP variants. It is defined on a complete, undirected and weighted graph $G = (V, E)$ where $V = \{0, 1, ..., n\}$ is a set of $n + 1$ vertices and E the set of weighted edges connecting every pair of vertices. Vertex 0 represents the depot while vertices $1, ..., n$ are the n customers to be served. The weight c_{ij} $(i, j = 0, ..., n : i \neq j)$ of an edge e_{ij} corresponds to the cost of traveling from vertex i to vertex j. The homogeneous fleet is limited to K vehicles, each associated with a maximum capacity D. With each customer i $(i = 1, ..., n)$ is associated a demand d_i. Let for the remainder of this paper $first(r)$ $(last(r))$ designate the first (last) customer vertex visited by route r. Let also $next(i, r)$ $(prev(i, r))$ designate the vertex visited after (before) customer i in route r.
Finally the goal is to devise a solution composed by at most K routes such that each customer is visited by exactly one route, such that the sum of demands of the customers on a route does not exceed the maximum capacity D and such that the total traveling cost, equal to the sum of the weights of traversed edges, is minimized. For more information on the CVRP its extensions, and optimization methods see [9].

2.2 Feasibility Black Box

In the VRPBB each feasible route, besides satisfying the constraints associated with the underlying VRP variant, must verify an unknown set of constraints C. Let $feas(r, c) = true$ indicate that route r satisfies constraint $c \in C$. A tentative route r, satisfying the constraints associated with the underlying VRP variant is considered feasible if and only if $\bigwedge_{c \in C} feas(r, c)$. The black box provides a function returning a boolean indicating the feasibility of route r with respect to C. This function is computationally expensive and of non-linear complexity.
In the following routes that are feasible wrt the constraints of the underlying VRP variant are called VRP-feasible. Routes that are feasible wrt C are called C-feasible. Routes that are called feasible are as well VRP-feasible as C-feasible.

3 Proposed Approach for the VRPBB

The presence of the unknown constraint set C entails that no assumptions can be made on the structure of the search space. A local search approach, commonly used for Vehicle Routing Problems, seems unadapted for several reasons. Contiguous feasible regions

in the search space can not be guaranteed. To be able to move freely through the search space it might therefore be necessary to pass through infeasible regions. Since no degree of violation can be obtained from the black box it is impossible to include an appropriate penalty in the objective function. For these reasons, it has been preferred to solve the problem as a set-covering problem over routes, as opposed to a vehicle routing problem. A route generation, as opposed to a solution generation approach allows furthermore to defer the respect of the fleet size constraint to the overlying set-covering master problem. We propose a column generation-based approach to tackle the VRPBB. Columns correspond to feasible routes. At each iteration columns are heuristically generated using a system of collector ants which are guided by pheromones. Each iteration an entire colony of M identical ants is executed. The columns returned by these ants are used to enrich the set of columns on which the relaxed set-covering problem is solved. The relaxed solution is then used to update the pheromones guiding the collector ants. An integer solution is obtained at the end by solving the integer problem on the available set of columns using the open-source SCIP solver. The overall algorithm is depicted in Algorithm 1, which will be described hereafter.

Algorithm 1: Main algorithm for VRPBB

```
1  initialize Pheromones, DualCosts; R* = ∅;
2  while ¬ stopping criterion do
3  |  foreach i ≤ M do
4  |  |  R* = R* ∪ CollectorAnt(DualCosts, Pheromones);
5  |  end
6  |  Sol = solveRelaxedRMP(R*);
7  |  if Sol ≠ ⊥ then
8  |  |  DualCosts = getDualCosts(Sol);
9  |  |  Pheromones = UpdatePheromones(Sol);
10 |  end
11 end
12 solve Integer Problem using MIP solver;
```

3.1 Reformulation as a set-covering problem

Every VRP can be seen as a set-covering problem, consisting in the selection of the optimal routes from the set of all possible (feasible) routes $\mathcal{R}$. Let c_r be the cost of route r, x_r a variable indicating whether route r is to be used in the solution and v_{ir} a constant indicating whether vertex i is visited in route r. The set covering problem analogous to the VRP is then defined as follows:

$$
\begin{aligned}
Min \quad & \sum_{r \in \mathcal{R}} c_r x_r \\
s.t. \quad & \sum_{i \in V \setminus 0} v_{ir} x_r = 1 \\
& \sum_{r \in \mathcal{R}} x_r \leq K \\
& x_r \in \{0,1\} \quad \forall r \in \mathcal{R}
\end{aligned}
$$

Note that this problem is defined over the set of all possible routes $\mathcal{R}$. Computing $\mathcal{R}$ is obviously intractable for all but the very smallest VRP problems. The principle of column generation is to generate columns by necessity. Only columns with potential to improve the objective function are generated and used to enrich the partial set $\mathcal{R}^*$. Identifying those columns corresponds to the Column generation subproblem. The Restricted Master Problem (RMP) then corresponds to the set-covering problem on the set of columns in $\mathcal{R}^*$.

3.2 Column generation subproblem

A classical approach to solving the column generation subproblem in a Vehicle Routing context is solving a Resource-constrained Shortest Path Problem (RCSPP). To do this the distances on the underlying problem graph are adapted using the dual costs from the relaxed RMP. Constraints such as the capacity constraint are modeled as a resource which is consumed along the produced path. Given the unknown character of constraint set C, modeling the latter as a resource is not an option. Verifying the feasibility of each partial path with respect to C is intractable due to the expensive feasibility check. Since C cannot be used to exclude partial paths, potentially a large number of infeasible paths would be generated, verifying the feasibility of all those paths is intractable as well. This is the reason we opted for a guided heuristic approach. Routes are generated and collected by individual probabilistic heuristic executions, called collector ants. Collector ants are focused on collecting routes of negative reduced cost. They do this while executing a solution construction heuristic, which helps them to ensure a good customer coverage in the collected routes. Note that collector ants follow pheromone trails but do not depose pheromones themselves. The pheromone deposit is ensured by an oracle and is based on the current relaxed solution. This allows the ants to build routes similar to those in the current relaxed solution, which are known to be feasible. The intuition behind this is that routes similar to feasible routes have a higher probability of being feasible as well. Also the ants should be able to rapidly gather a set of feasible routes, thus reducing the number of necessary feasibility checks.

3.3 Collector ants

Each collector ant executes a probabilistic version of the Clarke-Wright heuristic introduced in [2]. Their behavior is a variation of the one of the savings-based ants in [6].

Each collector ant starts with n routes where each of the n customers is visited in a route of its own. That is for every $i = 1, ..., n$ and its associated route r, $prev(i, r) = 0$ and $next(i, r) = 0$. Routes are then merged by and by until no further merges are possible. A merge of a route r_1 and a route r_2 corresponds to introducing an edge between customer $last(r_1)$ and $first(r_2)$ and removing the edges connecting those two customers to the depot. A merge $merge(r_1, r_2, ij)$ is thus defined by the two routes, r_1, r_2 to be merged, and the edge ij to be introduced. With each merge is associated a so-called savings value, defined as $s_{ij} = c_{i0} + c_{j0} - c_{ij}$. Each merge also has an associated attractiveness, depending on the savings value s_{ij} and the pheromone deposit τ_{ij} on the concerned edge.
Each ant disposes at each construction step of the list Ω_π of the π most attractive merges.

The merge from Ω_π to be executed is selected using roulette wheel selection.

The collector ants accept into Ω_π only merges that result in VRP-feasible routes. Those merges may or may not be C-feasible. The ants will only check routes of negative reduced cost for C-feasibility, since only those routes can contribute to further improve the relaxed RMP. This means that the ants may execute merges that are C-infeasible. The idea behind this is the fact that this may lead to future merges, resulting in C-feasible routes, that would otherwise not have been considered.

Finally, in order to maximize the utility of the expensive feasibility checks, each collector ant collects the set of (VRP- and C-) feasible routes that it encounters in its execution. This means that the ant not only collects feasible routes resulting from merges that it executed, but also those that it evaluated during the construction of Ω_π. These collected routes are used to enrich the set $\mathcal{R}^*$.

A clear advantage of the Clarke-Wright heuristic over other insertion-based heuristics in the case of the VRPBB is the number of feasibility checks that need to be executed in order to construct a route of size c, i.e. $O(c^2)$ checks in the case of an insertion-based technique, $O(c)$ checks in the case of the savings-based technique. Of course routes produced using the Clarke-Wright heuristic tend to be of lesser quality than those produced using an insertion-based technique. Also note that the Clarke-Wright heuristic is in fact a solution construction heuristic as opposed to a route construction heuristic. This allows to guarantee complete coverage of all customers in the produced routes.

3.4 Pheromone Update

Preliminary experiments have shown that the use of dual costs to guide the ants in their heuristic execution does not lead to routes of lower reduced cost. In fact, adapting the edges costs using the dual costs, as is commonly done, only led to an increased number of feasibility checks. This is due to the fact that the dual costs of the relaxed RMP do not take into account the unknown constraint set C and lead to routes that potentially allow to significantly decrease the value of the objective function, but are C-infeasible (which might be the reason they are not already in $\mathcal{R}^*$). The routes in the current relaxed solution are known to be C-feasible and at the same time contribute to the optimum of the relaxation. We follow two intuitions. First routes using a subset of edges from a C-feasible route, have a higher probability of being C-feasible themselves. Second routes in the neighborhood of routes contributing to the optimum have potential to help decrease the cost of the current relaxed solution. A somewhat similar intuition is used in the Relaxation-induced Neighborhood Search presented in [3] (although the latter focus on variables having the same value in both incumbent integer and relaxed solutions).
The proposed intensification is implemented using pheromones. The pheromone deposit on edges that appear in the relaxed solution is increased. This leads the collector ants to increasingly produce routes containing elements frequently encountered in the relaxed solution. At the same time, the pheromone deposit evaporates over time. This ensures that elements which were interesting in the past but did not appear in the last relaxed solutions lose attractiveness over time.

4 Conclusion

This paper introduces the novel generic problem of Vehicle Routing with Black Box Feasibility. In the VRPBB routes need to verify an unknown set of hard intra-route constraints. The feasibility of a route can only be verified using a computationally expensive black box algorithm. The difficulty of this problem stems mainly from the unknown problem structure, but also from the expensive feasibility check. To tackle the VRPBB we propose a column generation-based approach. Collector ants generate and collect columns with the potential to increase the quality of the current relaxed solution. Those ants are guided by pheromone deposits derived from the current relaxed solution. We are currently in the process of applying the described procedure to the 3L-CVRP, a challenging problem combining routing and loading. In future work the presented approach is to be included as the pricing step in a branch-and-price framework. The possibility of solving a Shortest Path Problem with Forbidden Paths, where the forbidden paths correspond to routes that are known to be C-infeasible, using a Constraint Programming approach will be explored as well.

References

1. Andreas Bortfeldt. A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. Technical report, FernUniversität in Hagen, 2010.
2. G. Clarke and J.V. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12, 1964.
3. Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102, 2005.
4. Guenther Fuellerer, Karl F. Doerner, Richard F. Hartl, and Manuel Iori. Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, 201, 2010.
5. Michel Gendreau, Manuel Iori, Gilbert Laporte, and Silvano Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40, 2006.
6. M.Reimann, M.Stummer, and K.F.Doerner. A savings based ant system for the vehicle routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2002.
7. Eric Prescott-Gagnon, Guy Desaulniers, Michael Drexl, and Louis-Martin Rousseau. European driver rules in vehicle routing with time windows. *Transportation Science*, 44, 2010.
8. Christos D. Tarantilis, Emmanouil E. Zachariadis, and Chris T. Kiranoudis. A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Transaction on Intelligent Transportation Systems*, 10, 2009.
9. P. Toth and D.Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, 2002.

Finding minimal siphons and traps as a Constraint satisfaction Problem

Faten Nabli
Faten.Nabli@inria.fr

EPI CONTRAINTES
INRIA Paris-Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105,
78153 LE CHESNAY CEDEX - FRANCE

Abstract. Bridging the gap between quantitative and qualitative models, Petri-nets have recently emerged as a promising tool for modeling biochemical networks. In this paper, we present a method to compute the minimal siphons and traps of a Petri-net as a resolution of a CSP. In our case, siphons and traps are purely structural properties that brings us information about the persistence of some molecular species. We present a program that finds minimal siphons and traps containing specific set of places in a Petri-net.

1 Introduction

During recent years, Systems Biology has become a rich field of study, trying to encompass the huge amount of heterogeneous information that becomes available thanks to the new high-throughput techniques of biologists, that requires the development of scalable analysis for detailed models of complex systems.

Some models have been growing bigger and bigger, filled with more and more mechanistic details, especially recently acquired post-transcriptional information, but lacking most of precise kinetic data. Unfortunately, very few analysis allow to extract information about the dynamics of these models, either because of their size or because of the imprecise kinetics. Other models remain of reasonable size, but have an even larger uncertainty about parameter values. For this other kind of model it is also important to be able to provide some dynamical analysis of the system's behavior.

The use of Petri-nets to represent biochemical reaction models, by mapping molecular species to places and reactions to transitions, was introduced quite late in [14], together with Petri-net concepts and tools new for the analysis of biochemical networks.

In this paper, we consider the Petri-net concepts of siphons and traps. These structures have already been considered for the analysis of metabolic networks in [19]. A siphon is a set of places that, once it is unmarked, remains so. A trap is a set of places that, once it is marked, can never loose all its tokens. Siphons can correspond to a set of metabolites that are gradually reduced during starvation

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 67–72,
Perugia, Italy, 12 September 2011.

whereas traps can correspond to accumulation of metabolites that are produced during the growth of an organism. In this article, after some preliminaries about Petri-nets and siphons and traps, we give our CSP model for enumerating minimal siphons and traps containing a given set (possibly empty) in a Petri-net.

2 Preliminaries

2.1 Petri Nets

A Petri-net graph PN is a weighted bipartite directed graph $PN = (P, T, W)$, where P is a finite set of vertices called places, T is a finite set of vertices (disjoint from P) called transitions and $W : ((P \times T) \cup (T \times P)) \to \mathbf{N}$ represents a set of directed arcs weighted by non-negative integers (the weight zero represents the absence of an arc). A marking of a Petri-net graph is a mapping $m : P \to \mathbf{N}$ which assigns a number of tokens to each place. A (marked) Petri-net is a 4-tuple (P, T, W, m_0) where (P, T, W) is a Petri net graph and m_0 is an initial marking.

The set of predecessors (resp. successors) of a transition $t \in T$ is the set of places ${}^\bullet t = \{p \in P \mid W(p,t) > 0\}$ (resp. $t^\bullet = \{p \in P \mid W(t,p) > 0\}$). Similarly, the set of predecessors (resp. successors) of a place $p \in P$ is the set of transitions ${}^\bullet p = \{t \in T \mid W(t,p) > 0\}$ (resp. $p^\bullet = \{t \in T \mid W(p,t) > 0\}$).

The classical Petri-net view of a reaction model is to associate biochemical *species* to *places* and biochemical *reactions* to *transitions*.

Example 1. The Michaelis-Menten enzymatic reaction system `A + E <=>A−E =>B +E` corresponds to the Petri-net depicted in Figure 1.

2.2 Siphons and Traps

Let $PN = (P, T, W)$ be a Petri-net graph.

Definition 1. *A trap is a non-empty set of places $P' \subseteq P$ whose successors are also predecessors: $P'^\bullet \subseteq {}^\bullet P'$.*

A siphon is a non-empty set of places $P' \subseteq P$ whose predecessors are also successors: ${}^\bullet P' \subseteq P'^\bullet$.

Example 2. In the Petri-net graph depicted in Figure 2, $\{A, B\}$ is a minimal siphon: ${}^\bullet\{A, B\} = \{r_1, r_2\} \subset \{A, B\}^\bullet = \{r_1, r_2, r_3\}$. $\{C, D\}$ is a minimal trap: $\{C, D\}^\bullet = \{r_4, r_5\} \subset {}^\bullet\{C, D\} = \{r_3, r_4, r_5\}$.

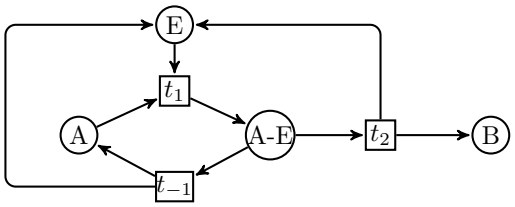


Fig. 1. Petri-net graph of Example 1

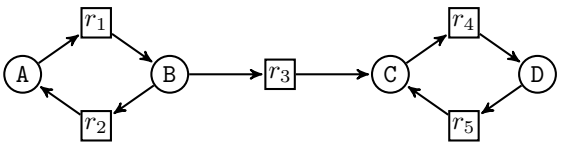


Fig. 2. Petri-net graph of Example 2.

,

2.3 Minimality

A trap (resp. siphon) is minimal if it does not contain any other trap (resp. siphon). One reason to consider minimal siphons is that they provide a sufficient condition for the non-existence of deadlocks. Indeed, it has been shown that in a deadlocked Petri-net (i.e. where no transition can fire) all unmarked places form a siphon [3]. Accordingly, the siphon-based approach for deadlocks detection checks if the net contains a proper siphon (a siphon is proper if its predecessors set is strictly included in its successors set) that can become unmarked by some firing sequence. A proper siphon does not become unmarked if it contains an initially marked trap. If such a siphon is identified, the initial marking is modified by the firing sequence and the check continues for the remaining siphons until a deadlock is identified, or until no further progress can be done. Considering only the set of minimal siphons is enough because if any siphon becomes unmarked during the analysis, then at least one of the minimal siphons must be unmarked.

Other links with behavioral properties of liveness are summarized in [8].

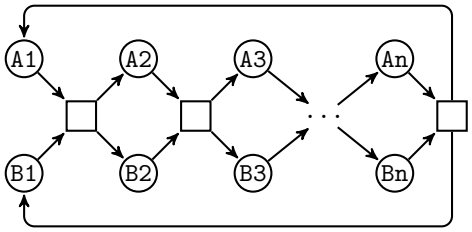
3 Complexity and Algorithms

3.1 Complexity

The problem of computing the set of minimal siphons of a given Petri-net is EXPSPACE since there can be an exponential number of such structures. The model of the following example has 2^n minimal siphons and 2^n minimal traps, each including either Ai or Bi for all i but not both of them.

Example 3.

A1 +B1 =>A2 +B2
A2 +B2 =>A3 +B3
...
An +Bn =>A1 +B1



Moreover, the decision problem of the existence of a minimal siphon containing a given place is NP-complete [17]. On the other hand, deciding whether a Petri-net contains a siphon or a trap and exhibiting one if it exists is polynomial [6].

3.2 Constraint Programming Algorithm

In the literature, many algorithms have been proposed to compute minimal siphons and traps of Petri-nets. Since a siphon in a Petri-net N is a trap of the dual net N', it is enough to focus on siphons, the traps are obtained by

duality. Some algorithms are based on inequalities [13], logic equations [12], or algebraic approaches [10]. More recent methods were presented in [17,18,6]. In this section we present a constraint program for solving this problem with a good practical efficiency. The search for siphons can be viewed as a Constraint Satisfaction Problem (CSP), in a similar manner to what has been done in mixed integer linear programming in [5] or in constraint logic programming for P- and T-invariants in [16]. For a Petri-net of n places and m transitions, a siphon S is a set of places whose predecessors are also successors. S can be represented with a vector $\boldsymbol{V}$ of $\{0,1\}^n$ such that for all $i \in \{1,2,..,n\}$, $V_i = 1$ if and only if $p_i \in S$.

It is quite natural to see this as a CSP on n Boolean variables. The siphon constraint can be formulated as $\forall i, V_i = 1 \Rightarrow (\forall t \in T, t \in {}^\bullet p_i \Rightarrow t \in (\cup_{V_j=1}\{p_j\})^\bullet)$. This constraint is equivalent to $\forall i, V_i = 1 \Rightarrow {}^\bullet p_i \subseteq (\bigcup_{V_j=1}\{p_j\})^\bullet$ which can be written again as $\forall i, V_i = 1 \Rightarrow \bigwedge_{t \in {}^\bullet p_i}(\bigvee_{p_j \in t^\bullet} V_j = 1)$. Under this latter form, the constraint is a Boolean constraint that can be directly processed in a constraint programming system.

To exclude the case of the empty set, we add the constraint $\bigvee_i V_i = 1$.

To ensure minimality, variable values are enumerated by trying sets of smallest cardinality first. A branch and bound procedure is wrapped around this enumeration, maintaining a set $\mathcal{M}$ of minimal siphons: after finding a new minimal siphon, the constraint that the next solution should not have its support bigger than any vector already present in $\mathcal{M}$ is added. In other words, a vector $\boldsymbol{V}$ is minimal in the set $\mathcal{M}$ if $\forall m \in \mathcal{M}, \exists i, V_i = 0 \wedge m_i = 1$.

In a post-processing phase, the set of minimal siphons can be filtered to only keep minimal siphons that contain a given set of places, in order to solve the above mentioned NP-complete problem.

These constraints and search strategy constitute a constraint satisfaction algorithm which has been implemented in GNU PROLOG [7], a Prolog compiler with constraint solving over finite domains facilities. The program can be used to either enumerate minimal siphons and traps containing a given set of places (possibly empty), check whether a set of places is a siphon or a trap.

4 Evaluation

Our CSP can be compared to the mixed integer linear model [5]. To find minimal siphons, the binary optimization problem on siphon constraint is iterated with minimal cardinality as objective function. At each iteration, a new constraint is added to exclude all siphons containing the already found siphons.

An experimental comparison on the biochemical models available in the Biomodels repository [1] , has shown that the CP enumeration with GNU PROLOG is at least 3 times faster than the MILP enumeration with CPLEX solver.

In this section, we evaluate the performance of our CSP model on some systems biology models.

[1] http://www.ebi.ac.uk/biomodels/models-main/tars/ (accessed January 2011)

The MAPK signal transduction cascade [11] is a well studied system that appears in lots of organisms and is very important for regulating cell division. In this example, 7 minimal siphons were computed in less than 5 ms. The proposed implementation appears to scale up quite well also on bigger models. In particular, on the largest interaction maps of the cell cycle control we have, the performance figures are as follows:

- Schoeberl's model of the MAP kinase cascade activated by surface and internalized EGF receptors [15] contains 94 places and 242 transitions. 13 minimal siphons and 15 minimal traps are computed in almost 30 ms [2].
- Calzone et al. E2F/Rb [1] has 404 places and 533 transitions. Its 70 minimal siphons and 246 minimal traps are both computed in around 284 ms.
- Khon's map [9,2] has 509 places and 775 transitions. Its 81 minimal siphons and 297 minimal traps are both computed in around 300 ms.

5 Conclusion

Siphons and traps define meaningful pools of compounds that display a specific behavior during the dynamical evolution of a biochemical system.

We have described a constraint satisfaction algorithm for computing siphons and traps and evaluated its scalability on the biochemical models available in the Biomodels repository.

The idea of applying constraint based methods to classical problems of the Petri-net community is not new, but seems currently mostly applied to the model-checking. We argue that structural problems can also benefit from the know-how developed for finite domain CP solving. The CSP model of minimal siphons and traps computation extends naturally to the previous model for P- and T-invariants search as a CSP [16].

In a parallel work, we have shown that siphons and traps entail a family of particular stability properties which can be characterized by a fragment of CTL [4] over infinite state structures. This fragment of Boolean CTL formulas can thus be verified efficiently thanks to these structural properties.

In a future work, we intend to find CP methods to make the model more efficient. We intend also to see how SAT solvers would perform on it and to consider other CP solvers.

References

1. L. Calzone, A. Gelay, A. Zinovyev, F. Radvanyi, and E. Barillot. A comprehensive imodular map of molecular interactions in RB/E2F pathway. *Molecular Systems Biology*, 4(173), 2008.
2. N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, and V. Schächter. Modeling and querying biochemical interaction networks. *Theoretical Computer Science*, 325(1):25–44, Sept. 2004.

[2] computation time on a PC with an intel processor 1.66 GHz and 3 GB of memory.

3. F. Chu and X.-L. Xie. Deadlock analysis of petri nets using siphons and mathematical programming. *IEEE Transactions on Robotics and Automation*, 13(6):793–804, 1997.
4. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
5. R. Cordone, L. Ferrarini, and L. Piroddi. Characterization of minimal and basis siphons with predicate logic and binary programming. In *Proceedings of IEEE International Symposium on Computer-Aided Control System Design*, pages 193–198, 2002.
6. R. Cordone, L. Ferrarini, and L. Piroddi. Some results on the computation of minimal siphons in petri nets. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, Hawaii USA, dec 2003.
7. D. Diaz and P. Codognet. Design and implementation of the GNU Prolog system. *Journal of Functional and Logic Programming*, 6, Oct. 2001.
8. M. Heiner, D. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *8th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Computational Systems Biology SFM'08*, volume 5016 of *Lecture Notes in Computer Science*, pages 215–264, Bertinoro, Italy, Feb. 2008. Springer-Verlag.
9. K. W. Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular Biology of the Cell*, 10(8):2703–2734, Aug. 1999.
10. K. Lautenbach. Linear algebraic calculation of deadlocks and traps. In G. Voss and Rozenberg, editors, *Concurrency and Nets Advances in Petri Nets*, pages 315–336, New York, 1987. Springer-Verlag.
11. A. Levchenko, J. Bruck, and P. W. Sternberg. Scaffold proteins may biphasically affect the levels of mitogen-activated protein kinase signaling and reduce its threshold properties. *PNAS*, 97(11):5818–5823, May 2000.
12. M. Minoux and K. Barkaoui. Deadlocks and traps in petri nets as horn-satisfiability solutions and some related polynomially solvable problems. *Discrete Applied Mathematics*, 29:195–210, 1990.
13. T. Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–579, Apr. 1989.
14. V. N. Reddy, M. L. Mavrovouniotis, and M. N. Liebman. Petri net representations in metabolic pathways. In L. Hunter, D. B. Searls, and J. W. Shavlik, editors, *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 328–336. AAAI Press, 1993.
15. B. Schoeberl, C. Eichler-Jonsson, E. Gilles, and G. Muller. Computational modeling of the dynamics of the map kinase cascade activated by surface and internalized egf receptors. *Nature Biotechnology*, 20(4):370–375, 2002.
16. S. Soliman. Finding minimal P/T-invariants as a CSP. In *Proceedings of the fourth Workshop on Constraint Based Methods for Bioinformatics WCB'08, associated to CPAIOR'08*, May 2008.
17. S. Tanimoto, M. Yamauchi, and T. Watanabe. Finding minimal siphons in general petri nets. *IEICE Trans. on Fundamentals in Electronics, Communications and Computer Science*, pages 1817–1824, 1996.
18. M. Yamauchi and T. Watanabe. Time complexity analysis of the minimal siphon extraction problem of petri nets. *EICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, pages 2558–2565, 1999.
19. I. Zevedei-Oancea and S. Schuster. Topological analysis of metabolic networks based on petri net theory. *In Silico Biology*, 3(29), 2003.

Adaptive Randomized Decompositions for Jobshop Scheduling

Dario Pacino (Student)[1] and Pascal Van Hentenryck[2]

[1] IT-University of Copenhagen (dpacino@itu.dk)
[2] Brown University (pvh@cs.brown.edu)

Abstract. Adaptive Randomized Decompositions (ARDs) have been instrumental in finding high-quality solutions to large-scale routing problems and flexible jobshop scheduling problems. This paper studies how ARDs behave on jobshop scheduling. Experimental results on benchmarks involving up to 2000 activities show that ARDs provide a simple and effective approach to produce high-quality results.

1 Introduction

Adaptive Randomized Decompositions (ARDs) have been proposed in [5, 6] to find high-quality solutions to large scale vehicle routing problems. It was shown to find high-quality solutions under time constraints and to improve many upper bounds to a wide variety of benchmarks when given more time. Recently, it was also shown to be effective on flexible shop problems, once again producing high-quality solutions quickly and finding new best solutions [11].

This paper aims at studying how ARDs behave on jobshop scheduling. It shows that simple ARDs produce solutions comparable in quality to the best constraint-based scheduling algorithms [4, 3] and outperform Large Neighborhood Search (LNS), when using the same search procedure, on many instance sets and, in particular, the hardest and largest instances. It significantly outperforms the model in [8], which was proposed as a simple and effective method.

The rest of the paper is organized as follows. Section 2 specifies the problem and discusses prior work. Section 3 presents the ARD algorithms. Section 4 presents the experimental results and Section 5 concludes the paper.

2 Problem Specification, Prior Work, and the CP Model

Specification A jobshop is specified by a set of jobs, each of which consists of a sequence of activities. Each activity must be executed on a specific machine, and has a fixed duration. No two activities can execute on the same machine at the same time and the goal is to minimize the makespan. Formally, each activity a is associated with a machine $m(a)$ on which it must execute for a duration $d(a)$. Every job defined by a sequence of activities $\langle a_1, ..., a_n \rangle$ generates a set of precedence constraints (a_{i-1}, a_i) for $i \geq 2$. We use $\mathcal{A}$ to denote the set of

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 73–78, Perugia, Italy, 12 September 2011.

activities, $\mathcal{P}$ the set of precedence constraints, and $\mathcal{M}$ the set of machines. The time horizon H for the schedule is given by $\left[0, \sum_{a \in \mathcal{A}} d(a)\right]$. A solution to the jobshop is an assignment σ, where $\sigma : \mathcal{A} \mapsto H$ assigns a starting date $\sigma(a)$ to each activity a. A solution is feasible if it satisfies the precedence and capacity constraints, i.e.,

$$\forall (a_i, a_j) \in \mathcal{P} : \ \sigma(a_j) \geq \sigma(a_i) + d(a_i)$$
$$\forall m \in \mathcal{M}, t \in H : |\mathcal{A}(\sigma, m, t)| \leq 1$$

where $\mathcal{A}(\sigma, m, t)$ is the set of activities associated with machine m at time t. An optimal solution is a feasible solution σ minimizing $\max_{a \in \mathcal{A}} \sigma(a) + d(a)$.

Prior Work Local Search is often the most efficient method for pure jobshop scheduling. Current state-of-the-art local searches, dedicated to the benchmarks used, are the TSAB [9] and i-TSAB tabu-search algorithms [10], solution-guided local search [1], and the TS/SA hybrid search [14]. The best results are currently given by a hybrid Constraint Programming/Local Search algorithm [13, 2] which alternatively runs a tabu search and a solution-guided constraint programming algorithm [3]. To our knowledge, the best constraint-based approach is SGMPCS [3] which uses a texture-based heuristic search [4], solution-guided search, and solution pools. Reference [8] argued that a very simple algorithm, not using sophisticated propagators, can be used to achieve comparable results.

3 Adaptive Randomized Decompositions

Adaptive Randomized Decomposition (ARD)'s aim is to find a sequence of decouplings, i.e., subproblems that can be independently optimized and whose solutions can be merged back into an existing solution to produce a better solution. Formally, given an instance P of a jobshop problem, the current solution π is used to find a decoupling (P_o, P_s) with projected solution π_o and π_s. The problem P_o is then re-optimized and its solution is merged into π_s to obtain a new solution for P. The ARD thus follows two simple principles:

1. Starting from an initial solution π_0 of P, it produces a sequence of solutions $\pi_1, ..., \pi_n$ such that the objective function $f(\pi_0) \geq f(\pi_1) \geq ... \geq f(\pi_n)$.
2. At step i, the solution π_{i-1} is used to obtain the decoupling (P_o, P_s) of P with solutions π_o and π_s. The problem P_o is then re-optimized and its solution π_o^* is used to obtain the new solution of $\pi_i = \text{MERGE}(\pi_o^*, \pi_{i-1})$.

Time Decomposition The time decomposition extracts a subproblem consisting of the activities that lie within a time window $\langle s, e \rangle$.

Definition 1. *A time decomposition* $\langle \mathcal{R}^d, \mathcal{P}^d, \alpha, \beta, \gamma, \phi \rangle$ *of a solution* σ_o *wrt* $\langle s, e \rangle$ *is a jobshop defined over the activities* $\mathcal{R}^d = \{a \in \mathcal{A} | \sigma_o(a) + d(a) > s \wedge \sigma_o(a) < e\}$, *with precedence constraints* $\mathcal{P}^d = \{(a, b) \in \mathcal{P} | a \in \mathcal{R}^d \wedge b \in \mathcal{R}^d\}$, *with availability constraints on the machines*

$$\gamma(m) = \min_{a \in \{a \in \mathcal{A} \setminus \mathcal{R}^d | \sigma_o(a) \geq e \wedge m(a) = m\}} \sigma_o(a)$$
$$\phi(m) = \max_{a \in \{a \in \mathcal{A} \setminus \mathcal{R}^d | \sigma_o(a) + d(a) \leq s\}} \sigma_o(a) + d(a)),$$

and with bounds on the activity starting times

$$\alpha(a) = \begin{cases} \sigma_o(b) + d(a) & \text{if } \exists (b,a) \in \mathcal{P} : \ b \notin \mathcal{R}^d \\ 0 & \text{otherwise;} \end{cases}$$

$$\beta(a) = \begin{cases} \sigma_o(b) & \text{if } \exists (a,b) \in \mathcal{P} : \ b \notin \mathcal{R}^d \\ \infty & \text{otherwise.} \end{cases}$$

A feasible solution to the time decomposition satisfies all traditional constraints of the jobshop, as well as the additional constraints:

$$\begin{aligned} &\forall a \in \mathcal{R}^d : \sigma(a) \geq \phi(m(a)) \\ &\forall a \in \mathcal{R}^d : \sigma(a) + d(a) \leq \gamma(m(a)) \\ &\forall a \in \mathcal{R}^d : \sigma(a) \geq \alpha(a) \\ &\forall a \in \mathcal{R}^d : \sigma(a) + d(a) \leq \beta(a). \end{aligned}$$

Since the problem is now decoupled, knowledge of how the re-optimized schedule will affect the overall solution is missing. It is thus more appropriate to use an objective function that maximizes the distance between each activity and their completion time bounds, allowing a better left shift of the entire schedule.

Definition 2 (Time Decomposition Objective). *The objective of a time decomposition* $\langle \mathcal{R}^d, \mathcal{P}^d, \alpha, \beta, \gamma, \phi \rangle$ *is defined by*

$$\textit{maximize} \min_{a \in \mathcal{R}^d} \min(\beta(a) - \sigma(a) + d(a), \gamma(m(a)) - \sigma(a) + d(a))$$

Machine Decomposition The idea behind the machine decomposition is to extract a subproblem by selecting activities on a subset of the machines $\mathcal{M}^d$.

Definition 3. *A machine decomposition* $\langle \mathcal{R}^d, \mathcal{P}^d, m^d, \alpha, \beta \rangle$ *of a solution* σ_o *wrt a set* $\mathcal{M}^d$ *of machines is a jobshop defined over the activities* $\mathcal{R}^d = \{a \in \mathcal{A} | m(a) \in \mathcal{M}^d\}$, *with precedence constraints* $\mathcal{P}^d = \{(a,b) \in \mathcal{P} | a \in \mathcal{R}^d \wedge b \in \mathcal{R}^d\}$, *with bounds on the activity starting times (* $\alpha(a)$ *and* $\beta(a)$*). A feasible solution to a machine decomposition satisfies all constraints of the jobshop, as well as the additional constraints:*

$$\forall a \in \mathcal{R}^d : (\sigma(a) \geq \alpha(a)) \ \wedge \ (\sigma(a) + d(a) \leq \beta(a)) \, .$$

Since the machine decomposition has full knowledge of the activities within each machine, minimizing the makespan guarantees the generation of non-degrading solutions as long as machines on the critical path are in the set $\mathcal{M}^d$.

Solution Merging Time and machine decompositions ensure that precedence and machine availability constraints are satisfied with respect to the original solution. It is thus possible to follow the machine precedences to merge the solutions.

Definition 4. *Let σ_d be a solution from the time decomposition $\langle \mathcal{R}^d, \mathcal{P}^d, \alpha, \beta, \gamma, \phi \rangle$ wrt σ_o and $\langle s, e \rangle$. The merging of σ_d and σ_o is the solution σ_m obtained by assigning a start date minimizing the set of precedence constraints:*

$$\{(a,b)|m(a) = m(b) \wedge a \notin \mathcal{R}^d \wedge b \in \mathcal{R}^d \wedge \sigma_d(b) \geq \sigma_o(a) + d(a)\} \cup$$
$$\{(a,b)|m(a) = m(b) \wedge a \in \mathcal{R}^d \wedge b \notin \mathcal{R}^d \wedge \sigma_o(b) \geq \sigma_d(a) + d(a)\} \cup$$
$$\{(a,b)|m(a) = m(b) \wedge a \in \mathcal{R}^d \wedge b \in \mathcal{R}^d \wedge \sigma_d(b) \geq \sigma_d(a) + d(a)\} \cup$$
$$\{(a,b)|m(a) = m(b) \wedge a \notin \mathcal{R}^d \wedge b \notin \mathcal{R}^d \wedge \sigma_o(b) \geq \sigma_o(a) + d(a)\}.$$

The merging is similar for the machine decomposition.

Elite Solution Pool Similarly to TSAB [9] and SGMPCS [3], also ARD uses the concept of elite solutions. A set of k solutions $\{\sigma_0, \sigma_1, ..., \sigma_k\}$ is initially generated and, at each iteration of the algorithm, a random solution is chosen to be improved. At the end of the search, the best solution σ^* is returned.

4 Experimental Evaluation

The proposed algorithm was evaluated on the standard jobshop scheduling benchmark problem instances of [12]. The algorithm was implemented on top of the COMET system and run on an Intel 2.8 GHz Xeon processor with 8Gb of RAM.

The algorithm of choice for the ARD is a LNS based on a simple Constraint Programming (CP) model. LNS, in this case, is used both to generate initial solutions, where the search is terminated after a few seconds, and to optimize the decoupled problems. This choice is due to preliminary experiments indicating that LNS, applied to those subproblems, outperforms a simple CP model.

The CP model states the precedence and disjunctive constraints. The disjunctive constraints are handled through edge-finder and not-first/not-last propagators. The search procedure is rather naive: It successively ranks the machines, starting with those with the least slack. The LNS uses temporal, machine and random neighborhoods in the same fashion as that of [7].

As in [3, 8], average results over 10 runs are reported and expressed in terms of Mean Relative Error (MRE) from the best-known upper bounds. The LNS and the decomposition algorithm ARD were compared to the most recent CP-based models, i.e., SGMPCS [3] and the model in [8] (hereby referred to as MM).

Table 1 presents the experimental results for runs of 60 minutes. It specifies the instance sets, the problem sizes, and the aggregated results of the MM, SGMPCS, LNS, and ARD algorithms, showing the MRE for both the average and best run. The bold fonts represents the best and *comparable* results, "comparable" being defined as at-most 0.005 MRE difference. Instances TA21-30 (20 $\times$ 20) and TA41-50 (30 $\times$ 20) are widely regarded as the most difficult JSP benchmark problems [14]. The results are quite interesting. First, ARD is extremely competitive with SGMPCS on the instance sets for which SGMPCS results are given. Second, ARD compares well and often outperforms LNS. In

Inst.	Size	MM		SGMPCS		LNS		ARD	
		Avg	Best	Avg	Best	Avg	Best	Avg	Best
TA11-20	20x15	0.0310	0.0204	**0.0125**	**0.0039**	0.0181	**0.0089**	**0.0144**	**0.0066**
TA21-30	20x20	0.0304	0.0231	**0.0147**	**0.0073**	**0.0187**	**0.0086**	**0.0168**	**0.0105**
TA31-40	30x15	0.0727	0.0602	**0.0150**	**0.0053**	0.0240	0.0143	**0.0164**	**0.0081**
TA41-50	30x20	0.0966	0.0844	-	-	0.0423	**0.0284**	**0.0343**	**0.0256**
TA51-60	50x15	-	-	-	-	**0.0004**	**0.0000**	0.0101	**0.0046**
TA61-70	50x20	-	-	-	-	**0.0129**	**0.0050**	0.0236	0.0126
TA71-80	100x20	-	-	-	-	0.0241	0.0100	**0.0057**	**0.0027**

Table 1. Comparison with State-of-The-Art CP Models in 60 Minute Runs.

inst.	LNS		LNS^t		ARD	
	Avg	Best	Avg	Best	Avg	Best
TA11-20	**0.022**	**0.010**	-	**0.013**	**0.020**	**0.010**
TA21-30	**0.025**	**0.012**	-	**0.014**	**0.024**	**0.014**
TA31-40	**0.031**	**0.018**	-	**0.015**	**0.027**	**0.017**
TA41-50	**0.053**	0.039	-	**0.031**	**0.058**	0.043
TA51-60	**0.012**	**0.005**	-	**0.000**	0.042	0.028
TA61-70	**0.047**	**0.034**	-	**0.034**	0.055	0.041
TA71-80	0.068	0.060	-	**0.020**	**0.037**	**0.023**

Table 2. Comparison with State-Of-The-Art CP Models in 10 Minutes Runs.

particular, this is the case on the hard TA41-50 instances and the largest TA41-50 instances. This is promising since it indicates that ARD has the potential to scale to very large-scale hard scheduling problems and obtain very high-quality solutions. Finally, MM is completely dominated by other approaches. So the simplicity advocated in [8] comes at a significant cost, especially given that it is not clear that MM scales to larger instances due to its memory requirements. This should be contrasted with the simplicity and performance of the approaches proposed herein.

Similar results are achieved within 10 minutes, as shown in Table 2. The algorithms being compared are our simple LNS, a large neighborhood search using texture-base heuristic (LNS^t) [7] (only the best results are given in the paper) and ARD. These results are surprisingly good, given that LNS and ARD uses a simple search procedure and require very little development effort on top of a modern constraint-based scheduler (e.g., LNS adds another 50 lines of code). ARD could certainly be improved by using texture-based heuristics [4] or some of the learning techniques used in [7]. Once again, the results seem to indicate that ARD will be a technique of choice to scale to very large-scale scheduling problems, as it already dominates LNS on the largest instances under time constraints when using the same search procedure.

5 Conclusion

This paper has shown that adaptive randomized decompositions (ARDs) are instrumental in finding very high-quality solutions to large-scale jobshop schedul-

ing problems, producing results comparable to the best existing algorithms and improving solution quality on hard or very large instances. ARDs thus seem to be a technique of choice to tackle very large-scale scheduling algorithms. This research also indicates that there is a strong need to generate new benchmarks for very large-scale scheduling problems. Possible future work is the application of ARD to other scheduling problems, such as cumulative scheduling and to problems with different objective functions.

References

1. Egon Balas and Alkis Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Manage. Sci.*, 44:262–275, February 1998.
2. J. C. Beck, T. K. Feng, and J.-P. Watson. Combining Constraint Programming and Local Search for Job-Shop Scheduling. *INFORMS Journal on Computing*, 23(1):1–14, May 2010.
3. J. Christopher Beck. Solution-guided multi-point constructive search for job shop scheduling. *Journal of Artificial Intelligence Research*, 29(1):49–77, May 2007.
4. J. Christopher Beck, Andrew J. Davenport, Edward M. Sitarski, and Mark S. Fox. Texture-based heuristics for scheduling revisited. In *Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence*, AAAI'97/IAAI'97, pages 241–248. AAAI Press, 1997.
5. Russell Bent and Pascal Van Hentenryck. Randomized adaptive spatial decoupling for large-scale vehicle routing with time windows. In *Proceedings of the 22nd national conference on Artificial intelligence*, 173–178. AAAI Press, 2007.
6. Russell Bent and Pascal Van Hentenryck. Spatial, Temporal, and Hybrid Decompositions for Large-Scale Vehicle Routing with Time Windows. *Principles and Practice of Constraint Programming (CP'2010)*, 99–113, 2010.
7. Tom Carchrae and J. Christopher Beck. Principles for the Design of Large Neighborhood Search. *Journal of Mathematical Modelling and Algorithms*, 8(3):245–270, 2009.
8. Diarmuid Grimes, Emmanuel Hebrard, and Arnaud Malapert. Closing the Open Shop: Contradicting Conventional Wisdom. In *Principles and Practice of Constraint Programming - CP 2009*, 400–408, 2009.
9. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 797–813, 1996.
10. E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.
11. Dario Pacino and Pascal Van Hentenryck. Large neighborhood search and adaptive randomized decompositions for flexiblem jobshop scheduling. In *Proceedings of the 22th International Joint Conference on Artifical Intelligence*, Barcelona, 2011.
12. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, January 1993.
13. Jean-Paul Watson and J. Christopher Beck. A Hybrid Constraint Programming / Local Search Approach to the Job-Shop Scheduling Problem. In CPAIOR-08, 263–277, 2008.
14. C. Zhang, P. Li, Y. Rao, and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, 35(1):282–294, January 2008.

Flexible timeline-based planning and its constraints

Student: Anna Roubíčková[1]
Supervised by: Rosella Gennari[1] and Marco Roveri[2]

[1] Free University of Bozen-Bolzano, 39100 Bolzano (BZ), Italy
[2] Fondazione Bruno Kessler, 38123 Povo (TN), Italy

Abstract. In our work we suggest a formalisation of the application-driven research of Cesta et al. [1] that is consistent with the traditional approach to temporal planning [3]. We revise the definitions of leading notions of flexible timeline-based planning under uncertainty and identify constraints stemming from such formalisation. As our research in this field is not finished yet, we also present few opened problems and questions to be addressed in the future.

1 Introduction

Our research is motivated by the work of Cesta et al., 2009 [1] on verification of timeline-based plans under uncertainty. However, we are more interested in the synthesis of such plans, which is a problem the original paper does not address.

The contribution of this paper is twofold: First, it unifies and formalises notions used in flexible timeline-based planning applications [1]; second, it shows how the proposed formalisation matches the more traditional models (e.g., based on chronicles) to allow plan synthesis by using already known algorithms. In order to properly define the flexible timeline-based framework, we modified some of the definitions of Ghallab, Nau an Traverso, 2004 [3] and even introduced some new definitions consistent with them.

2 Planning Model

The model of a planning system is built around the notion of *state variable*. The state variable describes legal evolutions of one property of the planning system (see Fig. 1 (a)), such as an activity of a space craft or a satellite's position on the orbit. Formally, it is a transition system:

Definition 1 (State Variable [1]). *A state variable is a tuple $x = \langle \mathcal{V}, \mathcal{T}, \mathcal{D} \rangle$, where $\mathcal{V}$ is a set of allowed values for x, that is, its domain; $\mathcal{T} : \mathcal{V} \to 2^{\mathcal{V}}$ is a value transition function defining legal successors for each value and $\mathcal{D} : \mathcal{V} \to N \times N$ is a value duration function providing a lower and upper bound of value's duration.*

A path in the transition system x describes an evolution of the corresponding property over time and is referred to as the *behaviour of* x. The behaviour can be also understood as a piecewise constant mapping from time to values of x (see Fig. 1 (b)).

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 79–84, Perugia, Italy, 12 September 2011.

Apart from state variables, the planning system is described using object constants and variables to refer to values of state variables (i.e., elements of $\mathcal{V}$) as well as temporal constants and variables to refer to timepoints. As time is considered to be continuous in this model, the temporal constants are real numbers and also the temporal variables range over $\mathbb{R}$. All these notions can be brought together to express propositions about the planning system by means of *temporal assertions*.

Definition 2 (Temporal Assertion [3]). *Temporal assertions are expressions of either of the following forms:*

a) $x@t : (v_1, v_2)$*, called an* event*, denotes a change of value of state variable* x *at timepoint* t *from* v_1 *to* v_2*,* $v_1 \neq v_2$*, (where a "change of value" refers to a transition taking place in the transition system* x *)*

b) $x@[t_1, t_2) : v$*, called a* persistence condition*, states that the value of variable* x *equals to* v *during the time period* $[t_1, t_2)$*,* $t_1 < t_2$*.*

Intuitively, the temporal assertions [3] describe the behaviour — an event defines a transition between two constant parts of the behaviour while a persistence condition defines a length of a constant part (see Fig. 1 (b)).

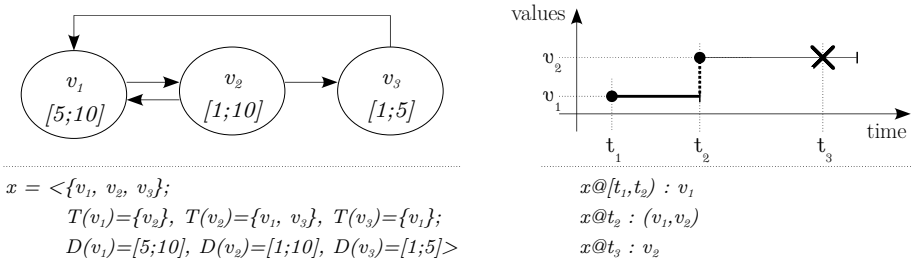


Fig. 1. (a) a state variable x; (b) a behaviour of x consistent with the temp. assertions

We believe that we one more kind of temporal assertion is needed to express initial conditions and even some kinds of goals, because the interval of validity, as stated by a persistence condition, may be unknown or unimportant while the preceding or following value, as needed for expressing an event, may be unknown or not applicable.

Definition 3 (Point Condition). *A point condition is a temporal assertion* $x@t : v$*, which states that the value of state variable* x *at timepoint* t *equals* v*.*

A point condition restricts the behaviour of x so that a point $[t; v]$ belongs to it.

A behaviour can be captured by a timeline. We suggest the following definition, which matches the traditional meaning of timeline [3] and also allows for extensions defined in Cesta et al., 2009 [1], such as completely specified or flexible timelines. These definitions are however meaningful only in the context

of bounded planning problems, which look for plans of length at most $H \in \mathbb{R}$. In such a case, the planning problem is considered only over finite temporal interval called a *planning horizon* $\mathcal{H} = [0; H)$.

Definition 4 (Timeline). *A* timeline [3] *for a state variable x is a pair $TL_x = \langle \mathcal{T}, \mathcal{C} \rangle$, where $\mathcal{T}$ is a set of temporal assertions about x and $\mathcal{C}$ is a consistent set of constraints*[3] *about values and times appearing in $\mathcal{T}$.*
A timeline is completely specified *if the union of all the persistence conditions in $\mathcal{T}$ covers the planning horizon $\mathcal{H}$.*

In case timeline is completely specified, we define $\mathcal{T}_e$ as a set of events that cause transitions between the consecutive values of x in correspondence to pairs of consecutive persistence conditions. A completely specified timeline for state variable x then becomes a pair $TL_x = \langle \mathcal{T}_e, \mathcal{C} \rangle$.

The notion of *flexible timeline* was introduced by Cesta et al., 2009 [1] to allow modelling uncertainty in timing of some activities. However, the notion of flexibility is only informally given in their work. To provide a formally sound definition of flexibility and of flexible timeline, we suggest to extend the definition of temporal assertions.

We defined a flexible version of the @ operator, denoted $@_f$, which replaces a crisp timepoint by a temporal interval. The flexible temporal assertion containing $@_f[t_s, t_e)$ are to be interpreted as an existential constraint, i.e., there exists a timepoint $t \in [t_s, t_e)$ such that the (crisp) temporal assertion featuring $@t$ takes place:

Definition 5 (Flexible Temporal Assertion). *Flexible temporal assertions are expressions of either of the following forms:*

a) $x@_f[t_1, t_2) : (v_1, v_2)$*, a* flexible event*, stands for* $\exists t \in [t_1, t_2)$ *st.* $x@t : (v_1, v_2)$*,*
b) $x@_f[t_{s_1}, t_{s_2}; t_{e_1}, t_{e_2}) : v$*, a* flexible persistence condition*, which stands for* $\exists t_s \in [t_{s_1}, t_{s_2}), t_e \in [t_{e_1}, t_{e_2})$ *st.* $x@[t_s, t_e) : v$*,* $t_s < t_e$*,*
c) $x@_f[t_1, t_2) : v$*, a* flexible point condition*, stands for* $\exists t \in [t_1, t_2)$ *st.* $x@t : v$*.*

As noted above, in case of completely specified timeline we do not need to keep track of the persistence conditions, the events that interleave them are sufficient to construct the whole timeline and so the persistence conditions are omitted.

This observation, together with the definition of flexible event, gives us an elegant way to define a flexible timeline in accordance to [1], where "transition events are associated to temporal intervals instead of to exact temporal occurrences":

Definition 6 (Flexible Timeline). *A flexible timeline for a state variable x is a pair $TL^*_x = \langle \mathcal{T}^*_e, \mathcal{C} \rangle$, where TL^*_x is completely specified, $\mathcal{T}^*_e$ is a set of events about x out of which at least one is flexible, and $\mathcal{C}$ is a consistent set of constraints about values and times of x.*

[3] Constraints are discussed separately in the following section.

The last notion to be mentioned here is the one of a *goal*. A goal can be composed of several sub-goals, which are specified as temporal assertions in the extended sense, including also point conditions and flexible assertions. Following the intended meaning of Cesta et al. [1] and the formalism of the textbook [3], we suggest following definition:

Definition 7 (Goal). *A* goal *is a pair* $\mathcal{G} = \langle \mathcal{T}_G^*, \mathcal{C}_G \rangle$*, where* $\mathcal{T}_G^*$ *is a set of possibly flexible temporal assertions* g *and* $\mathcal{C}_G$ *is a consistent set of constraints over variables from* $\mathcal{T}_G^*$*.*

Then, a *plan* [1] is a set of completely specified timelines $TL_x = \langle \mathcal{T}_e, \mathcal{C} \rangle$ consistent with $\mathcal{C}_G$, one for each state variable, such that $\forall g \in \mathcal{T}_G^* \;\; \exists TL_x$ such that $g \in \mathcal{T}_e$. A plan is called *flexible* if it contains at least one flexible timeline.

3 Constraints

In the previous section, we have mentioned a set of constraints as a part of definitions of timelines and goal. The notion of constraint comes from the textbook [3], but is not explicitly defined in the flexible timeline-based framework [1]. In this section, we study constraints implicitly included in their framework, the union of such constraints forms the previously mentioned set $\mathcal{C}$. Moreover, the constraints are in fact planning operators in our framework and hence are crucial for the plan synthesis as their consistent grounding results in a valid plan.

We will start with a review the constraints given by the traditional temporal planning [3] to compare them to constraints needed to handle the flexible timelines [1]. The constraints are over temporal and object variables (not over the state variables), and are of two corresponding types, temporal and binding. The *temporal constraints* impose qualitative restrictions on temporal variables and are expressed in point algebra [2]. The *binding constraints* range over the objects[4] of the planning domain and can express equality, inequality and belongingness to some class of objects. Formally,

Definition 8 (Constraint [3]). *A constraint is a binary relation of the form* $R(x_1, x_2)$ *that restricts allowed values of variables* x_1, x_2*. Depending on the type of the variables, we distinguish following two types of constraints:*

a) temporal: *where* x_1 *and* x_2 *are temporal variables and* R *is a temporal relation from point algebra, that is,* $R \in \{<, =, >\}$*.*
b) binding: *where* x_1 *is an object and* x_2 *is either an object with* $R \in \{=, \neq\}$ *or a class of objects, in which case* R *denotes a belongance* ($\in$)*.*

In the remainder of this section, we will study sources of constraints in the timeline-based framework of Cesta et al., 2009 [1] and formalize them using the formalisation provided in previous section, i.e., we will reformulate the constraints by means of temporal assertions. For simplicity, we will restrict the analysis to the crisp temporal assertions, noting that the flexibility only adds existential quantification over the temporally qualified formulae defining some of the constraints here.

[4] By object we understand either an object constant or an object variable.

The main source of constraints is the definition of a state variable. It restricts

- values the state variable is allowed to take,
- order of values taken by the state variable,
- durations of values taken by the state variable.

Allowed Values. Definition of state variable x contains a set of allowed values $\mathcal{V}$ for x. This results in a binding constraint over all temporal assertions x participates in:

$$\forall v \text{ such that } v \text{ appears in temporal assertion about } x \longrightarrow v \in V$$

Value Transition Function. Definition of state variable x contains a value transition function $\mathcal{T} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$ which defines binding constraints over pairs of consecutive values of variable x. The consecutive values of x are stated by events and by pairs of persistence conditions that follow each other:

$$\begin{gathered}\text{events: } \forall t, v_1, v_2 \text{ such that } x@t : (v_1, v_2) \longrightarrow v_2 \in \mathcal{T}(v_1) \\ \text{pers. cond.: } \forall t_1, t_2, t_3 \text{ such that } x@[t_1, t_2) : v_1 \wedge x@[t_2, t_3) : v_2 \longrightarrow v_2 \in \mathcal{T}(v_1)\end{gathered}$$

Value Duration Function. Definition of state variable x also contains a value duration function $\mathcal{D} : \mathcal{V} \rightarrow N \times N$ that sets the lower and upper bound of the length of the duration of value of x. Hence we can define metric [2] temporal constraints over the endpoints of intervals of persistence conditions:

$$\forall t_1, t_2 \text{ such that } x@[t_1, t_2) : v \longrightarrow (t_2 - t_1) \in \mathcal{D}(v)$$

The constraints induced by the value duration functions uses syntax not allowed by the textbook [3] as they are metric instead of qualitative. Therefore, the underlying structure to resolve the temporal constraints in our framework needs to be some kind of a Simple Temporal Network.

Synchronisation Rules. Note that the state variable definitions restrict only values belonging to the same state variable. To correlate values among different state variables, Cesta et al., 2009 [1] uses *synchronisation rules*:

Definition 9 (Synchronisation Rule [1]). *A* synchronisation *is an implication rule* $\langle TL_x, v\rangle \Rightarrow \{\langle TL_{x_i}, v_{i_j}, r_{i_j}\rangle; i, j \in I\}$ *that constraints the occurrence of reference value* v *on reference timeline* TL_x *with the occurrence of target values* v_{i_j} *on target timelines* TL_{x_i} *so that the timing of* v *and* v_{i_j} *relate through a metric temporal relation* r_{i_j}*.*

As we defined a timeline as a set of temporal assertions, we can rewrite the synchronisation rule to several constraints, depending on the temporal assertions used to define the synchronisation: The occurrence of a value v on a timeline TL_x means that there exists a temporal assertion (event, point condition or persistence condition) about x that assigns value v to x, meaning that:

$$\exists t \text{ such that } x@t : (v, v') \vee x@t : v \vee \{t \in [t_1, t_2) \wedge x@[t_1, t_2) : v\}$$

Using the above observation, we can rewrite the synchronisation rule as follows:

$$\forall t \text{ st. } x@t : (v, v') \vee x@t : v \vee \{t \in [t_1, t_2) \wedge x@[t_1, t_2) : v\} \Longrightarrow$$
$$\forall i, j \in I \; \exists t_{i_j} \text{ st.}$$
$$\{x_i@t_{i_j} : (v_{i_j}, v'_{i_j}) \vee x@t_{i_j} : v_{i_j} \vee \{t_{i_j} \in [t'_1, t'_2) \wedge x_i@[t'_1, t'_2) : v_{i_j}\}\} \wedge r_{i_j}(t, t_{i_j})$$

Note that the synchronisation rule behaves like a logical implication rather than a constraint, i.e., we quantify $\forall t, \forall i, j \in I : \; \exists t_{i_j}$ rather than $\exists t \; \exists t_{i_j}$.

Also note that the right-hand side of the implication is in fact a conjunction of existentially quantified expressions, i.e., there exist several timepoints t_{i_j} at which the target values v_{i_j} appear on corresponding target timelines TL_{x_i}.

4 Conclusion and Future Work

In our research we address the problem of synthesis of a temporal plan under uncertainty. Starting from the textbook of Automated Planning [3] and an application-driven research of Cesta et al. [1], we revised the definitions and proposed a formalisation of those.

Further we have studied and introduced a formal description of constraints present in the flexible timeline-based framework [1]. In the future we would like to compare this new set of constraints with constraints that are considered in the traditional temporal framework [3] to see whether the algorithms proposed for the traditional framework can be reused despite the flexibility and need for metric temporal information present in the flexible timeline-based framework.

Further, we would like to study the complexity issues of such algorithm, with a focus on the newly introduced temporal assertions and the fact that now the framework contains also disjunctions of constraints stemming from the synchronisation rules.

As an alternative approach for the plan synthesis we would like to modify the timed game automata approach which is used in [1] for the verification of the plan and compare the formal properties of such an approach to the traditional one [3], which is based on identifying and repairing flaws of a preliminary plan.

There also remains opened question about the representation of time, as one approach [1] considers it to be discrete and the other [3] continuous. Clearly such choice influences expressivity of the framework as well as its computational properties, so we would like to devote some part of future work to study of complexity under either of these options.

References

1. Cesta, A., Finzi, A., Fratini, S., Orlandini, A., Tronci, E.: Verifying flexible timeline-based plans. VVPS-09. Workshop on Verification and Validation of Planning and Scheduling Systems at ICAPS, Thessaloniki, Greece, September, 2009 (2009)
2. Gennari, R.: Temporal reasoning and constraint programming—a survey. CWI Quart 11, 163–214 (1998)
3. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA (2004)

Satisfiability Modulo Theory with Cost Optimization

Silvia Tomasi
silvia.tomasi@disi.unitn.it
Advisor: Prof. Roberto Sebastiani
roberto.sebastiani@disi.unitn.it

DISI, University of Trento
Via Sommarive, 14, I-38123 Povo, Trento, Italy

Abstract. Many practical real-world decision problems can be formalized as Boolean combinations of Boolean propositions and linear expressions over rational or integer variables (possibly combined with other theories). These problems can be encoded into Satisfiability Modulo Theory (SMT) and the modern SMT procedures can be used to find a solution to them. A quite new research area is Optimization Modulo Theories (OMT), i.e. a generalization of SMT that aims at finding an assignment which *optimizes* a given cost function. OMT captures and generalizes many optimization problems, including Pseudo-Boolean Optimization and Linear Generalized Disjunctive Programming. To the best of our knowledge, there exist very few and limited works addressing OMT and they all focus on pseudo-Boolean cost functions defined over predicates. In this paper we extend the classic SMT framework with linear cost functions over domain variables and propose two procedures for addressing optimization problems.

1 Motivations and goals

Many practical real-world decision problems (e.g. formal verification of RTL designs [5], or Real-Time Systems [1] and software pieces [4], resource planning [21]) can be encoded into Satisfiability Modulo Theory (SMT) formulas (or $\mathcal{T}$-formulas) which are Boolean combinations of atomic propositions and ground atomic formulas in a given first-order theory $\mathcal{T}$; $\mathcal{T}$ can be a combination of theories such that $\mathcal{T} = \mathcal{LA} \cup \bigcup_{i=0}^{n} \mathcal{T}_i$, for the linear arithmetic theory $\mathcal{LA}$ (either on the rationals, $\mathcal{LA}(\mathbb{Q})$, or on the integers, $\mathcal{LA}(\mathbb{Z})$) and some theory $\mathcal{T}_i$ (such as the theory of equality and uninterpreted functions or the theory of the arrays). These problems can be effectively solved exploiting modern SMT procedures (see, e.g., [3]).

However, in some applications (e.g. radio link frequency assignment [6], displacement of tools [11], resource planning [12] and job scheduling [16]) we are interested not only in finding a solution which satisfies the constraints, but also in finding a solution which also *optimizes* the usage of certain resources. Although optimization problems have been deeply addressed in different contexts, such as Satisfiability (SAT) and Mathematical Programming, there exist limitations on the constraints set, the cost function and the background theories that are used for expressing a problem (see section 5). A quite new research area is Optimization Modulo Theories (OMT). OMT is a generalization of SMT that aims at finding an assignment which optimizes a given cost function.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 85–90,
Perugia, Italy, 12 September 2011.

To the best of our knowledge only two works [15, 7] deal with OMT, but they focus on cost functions expressed as Boolean predicates.

Our work aims at extending the classic SMT framework with linear cost functions over *domain* (i.e. arithmetical) variables rather than predicates. We focus on the OMT with Linear Cost Function (LOMT($\mathcal{T}$)) problem that aims at finding a satisfying assignment that minimizes a linear cost function. We address LOMT as a sequence of SMT decision problems by implementing two classical approaches: branch and bound and binary search. LOMT captures and generalizes many optimization problems, including Pseudo-Boolean Optimization and Linear Generalized Disjunctive Programming.

This paper is structured as follows. Some background is presented in section 2. In sections 3 and 4 we respectively describe the problem statement and the main guidelines for addressing SMT optimization procedures. In section 5 we spotlight the differences between different optimization problems in the contexts of SAT, SMT and Mathematical Programming. In section 6 we describe ongoing work. Finally, in section 7 we draw some conclusions and outline directions for future research.

2 Background

Satisfiability Modulo Theory ($SMT(\mathcal{T})$) is the problem of deciding the satisfiability of ground formulas wrt. a background theory $\mathcal{T}$ (notice that $\mathcal{T}$ can be also defined as a combination of theories, i.e. $\mathcal{T} = \bigcup_i \mathcal{T}_i$).

A SMT solver is a procedure able to decide $SMT(\mathcal{T})$. A *theory solver* for $\mathcal{T}$, or $\mathcal{T}$*-solver*, is a procedure able to decide the satisfiability in $\mathcal{T}$ of sets/conjunctions of of atomic formulas and their negations ($\mathcal{T}$-literals). If $\mathcal{T}$*-solver* is invoked on a $\mathcal{T}$-inconsistent set μ of $\mathcal{T}$-literals, it returns unsat and the sub-assignment $\eta \subseteq \mu$ which was found inconsistent in $\mathcal{T}$ (hereafter η and $\neg\eta$ are respectively called $\mathcal{T}$*-conflict* and $\mathcal{T}$*-conflict clause*). Otherwise, $\mathcal{T}$*-solver* returns sat and a $\mathcal{T}$-model $\mathcal{I}$ that satisfies μ (i.e. $\mathcal{I} \models \mu$). This technique is called $\mathcal{T}$*-deduction* and $\neg\eta \vee l$ is called $\mathcal{T}$*-deduction clause*.

The standard *lazy* approach for solving $SMT(\mathcal{T})$ is to use a conflict-driven clause-learning (CDCL) SAT solver for enumerating truth assignments μ for the Boolean abstraction of the input formula φ (that maps Boolean variables into themselves and theory atoms into fresh Boolean variables and distributes with sets and Boolean connectives) and then invoke one (or more) $\mathcal{T}$*-solver* (s) for checking the consistency in $\mathcal{T}$ of the set of $\mathcal{T}$-literals corresponding to μ. If the $\mathcal{T}$*-solver* returns sat, then φ is $\mathcal{T}$-consistent, otherwise the $\mathcal{T}$-solver returns the $\mathcal{T}$-conflict η. The conflict clause $\neg\eta$ is added to φ ($\mathcal{T}$*-learning*) and then used to backtrack ($\mathcal{T}$*-backjumping*). The integration of modern SAT procedures and $\mathcal{T}$*-solvers* is enhanced by very effective techniques such as *early pruning*. In early pruning the $\mathcal{T}$*-solver* is invoked also on intermediate assignments μ. If μ is not $\mathcal{T}$-consistent then all possible extensions of μ are not $\mathcal{T}$-consistent and the procedure can backtrack pruning lots of search space. For more details we refer the reader to [3].

3 Optimization Modulo Theory with Linear Cost Function

We extend the classic SMT framework by adding linear cost functions over domain variables in order to deal with optimization problems. We focus on problems that can be formally defined as a tuple $\langle \varphi, cost, \mathsf{lb}, \mathsf{ub} \rangle$ such that:

- φ is a SMT formula with respect to some background theory $\mathcal{T} = \mathcal{LA} \cup \mathcal{T}_i$ for $i = 0, \ldots, n$ where $\mathcal{LA}$ can be on either $\mathcal{LA}(\mathbb{Q})$ or $\mathcal{LA}(\mathbb{Z})$. The variables contained into $\mathcal{LA}$-atoms belong to the domain D;
- $cost$ is a linear function of the form $cost = \sum_{i=1}^{N} a_i x_i$ where a_i are constant numbers and x_i are variables belonging to D;
- lb and ub are constant numbers (possibly $-\infty$ and $+\infty$ respectively) such that $\mathsf{lb} \leq cost < \mathsf{ub}$ and are called respectively *lower bound* and *upper bound* (and the interval $[\mathsf{lb}, \ldots, \mathsf{ub}[$ is called *range of the cost function*).

Rather than searching for a satisfying assignment, as in SMT, OMT with Linear Cost Function (LOMT) consists in finding an assignment satisfying φ whose value of $cost$ is minimum and within the range of the cost function.

4 Addressing the LOMT problem

We address the LOMT problem by extending a state-of-the-art CDCL-based SMT solver in order to implement two procedures based on the following approaches: branch and bound and binary search. These approaches can exploit CDCL techniques (such as $\mathcal{T}$-learning and $\mathcal{T}$-backjumping) in order to reduce the space of the possible solutions by cutting non-optimal solutions off and avoiding redundant search. In the following we briefly describe such procedures.

4.1 Branch-and-Bound Approach

We propose first a very naive branch-and-bound procedure for solving LOMT($\mathcal{T}$). First, it adds the cost function to the input formula and learns bound constraints in order to restrict the search within the range $[\mathsf{lb}, \mathsf{ub}[$ resulting in the formula $\varphi_C = \varphi \wedge (cost = \sum_{i=1} a_i x_i) \wedge \neg(cost < \mathsf{lb}) \wedge (cost < \mathsf{ub})$. Then the procedure iteratively explores the solutions space by invoking an incremental SMT solver[1]. If the SMT solver proves the satisfiability of φ_C, then it returns sat and a satisfiable truth assignment μ for φ_C. The procedure invokes $minimize()$[2] that minimizes the cost function $cost$ subject to μ and returns the current optimal value $\mathsf{mincost}$. The constraint $(cost < \mathsf{mincost})$ is added to φ_C so that the current solution becomes the new upper bound for the cost function. This process is performed until either φ_C is found unsatisfiable or the problem is unbounded (i.e. the optimal solution found so far is equal to $-\infty$). In the former case the last

[1] A SMT solver is incremental if it allows to prune the search by reusing clauses (both Boolean and theory ones) learned from previous calls.

[2] We implemented the procedure $minimize()$ by extended a *$\mathcal{LA}$-solver* proposed by Dutertre and de Moura [8] to support (a variant of) the Simplex method.

solution found is the optimal one. Notice that the procedure works also if $\mathsf{lb} = -\infty$, or $\mathsf{ub} = +\infty$, or both.

A much more sophisticated version is implemented by embedding the minimization loop inside the Boolean search performed by the CDCL-based SMT solver. When a satisfying assignment μ is found $\mathcal{T}$-consistent by the $\mathcal{T}$-solver, μ is also fed to $minimize()$; the latter returns the minimum cost of μ, which becomes the new value of mincost, and the subset η of $\mathcal{LA}$-atoms in μ involved in the minimum. Then the clauses $(cost < \mathsf{mincost})$ and $\neg\eta$ are learned and used for driving the backjumping mechanism. Notice that every partial assignment μ contains the atom $(cost < \mathsf{mincost})$. Thus, if a partial assignment μ is generated which cannot be expanded into a total one whose cost is better than mincost, then μ is found $\mathcal{T}$-inconsistent directly by early-pruning calls to $\mathcal{T}$-solver. To this extent, in this schema early-pruning plays the role of "bounding" in branch and bound.

4.2 Binary-Search Approach

The second procedure we present is based on the binary-search schema. A very naive version starts by adding cost function and bound constraints to the input formula resulting in the formula φ_C. The procedure restricts the search within the interval $[\mathsf{lb}, \mathsf{ub}[$ and divides it as a function of the upper and lower bounds by computing the value $\mathsf{pivot} \stackrel{\text{def}}{=} (1-t)\cdot\mathsf{lb} + t\cdot\mathsf{ub}$ (where t is a parameter in $]0,1]$). At each step, the incremental SMT solver checks the satisfiability of the formula $\varphi_C \wedge (cost < \mathsf{pivot})$. Notice that the constraint $(cost < \mathsf{pivot})$ restricts the search in the interval $[\mathsf{lb}, \mathsf{pivot}[$. If the SMT solver finds a satisfiable truth assignment μ for φ_C, the current optimal value mincost is found by invoking the procedure $minimize()$. The clause $(cost < \mathsf{mincost})$ is learned pruning all the solutions greater than mincost so that the search goes on in the interval $[\mathsf{lb}, \mathsf{mincost}[$. If φ_C is unsatisfiable, the search proceeds in the interval $[\mathsf{pivot}, \mathsf{ub}[$. The process terminates when either the search interval is empty (in this case the last solution found is the optimal one) or the problem is unbounded. Notice that, if t is 1, the procedure reduces to a variant of the branch-and-bound schema.

As with the branch-and-bound case, a much more sophisticated version can be implemented by embedding the minimization loop inside the Boolean search of a CDCL-based SMT solver. Every time the search of the CDCL solver is at decision level zero, the atom $(cost < \mathsf{pivot})$ is assumed at level one (where pivot is defined as above), ub [resp. lb] being the lowest [resp. highest] value ub_i [resp. lb_i] such that the atom $(cost < \mathsf{ub}_i)$ [resp. $\neg(cost < \mathsf{ub}_i)$] is currently assigned at level 0. The procedure terminates when $\mathsf{ub} < \mathsf{lb}$. During the search, when a satisfying assignment μ is found $\mathcal{T}$-consistent by the $\mathcal{T}$-solver, μ is also fed to $minimize()$; the latter returns the minimum cost of μ, which becomes the new value of mincost, and the subset η of $\mathcal{LA}$-atoms in μ involved in the minimum. Then the clauses $(cost < \mathsf{mincost})$ and $\neg\eta$ are learned and used for driving the $\mathcal{T}$-backjumping mechanism. Also the atom $(cost < \mathsf{pivot})$ is learned in order to activate the clauses in the form $\neg(cost < \mathsf{pivot}) \vee \ldots$ which have been learned during previous steps in the search.

Since the above schema can be applied only if the cost function is bounded, we propose a procedure that tries to find the bounds when they are infinite. If it fails, the branch-and-bound search is invoked instead.

5 Related Work

With respect to existing works related to optimization in SAT [13, 17] and SMT [15, 7], we express cost functions as linear functions over domain variables rather than as purely pseudo-Boolean functions.

In the context of Mathematical Programming we can consider the following optimization paradigms: *Mixed-Integer Linear Programming* (MILP), *Disjunctive Programming* (DP) and *Linear Generalized Disjunctive Programming* (LGDP). All of them have some limitations concerning the set of constraints and the background theories that are used for expressing a problem. In particular, MILP aims at optimizing a linear function subject to a set of linear constraints over rationals and integer variables [14]. Notice that MILP is a subcase of LOMT($\mathcal{T}$) where $\mathcal{T} = \mathcal{LA}(\mathbb{Q}) \cup \mathcal{LA}(\mathbb{Z})$ and the formula φ is a conjunction of literals. DP and LGDP are close in spirit to LOMT($\mathcal{T}$). The former focuses on optimizing a linear function subject to Boolean combination of linear constraints [2] and the latter extends DP by involving Boolean variables and logic propositions [18]. Unlike DP and LGDP, we do not impose any restriction on the background theory and we can also handle combinations of theories, such as *Equality and Uninterpreted Functions*, the theories of *bit-vectors* and *arrays*.

Mathematical Programming paradigms and LOMT also differ in the solution approach. While the first ones use arithmetical approaches that integrate branch-and-bound search with cutting plane methods [14, 2], we adopt a logic approach where the Boolean search guides the analysis of the solutions space as described in section 4.

6 Ongoing Work

We have implemented the "more sophisticated" versions of the optimization procedures described in section 4 and we are currently testing them on benchmarks in the SMT-LIB (augmented with randomly generated linear cost functions) and LGDP problems taken from [19]. We are also comparing our optimization procedures against two solvers, LogMIP [20] and EMP [9].

7 Conclusions and Future Work

Our work aims at addressing the problem of Optimization Modulo Theories with linear cost functions (LOMT) that involves finding a satisfying assignment for a SMT formula that minimizes a given a linear cost function.

In the future, we expect to develop efficient implementations of the two proposed procedures and compare them against other techniques on restricted problems (e.g. Disjunctive Programming and Linear Generalized Disjunctive Programming). Notice that efficient implementations will benefit from effective techniques for integrating theory and Boolean search and combining branch and bound with binary search and smarter heuristics (e.g. adaptive mechanisms and variable selection heuristics). We also plan to extend LOMT($\mathcal{T}$) to linear arithmetic on integers $\mathcal{LA}(\mathbb{Z})$ and convex non-linear cost function, if feasible. Finally, we would like to investigate the possibility of using Stochastic Local Search for solving LOMT($\mathcal{T}$) by continuing and extending the work we did in [10].

References

1. G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In *FORTE*, pages 243–259, 2002.
2. E. Balas. Disjunctive programming. In M. Junger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 283–340. Springer Berlin Heidelberg, 2010.
3. C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, pages 825–885. 2009.
4. D. Beyer, A. Cimatti, A. Griggio, M. E. Keremoglu, and R. Sebastiani. Software model checking via large-block encoding. In *FMCAD*, pages 25–32, 2009.
5. M. Bozzano, R. Bruttomesso, A. Cimatti, A. Franzen, Z. Hanna, Z. Khasidashvili, A. Palti, and R. Sebastiani. Encoding RTL Constructs for MathSAT: a Preliminary Report. *Electron. Notes Theor. Comput. Sci.*, 144:3–14, January 2006.
6. B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999.
7. A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico. Satisfiability modulo the theory of costs: Foundations and applications. In *TACAS*, pages 99–113, 2010.
8. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV*, volume 4144 of *LNCS*, 2006.
9. M. C. Ferris, S. P. Dirkse, J.-H. Jagla, and A. Meeraus. An Extended Mathematical Programming Framework , 2009.
10. A. Griggio, Q. S. Phan, R. Sebastiani, and S. Tomasi. Stochastic Local Search for SMT: Combining Theory Solvers with WalkSAT. In *FroCoS*, 2011.
11. M. Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & OR*, 25(11):925–940, 1998.
12. J. R. Jackson and I. E. Grossmann. High level optimization model for the retrofit planning of process networks. *Ind. Eng. Chem. Res*, 41:41–3762, 2002.
13. C. M. Li and F. Manyà. *MaxSAT, Hard and Soft Constraints*, volume 185, chapter 19, pages 613–631. IOS Press, 2009.
14. A. Lodi. Mixed Integer Programming Computation. In M. Junger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 619–645. Springer-Verlag, 2009.
15. R. Nieuwenhuis and A. Oliveras. On SAT Modulo Theories and Optimization Problems. In A. Biere and C. P. Gomes, editors, *9th International Conference on Theory and Applications of Satisfiability Testing, SAT'06*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2006.
16. R. R. and G. I.E. Modeling and computational techniques for logic based integer programming. *Computers and Chemical Engineering*, 18 (7):563–578, 1994.
17. O. Roussel and V. M. Manquinho. Pseudo-boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185, chapter 22, pages 695–733. IOS Press, 2009.
18. N. Sawaya and I. Grossmann. Reformulations, relaxations and cutting planes for linear generalized disjunctive programming.
19. N. W. Sawaya and I. E. Grossmann. A cutting plane method for solving linear generalized disjunctive programming problems. *Computers and Chemical Engineering*, 29:1891–1913, 2005.
20. A. Vecchietti and I. E. Grossmann. Computational experience with logmip solving linear and nonlinear disjunctive programming problems.
21. S. A. Wolfman and D. S. Weld. The lpsat engine & its application to resource planning. In *IJCAI*, pages 310–317, 1999.

Genetic Based Automatic Configurator for Minion

Hu Xu, Karen Petrie, and Keith Edwards

Computing School,
QMB 1.10, University of Dundee, Dundee, UK
{huxu,karenpetrie,kedwards}@computing.dundee.ac.uk

Abstract. The efficient choice of a preprocessing level can reduce the searching time of a constraint solver to find a solution to a constraint program. Currently the parameters in constraint solver are often picked by hand by experts in the field. Genetic algorithms are a robust technology for problem optimization, such as function optimization. In this paper Genetic algorithms are applied to create an automatic configuration mechanism for Minion[2], which is a popular general purpose constraint solver. The experiments in the paper are a proof of concept for the idea of combining Genetic algorithm with constraint programming to aid in the parameter selection problem.

Keywords: Genetic Algorithm, Constraint Programming, Minion, Automatic Configurator, Parameter Tuning

1 Introduction

Problems often consist of choices. Making an optimal choice which is compatible with all other choices made is difficult. Constraint programming (CP) is the branch of Artificial Intelligence, where computers help us to make these choices.Constraint programming applies constraints to trim and reduce the search space. According to the requirements of the problem, it can find just one or all the solutions.

In general a constraint solver is responsible for finding the solution(s) following the modeling of constraint satisfaction problem. Minion is a free, open source and general-purpose constraint solver, with an expressive input language based on the common constraint modelling device of matrix models. The selection of suitable preprocessing levels for a given constraint problem is an important part of constraint programming. Efficiently tuning a constraint solver will shorten the search time and reduce the running cost. One of the keys to increase the search speed for a constraint solver is tuning the solvers parameters. Currently the job of tuning parameters is done by hand. It's unfair for researcher using algorithm with perfect setting compare the efficiency with their competitors. The learning curve involved can be a barrier to a novice user in learning how to efficiently use a CP solver.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, pages 91–96,
Perugia, Italy, 12 September 2011.

Genetic algorithm is a classic global optimization method, which proposed by John Holland and colleague [3]. Genetic algorithms are usually implemented in a computer simulation in which a population of abstract representations of candidate solutions to an optimization problem evolves toward better solutions. Genetic algorithms can find an optimal solution very quickly in optimization problems often fater than the traditional CP methods. However genetic algorithms are tend to find good solutions rather than best solution. Therefore the idea of combining the speed genetic algorithms and the expressive power of constraint programming seem worth exploring. Successful attempts have been made in this area before. In 2002 Kanoh [8] proposed a hybrid search method that combines the genetic algorithm with min-conflicts hill-climbing in solving constraint satisfaction problems. Jorge [9] also applied genetic algorithms to solve a Constraint Satisfaction Problem- Puzzle Eternity II in 2009.

However the general framework of combining genetic algorithms and constraint programming to any problems, has not been achieved. In my research project I wish to combine the expressive power of constraint programming with the speed of genetic algorithms and explore deeply the efficiency of the hybrid strategy in various problems.

In this paper genetic algorithms are chosen to optimize the tuning of Minion. There are two main reasons to choose genetic algorithms to optimize parameter tuning. One is that genetic algorithms have a powerful ability to tackle optimization problems which have a lack of auxiliary information [1]. Another is that genetic algorithms do parallel search rather than linear search. Each chromosome races against another in each generation. Therefore the idea of creating automatic configurations with genetic algorithms catches lots of researcher's attention [4].The automatic configuration will reduce time consuming of solver user on parameter tuning rather than done by hand.

Ansótegui [1] has posed a gender-based genetic algorithm for the automatic configuration of algorithms. He uses a variable tree (AND/OR Search Trees) to divide variables into several parts which can be optimized independently. Automatic tuning will lead to improvements over manual tuning by researchers themselves. ParamILS and CALIBRA[6] have also shown the efficiency and possibility of automatic configuration for a constraints solver. In this paper genetic algorithms will help Minion to pick the correct switches in three different classical constraint problems.

2 Method

We have created a Genetic Based Automatic Configurator for Minion, named (GACM). *GACM* suggests a setting for the Minion parameters for a given CP problem, the problem is then run through Minion using this setting. The statistics as to how efficiently MInion solved the problem using these parameters are then fed back to the genetic algorithm to be used to guide the next iteration.

Genetic Algorithm

The first step of a genetic algorithm (GA) is called the encoding which is to construct the suitable chromosome for the optimization problem. In our automatic genetic configurator, the tuning of four switches is considered. The chromosomes length depends on the number of those switches' values. For example varorder has nine statuses, four binary bits are required to present it.

Fitness describes the ability of an individual to reproduce in biology. In this project, we focus on the minimum running time to find the solution of optimization problem with specific switches. GA used to find the maximum value. Therefore the fitness in this project is $Fitness(x) = 1/x$, where x is the running time of finding the solution with relative switches.

The Selection in genetic algorithm is a strategy which allows the perfect parents (with high fitness) to have more of a chance to be selected to generate the next generation. In our genetic configurator, the selection is the roulette wheel selection. Roulette wheel selection is a way of choosing individuals from the population of chromosomes in a way that is proportional to their fitness. The following is the selection probability formula for each chromosome:

$$P(C_i) = \frac{f_i}{\sum_{i=1}^{n} f_i}$$

Where $P(C_i)$ is the selection probability of each chromosome, C_i is the ith chromosome in population, f_i is the fitness of chromosome C_i and n is the population number.

Crossover can improve the whole population fitness quickly by mating parents to produce an offspring. It is a very important operator in genetic algorithms.Mutations which change one or more genes in an individual is another operator used in GA. Mutation can help genetic algorithm escape the local maximum state by creating a new gene string. This paper only choose one point crossover and on point mutation in [3].

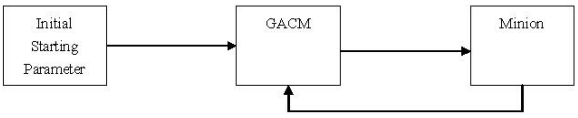


Fig. 1. The Framework of Genetic Algorithms Configurator for Minion

GACM Interface Design

The communication interface is a data bridge between the automatic configurator and Minion. The interface will send the suggested parameters to Minion

and get the running cost of optimizing the problem back to our configurator to evaluate the evolutionary result. Figure 1 shows that some set of switches will be randomly generated and sent to Minion. Minion will gain the running cost back of each set of switches and feed back them to genetic based automatic configurator.

Experiment Design

To prove the efficiency of our genetic configurator, the optimized tuning results of GACM will be compare with the random selection. In this paper we consider the, N-queen problem, the open stack problem and the Landford's number problem. These three classical constraint problems were chosen as optimization problem in Minion. We hope genetic based automatic configurator could be applicable to different constraint satisfaction problem. Meanwhile for each optimization problems we tested two different instances,for example the n was both 26 and 28 in N-queens. In Goldberg's book [3], it said that the parameter setting of genetic algorithms itself is very hard to control. A proper genetic algorithm parameter setting will lead to a great searching speed and vice versa. Following the David's MicroGA Settings [5], the crossover rate is 0.5 and the mutation rate is 0.04 in all experiments. Since the search space of all the Minion switches is not large, we only considered 10 generations of the GA and the population size of GA was set to 6 and 8. Each trial was run 10 times and we observe the average of the minimums. In the random selection testing, we ran Minion with all the possible switch settings. After the comparison of GACM and random selection, the parameter sensitivity of genetic algorithm will be explored as well. The crossover rate and mutation rate in genetic algorithm are range from 0 to 0.9. Because the GACM will can't find good parameter at all when the crossover rate or the mutation is 1.

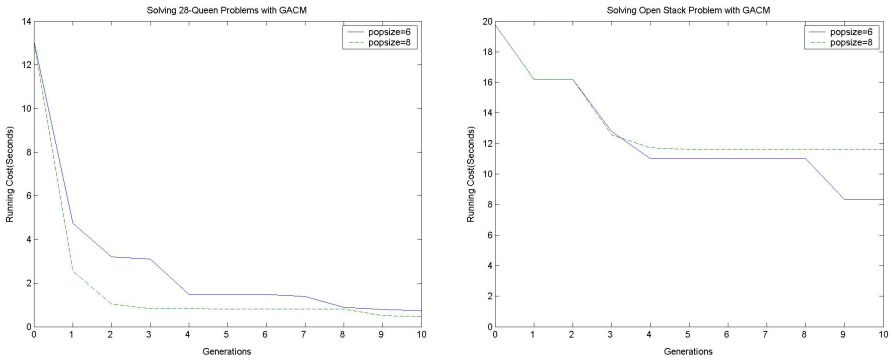


Fig. 2. The Efficiency of Solving Optimization Problems with Different Population Size by GACM

3 Experimental Results

From Figure 2, we can see the efficiency of solving optimization problems with GACM. The curves in Fig 2 illustrate that GACM will gained a satisfied parameter setting for Minion after just four generations whatever the population size or problem. The evolutionary speed of the genetic algorithm approached the minimum running cost after a few generations of the N-Queen problem (n=28). In the N-Queen problem the evolutionary speed of the genetic algorithm with population size 8 is better than the one with population size 6 from the left graph in figure 2. The wave situation of curves keeps the same when the queen size changes to 26 in N-Queen problem. The evolutionary result by using eight chromosomes is better than using six chromosomes. We gained the similar result when we using GACM to optimize the Minion tuning in the langford's number problem.

However the running cost curves in solving the N-queen and the Langford's number problem decrease comparatively gently compared to the one describing the running cost for solving the Open Stack problem. In the graph describing solving the Langford's number problem, it can be seen that the evolutionary speed will collapse after four generations when the population size is 8. The curve always stays level for a few generations and then descends when the population size is 6. In the open stack problem the figure shows a different result from the other two problems in that the final optimized value by 6 is greater than the one by 8.

To convince us of the efficiency of GACM, it is useful to compare the configuration result with the random selection in table 1. In table 1 there are three problems and two instances for each problem in the experiment. The data statistic on random selection are presented in the first 4 columns. The default solution in the first column of table 1 is the running cost of finding the solution by using Minion without using any switches turned on (default parameters). The optimal solution is the number of all Minion switches which provide a shorter running cost than without any switches. The average in table 1 means the running cost average of optimized solution which the minion can gain shorter running cost than using Minion default switch setting. The minimum solution is the minimum value of the running cost of using all the switch settings possibie.

According to the number and the value range of switches in Minion, the search space of GACM is 648. Table 1 shows that for all the problems nearly two-thirds of the switches lead Minion to less efficient performance than the default (no switches on). Although the GACM can't find the best switch setting for Minion in 10 generation, it can get a better switch setting than the random selection.

4 Future work

The result of this experiment, shows the efficiency of our GACM in Minion tuning. However there are a few challenges we need to face in the future. The

Question	Default Solution	Optimal Solution	Average	Minimum solution	Population Size=6	Population Size=8
N-Queen (n=26)	2.1	212	1.87	0.0313	0.52	0.31
N-Queen (n=28)	17.9	222	7.03	0.0313	0.73	0.47
Open Stack	2.4	142	2.3	0.48	1.61	1.03
Open Stack (new)	19.8	160	15.18	1.91	8.3	11.6
Lanford (2,10)	3.4	290	3.25	1.54	2.3	1.83
Lanford (3,17)	13.2	209	6.829	0.125	2.51	1.67

Table 1. The Efficiency of GACM in Solving Different Problems by comparing the Random Selection

GACM can definitely find a good parameter setting in a few generations, but we are unconvinced it can find the best setting. The genetic operator, crossover rate and mutation rate will be adapted to try to find this best setting. Different population sizes lead to different evolution for various problems. Can these constraint problems be classified into several categories? The parameter sensitivity of GA in GACM will also be explored further.

References

1. Ansótegui, Carlos and Sellmann, Meinolf and Tierney, Kevin, M. Sellmann and K. Tierney. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In Proceedings of 15th International Conference on Principles and Practice of Constraint Programming pp. 142-157(2009)
2. http://minion.sourceforge.net/
3. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison Wesley (1989)
4. Lars Kotthoff, Ian Miguel and Peter Nightingale. Ensemble classification for constraint solver configuration.In Proceedings of 16th International Conference on Principles and Practice of Constraint Programming, pp. 321-329(2010).
5. Carroll, D. L., Chemical Laser Modeling with Genetic Algorithms,AIAA J., Vol. 34, 2, pp. 338-346(1996)
6. Hutter, F., Hoos, H.H., Stutzle, T.: Automatic Algorithm Configuration based on Local Search. In: AAAI, pp. 1152C1157 (2007)
7. Adenso-Diaz, B., Laguna, M.: Fine-tuning of Algorithms using Fractional Experimental Design and Local Search. Operations Research 54(1), 99C114(2006)
8. Kanoh, H. and Matsumoto, M. and Nishihara, S.Systems, Man and Cybernetics, Genetic algorithms for constraint satisfaction problems. Intelligent Systems for the 21st Century. IEEE International Conference. pp 626-631. (2002)
9. Munoz, Jorge and Gutierrez, German and Sanchis, Araceli. Evolutionary Genetic Algorithms in a Constraint Satisfaction Problem: Puzzle Eternity II. Bio-Inspired Systems: Computational and Ambient Intelligence, vol 5517, pp720-727.(2009)

A constraint based approach to cyclic RCPSP

Alessio Bonfietti, Michele Lombardi, Luca Benini, and Michela Milano

DEIS, University of Bologna,
Viale del Risorgimento 2, 40136 Bologna, Italy
{alessio.bonfietti,michele.lombardi2,michela.milano,luca.benini}@unibo.it

Abstract. A cyclic scheduling problem is specified by a set of activities that are executed an infinite number of times subject to precedence and resource constraints. The cyclic scheduling problem has many applications in manufacturing, production systems, embedded systems, compiler design and chemical systems. This paper proposes a Constraint Programming approach based on Modular Arithmetic, taking into account temporal resource constraints. In particular, we propose an original modular precedence constraint along with its filtering algorithm. Classical "modular" approaches fix the modulus and solve an integer linear sub-problem in a generate-and-test fashion. Conversely, our technique is based on a non-linear model that faces the problem as a whole: the modulus domain bounds are inferred from the activity-related and iteration-related variables. The method has been extensively tested on a number of non-trivial synthetic instances and on a set of realistic industrial instances. Both the time to compute a solution and its quality have been assessed. The method is extremely fast to find close to optimal solutions in a very short time also for large instances. In addition, we have found a solution for one instance that was previously unsolved and improved the bound of another of a factor of 11.5%.

Keywords: Constraint Resource Constrained Cyclic Scheduling

Solving MAXSAT by Solving a Sequence of Simpler SAT Instances

Jessica Davies and Fahiem Bacchus

Department of Computer Science, University of Toronto,
Toronto, Ontario, Canada, M5S 3H5
{jdavies,fbacchus}@cs.toronto.edu

Abstract. MAXSAT is an optimization version of Satisfiability aimed at finding a truth assignment that maximizes the satisfaction of the theory. The technique of solving a sequence of SAT decision problems has been quite successful for solving larger, more industrially focused MAXSAT instances, particularly when only a small number of clauses need to be falsified. The SAT decision problems, however, become more and more complicated as the minimal number of clauses that must be falsified increases. This can significantly degrade the performance of the approach. This technique also has more difficulty with the important generalization where each clause is given a weight: the weights generate SAT decision problems that are harder for SAT solvers to solve. In this paper we introduce a new MAXSAT algorithm that avoids these problems. Our algorithm also solves a sequence of SAT instances. However, these SAT instances are always simplifications of the initial MAXSAT formula, and thus are relatively easy for modern SAT solvers. This is accomplished by moving all of the arithmetic reasoning into a separate hitting set problem which can then be solved with techniques better suited to numeric reasoning, e.g., techniques from mathematical programming. As a result the performance of our algorithm is unaffected by the addition of clause weights. Our algorithm can, however, require solving more SAT instances than previous approaches. Nevertheless, the approach is simpler than previous methods and displays superior performance on some benchmarks.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 98,
Perugia, Italy, 12 September 2011.

Filtering Algorithms for Discrete Cumulative Problems with Overloads of Resource

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

École des Mines de Nantes, LINA UMR CNRS 6241, 4, rue Alfred Kastler,
FR-44307 Nantes, France.
{ADeClerc, TPetit, NBeldice, NJussien}@mines-nantes.fr

Abstract. Many cumulative problems are such that the horizon is fixed and cannot be delayed. In this situation, it often occurs that all the activities cannot be scheduled without exceeding the capacity at some points in time. Moreover, this capacity is not necessarily always the same during the scheduling period. This article introduces a new constraint for solving this class of problems. We adapt two filtering algorithms to our context: Sweep and P. Vilím's Edge-Finding algorithm. We emphasize that in some problems violations are imposed. We design a new filtering procedure specific to this kind of events. We introduce a search heuristic specific to our constraint. We successfully experiment our constraint.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 99,
Perugia, Italy, 12 September 2011.

Synthesis of Search Algorithms from High-level CP Models*

Samir A. Mohamed Elsayed**, Laurent Michel

Computer Science Department, University of Connecticut.

Abstract. The ability to specify CP programs in terms of a declarative model and a search procedure is instrumental to the industrial CP successes. Yet, writing search procedures is often difficult for novices or people accustomed to model & run approaches. The viewpoint adopted in this paper argues for the synthesis of a search from the declarative model to exploit the problem instance structures. The intent is not to eliminate the search. Instead, it is to have a default that performs adequately in the majority of cases while retaining the ability to write full-fledged procedures. Empirical results demonstrate that the approach is viable, yielding procedures approaching and sometimes rivaling hand-crafted searches.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 100, Perugia, Italy, 12 September 2011.

Revisiting the `tree` Constraint

Jean-Guillaume Fages and Xavier Lorca

École des Mines de Nantes, INRIA, LINA UMR CNRS 6241,
FR-44307 Nantes Cedex 3, France
{Jean-guillaume.Fages,Xavier.Lorca}@mines-nantes.fr

Abstract. This paper revisits the `tree` constraint introduced in [2] which partitions the nodes of a n-nodes, m-arcs directed graph into a set of node-disjoint anti-arborescences for which only certain nodes can be tree roots. We introduce a new filtering algorithm that enforces generalized arc-consistency in $O(n + m)$ time while the original filtering algorithm reaches $O(nm)$ time. This result allows to tackle larger scale problems involving graph partitioning.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 101,
Perugia, Italy, 12 September 2011.

Grid-Based SAT Solving with Iterative Partitioning and Clause Learning

Antti E. J. Hyvärinen, Tommi Junttila, and Ilkka Niemelä

Aalto University
Department of Information and Computer Science
PO Box 15400, FI-00076 AALTO, Finland
{Antti.Hyvarinen,Tommi.Junttila,Ilkka.Niemela}@aalto.fi

Abstract. This work studies the solving of challenging SAT problem instances in distributed computing environments that have massive amounts of parallel resources but place limits on individual computations. We present an abstract framework which extends a previously presented iterative partitioning approach with clause learning, a key technique applied in modern SAT solvers. In addition we present two techniques that alter the clause learning of modern SAT solvers to fit the framework. An implementation of the proposed framework is then analyzed experimentally using a well-known set of benchmark instances. The results are very encouraging. For example, the implementation is able to solve challenging SAT instances not solvable in reasonable time by state-of-the-art sequential and parallel SAT solvers.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 102, Perugia, Italy, 12 September 2011.

Constraint Reasoning and Kernel Clustering for Pattern Decomposition With Scaling

Ronan LeBras[1], Theodoros Damoulas[1], John M. Gregoire[2]
Ashish Sabharwal[3], Carla P. Gomes[1], and R. Bruce van Dover[4]

[1] Dept. of Computer Science, Cornell University, Ithaca, NY 14853, USA
[2] School of Engr. and Applied Sciences, Harvard University, Cambridge, MA 02138
[3] IBM Watson Research Center, Yorktown Heights, NY 10598, USA
[4] Dept. of Materials Science and Engr., Cornell University, Ithaca, NY 14853, USA

Abstract. Motivated by an important and challenging task encountered in material discovery, we consider the problem of finding K basis patterns of numbers that jointly compose N observed patterns while enforcing additional spatial and scaling constraints. We propose a Constraint Programming (CP) model which captures the exact problem structure yet fails to scale in the presence of noisy data about the patterns. We alleviate this issue by employing Machine Learning (ML) techniques, namely kernel methods and clustering, to decompose the problem into smaller ones based on a global data-driven view, and then stitch the partial solutions together using a global CP model. Combining the complementary strengths of CP and ML techniques yields a more accurate and scalable method than the few found in the literature for this complex problem.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 103,
Perugia, Italy, 12 September 2011.

An Efficient Light Solver for Querying the Semantic Web

Vianney le Clément de Saint-Marcq[1,2], Yves Deville[1], and Christine Solnon[2]

[1] ICTEAM Research Institute, Université catholique de Louvain,
Place Sainte-Barbe 2, 1348 Louvain-la-Neuve (Belgium)
{vianney.leclement,yves.deville}@uclouvain.be
[2] Université de Lyon, Université Lyon 1, LIRIS, CNRS UMR5205, 69622 Villeurbanne (France)
christine.solnon@liris.cnrs.fr

Abstract. The Semantic Web aims at building cross-domain and distributed databases across the Internet. SPARQL is a standard query language for such databases. Evaluating such queries is however NP-hard. We model SPARQL queries in a declarative way, by means of CSPs. A CP operational semantics is proposed. It can be used for a direct implementation in existing CP solvers. To handle large databases, we introduce a specialized and efficient light solver, Castor. Benchmarks show the feasibility and efficiency of the approach.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 104, Perugia, Italy, 12 September 2011.

Boolean Equi-propagation for Optimized SAT Encoding

Amit Metodi[1], Michael Codish[1], Vitaly Lagoon[2], and Peter J. Stuckey[3]

[1] Department of Computer Science, Ben Gurion University of the Negev, Israel
[2] Cadence Design Systems, USA
[3] Department of Computer Science and Software Engineering, and NICTA Victoria Laboratory, The University of Melbourne, Australia

Abstract. We present an approach to propagation based SAT encoding, Boolean equi-propagation, where constraints are modelled as Boolean functions which propagate information about equalities between Boolean literals. This information is then applied as a form of partial evaluation to simplify constraints prior to their encoding as CNF formulae. We demonstrate for a variety of benchmarks that our approach leads to a considerable reduction in the size of CNF encodings and subsequent speed-ups in SAT solving times.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 105, Perugia, Italy, 12 September 2011.

Incorporating Variance in Impact-Based Search

Serdar Kadioglu[1], Eoin O'Mahony[2], Philippe Refalo[3], and Meinolf Sellmann[4]

[1] Brown University, Dept. of Computer Science, Providence, RI 02912, USA
serdark@cs.brown.edu
[2] Cornell University, Dept. of Computer Science, Ithaca, NY 14850, USA
eoin@cs.cornell.edu
[3] IBM, 1681 route des Dolines, 06560 Sophia-Antipolis, France
philippe.refalo@fr.ibm.com
[4] IBM Watson Research Center, Yorktown Heights, NY 10598, USA
meinolf@us.ibm.com

Abstract. We present a simple modification to the idea of impact-based search which has proven highly effective for several applications. Impacts measure the average reduction in search space due to propagation after a variable assignment has been committed. Rather than considering the mean reduction only, we consider the idea of incorporating the variance in reduction. Experimental results show that using variance can result in improved search performance.

Keywords: Search Strategies, Impact-based Search, Robust Search

Octagonal Domains for Continuous Constraints

Marie Pelleau, Charlotte Truchet, Frédéric Benhamou

LINA, UMR CNRS 6241
Université de Nantes, France
firstName.lastName@univ-nantes.fr

Abstract. Domains in Continuous Constraint Programming (CP) are generally represented with intervals whose n-ary Cartesian product (box) approximates the solution space. This paper proposes a new representation for continuous variable domains based on octagons. We generalize local consistency and split to this octagon representation, and we propose an octagonal-based branch and prune algorithm. Preliminary experimental results show promising performance improvements on several classical benchmarks.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 107,
Perugia, Italy, 12 September 2011.

A Quadratic Edge-Finding Filtering Algorithm for Cumulative Resource Constraints

Roger Kameugne[1,2], Laure Pauline Fotso[3], Joseph Scott[4], and Youcheu Ngo-Kateu[3]

[1] University of Maroua, Higher Teachers' Training College, Dept. of Mathematics, P.O Box 55 Maroua-Cameroon

[2] University of Yaoundé I, Faculty of Sciences, Dept. of Mathematics, PO Box 812, Yaoundé, Cameroon, rkameugne@yahoo.fr, rkameugne@gmail.com

[3] University of Yaoundé I, Faculty of Sciences, Dept. of Computer Sciences, PO Box 812, Yaoundé, Cameroon, lpfotso@ballstate.bsu.edu, mireille_youcheu@yahoo.fr

[4] Uppsala University, Dept. of Information Technology, Computing Science Division, Box 337, SE-751 05 Uppsala Sweden joseph.scott@it.uu.se

Abstract. The cumulative scheduling constraint, which enforces the sharing of a finite resource by several tasks, is widely used in constraint-based scheduling applications. Propagation of the cumulative constraint can be performed by several different filtering algorithms, often used in combination. One of the most important and successful of these filtering algorithms is edge-finding. Recent work by Vilím has resulted in a $\mathcal{O}(kn \log n)$ algorithm for cumulative edge-finding, where n is the number of tasks and k is the number of distinct capacity requirements. In this paper, we present a sound $\mathcal{O}(n^2)$ cumulative edge-finder. This algorithm reaches the same fixpoint as previous edge-finding algorithms, although it may take additional iterations to do so. While the complexity of this new algorithm does not strictly dominate Vilím's for small k, experimental results on benchmarks from the Project Scheduling Problem Library suggest that it typically has a substantially reduced runtime. Furthermore, the results demonstrate that in practice the new algorithm rarely requires more propagations than previous edge-finders.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 108, Perugia, Italy, 12 September 2011.

Pruning Rules for Constrained Optimisation for Conditional Preferences

Nic Wilson and Walid Trabelsi

Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland
n.wilson@4c.ucc.ie, w.trabelsi@4c.ucc.ie

Abstract. A depth-first search algorithm can be used to find optimal solutions of a Constraint Satisfaction Problem (CSP) with respect to a set of conditional preferences statements (e.g., a CP-net). This involves checking at each leaf node if the corresponding solution of the CSP is dominated by any of the optimal solutions found so far; if not, then we add this solution to the set of optimal solutions. This kind of algorithm can clearly be computationally expensive if the number of solutions is large. At a node N of the search tree, with associated assignment b to a subset of the variables B, it may happen that, for some previously found solution α, either (a) α dominates all extensions of b; or (b) α does not dominate any extension of b. The algorithm can be significantly improved if we can find sufficient conditions for (a) and (b) that can be efficiently checked. In case (a), we can backtrack since we need not continue the search below N; in case (b), α does not need to be considered in any node below the current node N. We derive a sufficient condition for (b), and three sufficient conditions for (a). Our experimental testing indicates that this can make a major difference to the efficiency of constrained optimisation for conditional preference theories including CP-nets.

Doctoral Program at the Interational Conference on Principles and Practice of Constraint Programming, CP 2011, page 109, Perugia, Italy, 12 September 2011.

Author Index