SofT'11 Co-located with CP 2011

11th Workshop on Preferences and Soft Constraints

Perugia, Italy 12th September 2011

Preface

This volume contains the papers presented at SofT-11: the 11th Workshop on Preferences and Soft Constraints held on 12th September 2011 and co-located with CP 2011 in Perugia, Italy.

Preferences are ubiquitous in real life: most problems are over-constrained and would not be solvable if we insist that all their requirements are strictly met. Moreover, many problems are more naturally described via preferences rather than hard statements. Soft constraints are the way the constraint community has extended its classical framework to deal with the concept of preferences.

The SofT-11 workshop will bring together researchers interested in all aspects of soft constraints and cost function processing, such as:

– theoretical frameworks
– problem modeling
– solving algorithms
– languages
– preference aggregation and elicitation
– multi-objective or qualitative optimization
– combining/integrating different frameworks and algorithms
– comparative studies
– real-life applications

The workshop is an opportunity to share knowledge between people working around algorithms and solvers for different formalisms, including Weighted Max-SAT, Soft CSP, Bayesian Networks, Random Markov Field, Factor Graphs, Pseudo Boolean Optimization, SAT Modulo Theories, and related formalisms.

There were 10 submissions. Each submission was reviewed by at least 2 programme committee members. The committee decided to accept all the papers. The programme gives a broad overview of the various researches done in the field of Preferences and Soft Constraints. There are papers focusing on finding the m-best solutions to a combinatorial optimization problem using Best-First or Branch-and-Bound search, papers on preference combination using Subjective Logic and papers efficiently solving a problem that generalizes submodular binary VCSPs. Other papers propose an extension to the Pareto Dominance relation and its application in Soft Constraints, or propose a Function Filtering enhancing Dynamic Programming methods. Moreover, other papers deal with expressing nonlinearity constraints in terms of Soft Global n-ary Constraints, solve the Crop Allocation Problem with constraints, formalize the m-best task within the unifying framework of Semirings, deal with decomposing Global Cost Functions, or model and solve the University course Timetabling Problem.

August 2011

Maria Silvia Pini
Francesco Santini
Kristen Brent Venable

Soft-11 Workshop Organization

Programme Chairs

Maria Silvia Pini, University of Padova, Italy
Francesco Santini, CWI, The Netherlands
Kristen Brent Venable, University of Padova, Italy

Programme Committee

Stefano Bistarelli, University of Perugia and IIT-CNR, Italy
Simon de Givry, INRA-Tolouse, France
Jimmy Lee, Chinese University of Hong Kong, China
Ines Lynce, Technical University of Lisbon, Portugal
Felip Manyà, IIIA-CSIC, Spain
Joao Marques-Silva, University College Dublin, Ireland
Pedro Meseguer, IIIA-CSIC, Spain
Barry O'Sullivan, 4C and University College Cork, Ireland
Emma Rollon, Technical University of Catalonia, Spain
Francesca Rossi, University of Padova, Italy
Toby Walsh, NICTA and University of New South Wales, Australia
Nic Wilson, 4C, Ireland

Additional Reviewers

Terrence Mak

Table of Contents

A Weighted CSP approach for solving spatio-temporal farm planning problems 1
Mahuna Akplogan, Jerome Dury, Simon De Givry, Gauthier Quesnel, Alexandre Joannon, Arnaud Reynaud, Jacques Eric Bergez and Frederick Garcia

Decomposing Global Cost Functions 16
Christian Bessiere, Patrice Boizumault, Simon De Givry, Patricia Gutierrez, Samir Loudni, Jean-Philippe Métivier, Thomas Schiex and Allouche David

Regularity Requirements in University Course Timetabling 31
Broes Cat, Gerda Janssens and Marc Denecker

Heuristic Search for m Best Solutions with Applications to Graphical Models 46
Rina Dechter, Natalia Flerova

Multi-Agent Preference Combination using Subjective Logic 61
Audun Jøsang

Domain permutation reduction for Valued CSPs 76
Wady Naanaa and Maher Helaoui

Sorted-Pareto Dominance: an extension to the Pareto Dominance relation and its application in Soft Constraints 91
Conor O'Mahony and Nic Wilson

Two-sided function filtering 104
Marc Pujol-Gonzalez, Jesus Cerquides, Pedro Meseguer and Juan Antonio Rodriguez Aguilar

Soft Nonlinearity Constraints and their Lower-Arity Decomposition 113
Venkatesh Ramamoorthy, Marius Silaghi, Toshihiro Matsui, Katsutoshi Hirayama and Makoto Yokoo

Inference Schemes for M Best Solutions for Soft CSPs 124
Emma Rollon, Natalia Flerova and Rina Dechter

A Weighted CSP approach for solving spatio-temporal farm planning problems

Mahuna Akplogan, Jérome Dury, Simon de Givry,
Gauthier Quesnel, Alexandre Joannon, Arnaud Reynaud,
Jacques Éric Bergez, and Frédérick Garcia

INRA, F-31320 Castanet Tolosan, France
{makploga,jdury,degivry,gquesnel,joannon,
areynaud,jbergez,fgarcia}@toulouse.inra.fr

Abstract. Applications regarding the crop allocation problem (CAP) are required tools for agricultural advisors to design more efficient farming systems. Despite this issue has been extensively treated by agronomists in the past, few methods tackle the crop allocation problem considering both the spatial and the temporal aspects of the CAP. In this paper, we precisely propose an original approach based on weighted CSP (WCSP) to address the crop allocation planning problem while taking farmers' management choices into account. These are represented as hard and preference constraints. We illustrate our proposition by some results based on a virtual case study. This preliminary work foreshadows the development of a decision-aid tool for supporting farmers in their crop allocation strategies.

Keywords: Weighted CSP, constraint satisfaction, optimization, spatio-temporal planning, crop allocation problem

1 Introduction

The design of a cropping plan is one of the first step in the process of crop production and is an important decision that farmers have to take. By cropping plan, we mean the *acreages* occupied by all the different crops every year and their *spatial allocation* within a farming land. The cropping plan decision can be summarized as (1) the choice of crops to be grown, (2) the determination of all crops' acreages, and (3) their allocation to plots. Despite the apparent simplicity of the decision problem, the cropping plan decisions depend on multiple spatial and temporal factors interacting at different levels of the farm management. The cropping plan decision-making combines long term planning activities, with managerial and operational activities to timely control the crop production process. Modelling a decision-making process to support such farmers' decisions therefore requires to consider the planning of crop allocation over a finite horizon, and to explicitly consider the sequence of problem-solving imposed by the changing context (e.g. weather, price).

In this paper, we precisely focus on the activity of planning seen as a spatio-temporal crop allocation problem (CAP) whose relevance is assessed by a global objective function. In addition to many approaches based on optimization procedure, the objective of the work is to propose new directions to address crop allocation while taking farmers' decision factors into account. These factors are formalized as hard and preference constraints in the WCSP framework. The choice of constraints is based on a survey of farmers' processes taking into account annual working hours capacity restrictions [5]. However, designing cropping plans with such an approach is still an open question due to many other decision factors that could be taken into account to solve the crop allocation problem. This preliminary work foreshadows the implementation of a spatially explicit decision-aid tool, namely CRASH (Crop Rotation and Allocation Simulator using Heuristics), developed for supporting farmers in their crop allocation strategies.

This paper is organized as follows. In section 2, we describe the crop allocation problem. It introduces some specific definitions and emphasize crop allocation problem. Section 3 describes existing approaches used to design cropping plans with a focus on their main limitations. In section 4, we introduce the constraint model compliant with the weighted CSP framework. In section 5, we illustrate our modelling approach by a virtual case study in order to highlight the interests of the proposed approach. And finally in section 6 we discuss and conclude the relevance and limits of using WCSP to solve the CAP.

2 Crop allocation problem (CAP)

2.1 Global description of the problem

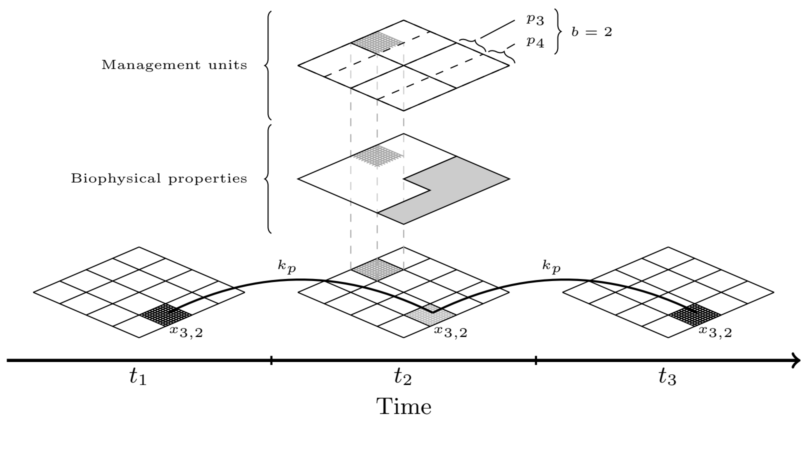


Fig. 1. Schematic representation of the spatial and temporal aspect of the decision-making problem (t_i: year, b: block, p_j: plot, $x_{b,i}$: landunit, k_p: preceding effect)

Let us consider a set of *landunits* defined as a piece of indivisible and homogeneous land whose historic and biophysical properties are identical. We define crop allocation as a spatio-temporal planning problem in which crops are assigned to landunits $x_{b,i}$ over a fixed horizon $\mathcal{H}$ of time (Fig. 1). These landunits are spatial sampling of the farmland where $x_{b,i}$ denotes the landunit i of *block* b.

The planning problem depends on multiple spatial and temporal factors. In space, these factors are organized in many different organizational levels called *management units* (Fig.1). These management units are decided by the farmer to organize his work and allocate resources. In order to simplify our example, we only considered the two main management units: *plot* (p_j) and *block* b. The first concerns the annual management of crops. A plot is a combination of landunits. Their delimitations are adapted over years in order to enforce the spatial balanced of crop acreages. As shown by Fig.1 *blocks* are subset of plots managed in a coherent way. Blocks are characterized by one cropping system defined by the same collection of crops and by the use of a coherent set of production techniques applied to these crops (e.g. fertilizer, irrigation water). The delimitation of blocks are not reshaped in the CAP considered in this work. They are mostly defined by the structural properties of the farm such as the availability of resources (e.g. access to irrigation water) and by the biophysical properties (eg. soil type, accessibility, topography). These *biophysical properties* are also used to define if a crop could not be produced in good condition on certain soil types.

In time, the sequence of crops on the same landunit is not allowed or not advisable without facing decrease in soil fertility, or increase in diseases or weeds infestation. We deal with these temporal factors by summarizing the assessment of crop sequence quality in two indicators: the *minimum return time (rt)* and the *Preceding effect(k_p)*. The *minimum return time (rt)* is defined as the minimum number of years before growing the same crop on a same landunit. On the figure 1, the minimum return time of the crop produced on $x_{3,2}$ (landunit 2 of block 3) at t_1 is equal to 2 years. More generally let t, t' be two different years ($t < t'$), $x_{b,i}$ a landunit and v a crop, $x_{b,i}^{t} = x_{b,i}^{t'} = v$ **if** $(t' - t) \geq rt(v)$.

The *preceding effect (k_p)* is an indicator representing the effect of the previous crop on the next one [12]. Based on k_p, some crop sequence can be ignored for their effects or recommended for their beneficial effects for production purposes. Further, some authors [4] have argued that the reproducibility of a cropping system over time is only ensured when crop allocation choices are derived from finite crop sequence which can be repeated over the time. We therefore introduce the concept of repeatability while looking for such a crop sequence. This means that the proposed crop sequence could be repeated over time without breaking the constraint rt. We introduce this concept, known as a *"crop rotation"*, because it is widely used by farmers as decision indicator.

2.2 Constraints description

Solving the crop allocation problem (CAP) is to assign crops to landunits $x_{b,i}$ over a fixed horizon $\mathcal{H}$ of time. An assignment of crops must satisfy a set of constraints.

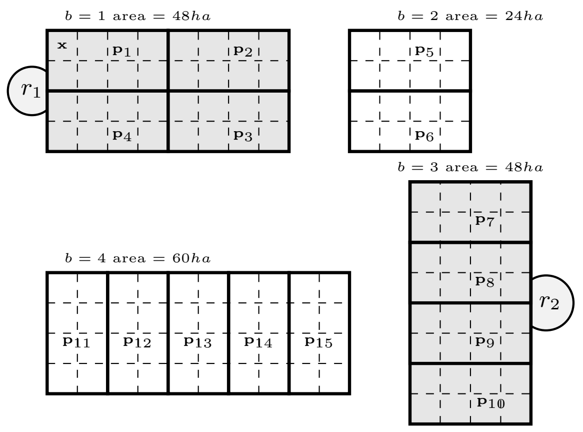
We retained as hard constraints the minimum *returned time* (rt), the *historic* of landunits and the *physical properties* (soil types, resource accessibility). Preference constraints are related to the *preceding effects* (k_p) and the spatio-temporal balance of crop acreages such that resources are efficiently used. Hard and preference constraints are defined either at:

– *plot level* to express for each plot (i) if they can be split/combined, (ii) if they must be fixed over the planning horizon in order to enforce the static aspect of the plot.
– *block level* to express for each landunit and crop the spatial compatibility of crop, the return time and the preceding effect.
– *farm level* to express preferences or the global use of resources.

Let us consider the crop allocation problem described in Fig. 2. In this problem, we consider 4 blocks and 15 plots sampled into 120 landunits. The size of the farmland (180 *ha*) and its sampling into landunits correspond to a middle real-world CAP. Four crops are produced over the all blocks: *winter wheat* (BH), *spring barley* (OP), *maize* (MA) and *winter rape* (CH). Each block has a fixed area (see Fig. 2). The blocks 1 and 3 have an access to irrigation equipments r_1 and r_2. The annual quota of irrigation water over the blocks is $6000m^3$ (respectively $4000m^3$) for r_1 (respectively r_2). Only the *maize* (MA) can be irrigated. There are two different types of soil: type 1 (block 1, 3) and type 2 (block 2, 4). The table on Fig. 2 shows the sequence of crops produced by each plot during the five previous years.

Spatio-temporal hard constraints

1. **h-SCC** - *spatial compatibility of crops*: for instance, the crop CH cannot be assigned to landunits whose soil type is 1 (block 1,3). This biophysical property is not suitable for the crop growing.
2. **h-EQU** - *landunit equality*: landunits on the plots p_7 (respectively p_9) and p_8 (respectively p_{10}) must have the same crop every year. Indeed, these landunits are decided by the farmer to be managed in the same manner.
3. **h-HST** - *landunit historic*: each landunit has defined historic values. The table in Fig. 2 defines the historic of each plot.
4. **h-TSC** - *temporal sequence of crop*: for each couple of crops and landunits, the minimum returned time rt must always be enforced. For instance in the CAP above, $rt(BH) = 2$, $rt(OP) = 3$, $rt(MA) = 2$ and $rt(CH) = 3$.
5. **h-CCS** - *cyclicity of crop sequence*: for each landunit, the crop sequence after the historic must be endlessly repeated by enforcing temporal sequence of crops.



Plots	t1	t2	t3	t4	t5
$\mathbf{p}_1$	MA	MA	BH	OP	MA
$\mathbf{p}_2$	OP	MA	MA	BH	OP
$\mathbf{p}_3$	BH	OP	MA	MA	BH
$\mathbf{p}_4$	MA	BH	OP	MA	MA
$\mathbf{p}_5$	BH	OP	BH	CH	BH
$\mathbf{p}_6$	OP	BH	CH	BH	OP
$\mathbf{p}_7$	MA	MA	MA	MA	MA
$\mathbf{p}_8$	MA	MA	MA	MA	MA
$\mathbf{p}_9$	MA	MA	MA	MA	MA
$\mathbf{p}_{10}$	MA	MA	MA	MA	MA
$\mathbf{p}_{11}$	BH	CH	BH	OP	BH
$\mathbf{p}_{12}$	CH	BH	OP	BH	CH
$\mathbf{p}_{13}$	BH	OP	BH	CH	BH
$\mathbf{p}_{14}$	OP	BH	CH	BH	OP
$\mathbf{p}_{15}$	BH	CH	BH	OP	BH

Fig. 2. A virtual farm with 4 blocks, 15 plots (12ha for each plot) split into 120 landunits. The grey blocks have their own irrigation equipment (r_1, r_2). The table contains the historic values for each plot

	previous crops			
	BH	OP	MA	CH
BH	4	1	1	0
OP	2	3	1	0
MA	0	0	3	0
CH	0	0	0	4

Fig. 3. Table of preceding effect

6. **h-RSC** - *resources capacity*: a fixed amount of resources are available. The quantities of resources accumulated on the landunits do not exceed some limits. For instance, in the CAP defined above, we have only one irrigated crop (*maize* - MA). Knowing that we need $165m^3$ of water by hectare, the annual production of MA on the blocks 1 cannot exceed $36,36$ *ha*.
7. **h-SCA** - *same crops assigned*: over the time, the same subset of crops must be assigned to every landunit of the same block.

Spatio-temporal preferences

1. **s-TOP** - *Farm topology*: landunits where the same crops are assigned must be spatially grouped. By this we mean that it is preferable to group as most as possible the same crop on the same block. Thus, traveling time can be reduced as well as the time spend by the farmers on operational activities that control the crop production process. Therefore, every isolated landunit is penalized by a cost δ_1.
2. **s-SBC** - *Spatial balanced of crop acreages*: a defined acreage of some crops every year. For instance, in the CAP defined above, the acreage of MA should

be within the range $[24, 48]$ ha on block 1 and $[12, 24]$ ha on block 3. Any deviation is penalized by a cost δ_2.
3. **s-TBC** - *temporal balanced of crop acreages*: a defined acreage of some crops on each landunit over years. In the CAP defined above, between $[12, 24]$ ha of crop CH should be produced on every landunit. Any deviation is penalized by a cost δ_3.
4. **s-CSQ** - *Crop sequence quality*: each pair of successive crops is associated to a cost k_p that defines its preceding effect. Fig. 3 define all k_p values.

In practice, we suggest to define the costs k_p, δ_1, δ_2 and δ_3 such that $\sum k_p > \sum \delta_2 > \sum \delta_1 > \sum \delta_3$. By doing so, a realistic hierarchy can be introduced among the soft constraints. Indeed, first and foremost, the preceding effects k_p must be minimized because of their consequences on the next crops. The spatial balanced of crop acreages related to cost δ_2, implicitly defines the annual receipts of the farmer. It must be ensured as much as possible. Afterwards the working hours can be reduced by grouping the same crops together (δ_1). Lastly, the additional preferences related to the temporal balanced of crop acreages (δ_3) can be enforced.

3 Related work

Since Heady [7], the cropping plan decision was represented in most modelling approaches as the search of the best land-crop combination [11]. Objectives to achieve a suitable cropping plan were often based on complete rationality paradigm using a single monetary criteria optimization, multi-attribute optimization [1] or assessment procedures [2]. In these approaches, the cropping plan decision is mainly represented into models by one of the two concepts, i.e. the cropping acreage [13, 10, 18] or crop rotation [6, 4]. These two concepts are two sides of the cropping plan decision problem, i.e. the spatial and temporal aspects. The originality of our approach lies on the consideration of both dimensions, i.e. spatial and temporal while solving the CAP. In most of the modelling approaches, the cropping plan is not spatially represented and is summarized as simple crop acreage distributions across various land types. At the farm level, the heterogeneity of a farm territory is generally described using soil type as the sole criterion [5].

4 Weighted CSP model of crop allocation

4.1 Weighted CSP Formalism

According to the CAP definition, and assuming a purely CSP formalism cannot deal with preferences easily, we focus on the Weighted CSP (WCSP) formalism which is more appropriate for solving optimization problems. The WCSP formalism [14] extends the CSP formalism by associating cost functions (or preferences) to constraints. A WCSP is a triplet $\langle \mathcal{X}, \mathcal{D}, \mathcal{W} \rangle$ where:

- $\mathcal{X} = \{1, \cdots, n\}$ is a finite set of n variables.
- $\mathcal{D} = \{D_1, \cdots, D_n\}$ is a finite set of variables domain. Each variable $i \in \mathcal{X}$ has a finite domain $D_i \in \mathcal{D}$ of values.
- $\mathcal{W} = \{W_{S_1}, \cdots, W_{S_e}\}$ is a set of cost functions where $S_i \subset \mathcal{X}$ be a subset of variables (i.e., the scope). We denote $l(S_i)$ the set of tuples over S_i. Each cost function W_{S_i} is defined over a subset of variables S_i ($W_{S_i} : l(S_i) \rightarrow [0, m]$ where $m \in [1, \cdots, +\infty]$).

Solving a WCSP is to find a complete assignment $A \in l(\mathcal{X})$ that minimizes $min_{(A \in l(\mathcal{X}))} \left[\sum_{W_{S_i} \in \mathcal{W}} W_{S_i}(A[S_i])\right]$, where $A[S_i]$ is the projection of a tuple on the set of variables S_i.

4.2 Crop allocation problem definition

The CAP is defined by a set of landunits and crops. The planning problem is defined over a finite horizon $\mathcal{H}$. We define the associated WCSP problem as follow.

$\mathcal{X}$ a set of variables $x^t_{b,i}$ that define the landunit i in block b ($i \in [1, \mathcal{N}_b]$, $b \in [1, \mathcal{B}]$ $\mathcal{B} = 4$ and $\mathcal{N}_1 = 32$ in the CAP described in Fig. 2) at year t ($t \in [1, \mathcal{H}]$). Thus, each landunit is described by $\mathcal{H}$ variables that represent the landunit occupation at every time. We define $[1, h]$ and $[h+1, \mathcal{H}]$ respectively the historic and the future times. For instance, following Fig. 2) and considering $\mathcal{H} = 9$ and $h = 5$, landunit i in block b will be represented by 9 variables where the first five variables (white nodes) are historic variables.

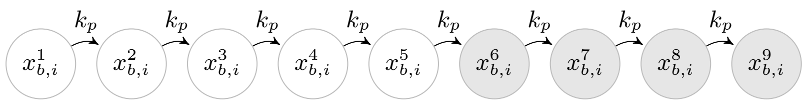


Fig. 4. A temporal sequence of variables over landunit i in block b

$\mathcal{D}$ the domains $D_{b,i}$ of variables $x^t_{b,i}$ is the set of possible crops over the landunit i in block b. Considering the problem in Fig. 2, $\forall b \in [1, \mathcal{B}], \forall i \in [1, \mathcal{N}_b]$, $D_{b,i} = \{1, 2, 3, 4\} = \{BH, OP, MA, CH\}$

$\mathcal{W}$ the cost functions are divided into five different types of hard and soft constraints: (1) simple tabular cost functions (arity up to 5), (2) same global constraint, (3) regular global constraint, (4) gcc global cardinality constraint, (5) soft-gcc soft global cardinality constraint. These cost functions are precisely defined in the next sections.

4.3 Simple cost functions

The hard and soft constraints h-SCC, h-EQU, h-HST, s-TOP and s-CSQ are defined by:

h-SCC : $\forall t \in [h+1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, let $W^{SCC}_{x^t_{b,i}}$ be a unary cost function associated to spatial compatibility of crops.

$$\forall a \in D_{b,i}, W^{SCC}_{x^t_{b,i}}(a) = \begin{cases} \infty & \textit{if } a \textit{ is forbidden for block } b\textit{, landunit } i \\ 0 & \textit{otherwise} \end{cases} \quad (1)$$

h-EQU : $\forall t \in [h+1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, for all couple of landunits $(i,j) \in \mathcal{N}_b \times \mathcal{N}_b$ that are decided by the farmer to be managed in the same manner, we define an equality constraint $W^{EQU}_{x^t_{b,i},x^t_{b,j}}$ between the two landunits.

$$\forall a \in D_{b,i}, \forall a' \in D_{b,j}, W^{EQU}_{x^t_{b,i},x^t_{b,j}}(a,a') = \begin{cases} 0 & \textit{if } a = a' \\ \infty & \textit{otherwise} \end{cases} \quad (2)$$

h-HST : $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, $\forall t \in [1, h]$, let $W^{HST}_{x^t_{b,i}}$ be an unary cost function associated to the historic values of landunits.

$$\forall a \in D_{b,i}, W^{HST}_{x^t_{b,i}}(a) = \begin{cases} 0 \text{ if } a = \mathsf{historic}(x^t_{b,i}) \\ \infty \quad \textit{otherwise} \end{cases} \quad (3)$$

where $\mathsf{historic}(x^t_{b,i})$ returns the historic value of landunit i in block b at time t.

s-TOP : $\forall t \in [1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, let W^{TOP}_S be an n-ary cost function associated to the farm land topology. We define a neighborhood function $\mathsf{neighbor}(i)$ which returns the landunits $j \in \mathcal{N}_b$ spatially close to i. For instance, in the CAP presented on Fig. 2, we consider the 4 nearest neighbors, the so-called von Neumann neighborhood. Here, the scope S is equal to $\{x^t_{b,i}, x^t_{b,n}, x^t_{b,s}, x^t_{b,e}, x^t_{b,w}\}$ where landunits n, s, e, w are the 4 nearest neighbors respectively at the North, South, East and West of i. $\forall a \in D_{b,i}, \forall a_n \in D_{b,n}, \forall a_s \in D_{b,s}, \forall a_e \in D_{b,e}, \forall a_w \in D_{b,w}$

$$W^{TOP}_S(a, a_n, a_s, a_e, a_w) = \begin{cases} 0 \text{ if } a = a_n = a_s = a_e = a_w \\ \delta_1 \quad \text{otherwise} \end{cases} \quad (4)$$

According to the position of i in its block, the arity of W^{TOP}_S could be reduced to 4 or 3.

s-CSQ : $\forall t \in [1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, let $W^{CSQ}_{x^t_{b,i},x^{t+1}_{b,i}}$ be a binary cost function associated to the preceding effect k_p.

We define a function $\mathsf{KP}(a,a')$ that returns the preceding effect k_p of doing the crop a' after a.

$$\forall a \in D_{b,i}, \forall a' \in D_{b,i}, W^{CSQ}_{x^t_{b,i},x^{t+1}_{b,i}}(a,a') = \mathsf{KP}(a,a') \quad (5)$$

4.4 Crop collection over a block using same constraints

h-SCA : considering a block b, the subset of $(\mathcal{H} - h)$ future variables $x^t_{b,i}$ (with $t \in [h+1, \mathcal{H}]$) associated to each landunit i in b must be assigned to the same crop collection. Thus, $\forall (i,j) \in \mathcal{N}_b \times \mathcal{N}_b$ (with $i \neq j$), the set of values assigned to the temporal sequence of variables defining i is a permutation of those of j. By using the same constraint introduced in [3] we define h-SCA. For each block b, we choose a leading landunit i. We then define a $2 * (\mathcal{H} - h)$-ary cost function W^{SCA}_S associated to each pair of sequence of variables that defines $x^t_{b,i}$ and $x^t_{b,j}$ $(i \neq j)$. Thus, the scope S is $\{x^{h+1}_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}, x^{h+1}_{b,j}, \cdots, x^{\mathcal{H}}_{b,j}\}$. Let $A[x^{h+1}_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}]$ and $A[x^{h+1}_{b,j}, \cdots, x^{\mathcal{H}}_{b,j}]$ denote the two sub-assignments of the variables in S. The constraint W^{SCA}_S requires that $A[x^{h+1}_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}]$ is a permutation of $A[x^{h+1}_{b,j}, \cdots, x^{\mathcal{H}}_{b,j}]$.

$$W^{SCA}_S = \mathsf{same}(\underbrace{x^{h+1}_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}}_{i}, \underbrace{x^{h+1}_{b,j}, \cdots, x^{\mathcal{H}}_{b,j}}_{j}) \tag{6}$$

4.5 Crop sequence using regular global constraints

The constraints h-TSC and h-CCS are related to temporal crop sequences. We represent them by using the regular constraint [16]. $\forall t \in [1, \mathcal{H}]$, $\forall b \in \mathcal{B}$, $\forall i \in \mathcal{N}_b$, $\forall a \in D_{b,i}$, let $M^a_{b,i}$ be a non deterministic finite automaton (NFA), $\mathcal{L}(M^a_{b,i})$ the language defined by $M^a_{b,i}$, and $S_{b,i}$ a temporal sequence of $\mathcal{H}$ variables that describes landunit i of block b over the horizon. Solving a $\mathsf{regular}(S_{b,i}, M^a_{b,i})$ constraint is to find an assignment $A[S_{b,i}]$ such that $A[S_{b,i}] \in \mathcal{L}(M^a_{b,i})$.

h-TSC : considering each landunit $x_{b,i}$, the crop sequence is enforced by defining for each crop $a \in D_{b,i}$ a language $\mathcal{L}(M^a_{b,i})$ such that the same value a is assigned to ($x^t_{b,i}$ and $x^{t'}_{b,i}$) *iff* $x^{t'}_{b,i}$ enforces the minimum returned time $rt(a)$ i.e., $\forall t' \neq t, \quad t' \geq t + rt(a)$. We define $\mathsf{regular}(S_{b,i}, M^a_{b,i})$ where $M^a_{b,i}$ is described as in Fig. 5 for crop $a = CH$ the minimum return time of which is $rt(CH) = 3$ years. Here, the initial state is 0 while final states are $4, 5, 6$. Arcs are labelled with crop values.

As shown by the NFA in Fig. 5, the historic variables are used to enforce the minimum return time over the future variables. We then define an $\mathcal{H}$-ary cost function $W^{TSC^a}_{S_{b,i}}$ associated to each pair of landunit i in block b and each crop a such that:

$$\forall b \in \mathcal{B}, \forall i \in \mathcal{N}_b, \forall a \in D_{b,i}, W^{TSC^a}_{S_{b,i}} = \mathsf{regular}(x^1_{b,i}, \cdots, x^t_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}, M^a_{b,i}) \tag{7}$$

h-CCS : considering each landunit $x_{b,i}$, we combine h-TSC with a repeatability constraint also defined by a set of regular constraints. The constraint h-CCS ensures that any crop sequence assignment after the historic can be endlessly

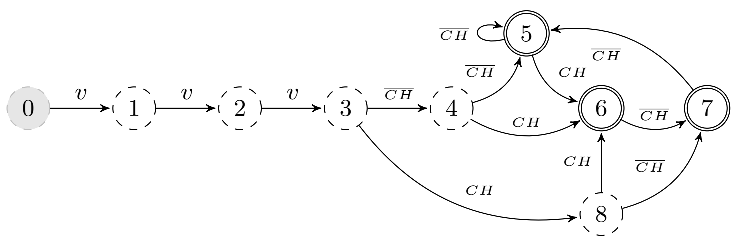


Fig. 5. Automaton for crop CH with $rt(CH) = 3$ and $h = 5$. v denotes any value in $D_{b,i}$. The notation $\overline{CH}$ corresponds to $D_{b,i} \setminus \{CH\}$. The associated language accepts every pattern over the historic variables and only the patterns that enforce the minimum return time in the future variables (e.g., CH-OP-CH-OP-CH-BH-OP-CH-BH).

repeated without violating the minimum return time constraint h-TSC. Fig. 6 describes a cyclic NFA for crop CH. The initial state is 0 while final states are $3, 6, 9, 12$. The scope of the cost function $W^{CCS^a}_{S_{b,i}}$ is restricted to future variables.

$$\forall b \in \mathcal{B}, \forall i \in \mathcal{N}_b, \forall a \in D_{b,i}, W^{CCS^a}_{S_{b,i}} = \mathsf{regular}(x^{h+1}_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}, M^a_{b,i}) \qquad (8)$$

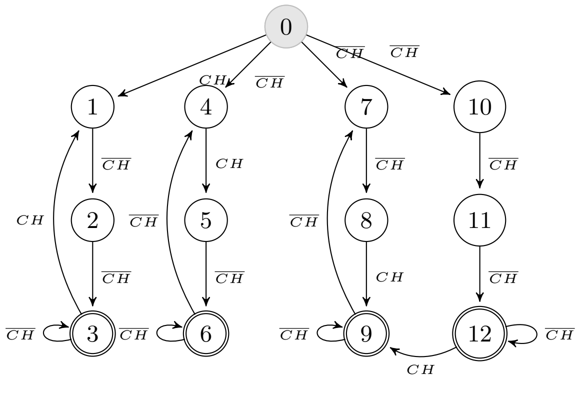


Fig. 6. Cyclic automaton for the crop CH with $rt(CH) = 3$ and $\mathcal{H} - h = 4$.

4.6 Resource capacity constraints using global cardinality constraints

In CAP, each landunit consumes a fixed amount of resources according to some structural (crop type, the area of landunits, etc.) and numerical (the irrigation dose) requirements. For instance, the maize (MA) is an irrigated crop whereas winter wheat (BH) does not need irrigation. A classical approach to deal with resources is to solve a shortest path problem with resource constraints [9]. The problem is NP-hard if the path needed is elementary. Loosely, solving a resource allocation problem involves both sequencing and counting reasoning. We assume in the CAP that this problem can be reduced to a counting problem under hypothesis 1 and 2.

Hypothesis 1 *: Resources are supposed to be usable and systematically renewed every year without doing anything (e.g. annual quota of irrigation water).*

This hypothesis is closed to a real CAP because farmers usually have a fixed quota of irrigation water. That can be exactly the case for the working hours capacity in a year if work regulations is taken into account.

Hypothesis 2 *: $\forall t \in [1, \mathcal{H}]$, $\forall (b, b') \in \mathcal{B} \times \mathcal{B}$ a couple of blocks, $\forall (i, j) \in \mathcal{N}_b \times \mathcal{N}'_b$ a couple of landunits. The areas of landunits i and j of block b (respectively b') can be considered equivalent according to the problem size.*

We make the assumption that the spatial sampling of the farm land into landunits is homogeneous. Under these hypothesis the annual resource allocation is seen as a counting problem at every time $t \in [h+1, \mathcal{H}]$. Thus, given annual resources capacities for a CAP, we define for each time $t \in [h+1, \mathcal{H}]$ an upper and lower bound to the number of variables $x^t_{i,b}$ that are assigned to a given crop according to both structural and numerical requirements.

h-RSC : to enforce resource capacity constraints h-RSC, we use the global cardinality constraint gcc [17] over the assignments of crops to landunits.

$\forall t \in [h+1, \mathcal{H}]$, let $W^{RSC}_{S^t_b}$ be a $\mathcal{N}_b$-ary global constraint associated to resource capacities.

Given $S^t_b = (x^t_{b,1}, \cdots, x^t_{b,\mathcal{N}_b})$ the global cardinality constraint (gcc) specifies, for each value $a \in \bigcup D_{b,i}$, an upper bound $ub(a)$ and a lower bound $lb(a)$ to the number of variables $x^t_{b,i}$ that are assigned to a.

$$W^{RSC}_{S^t_b} = \mathsf{gcc}(S^t_b, lb, ub) \tag{9}$$

has a solution if there exists an assignment of S^t_b such that

$$\forall a \in \bigcup D_{b,i},\ lb(a) \leq |\{x^t_{b,i} \in S^t_b | x^t_{b,i} = a\}| \leq ub(a) \tag{10}$$

4.7 Spatio-temporal balance of crops using soft-gcc

Preferences related to the spatio-temporal balance of crops (s-SBC and s-TBC) are defined as soft global cardinality constraints (soft-gcc) that allow the violation of both lower and upper bounds of the associated hard constraint gcc.

$$\textsf{soft-gcc}(S, lb, ub, z, \mu) = \{(A[S], a_z) | A[S] \in l(S), a_z \in D_z, \mu(A[S]) \leq a_z\} \quad (11)$$

where lb and ub are respectively the lower and upper bounds, z a cost variable with finite domain D_z, μ the violation measure for the global constraint soft-gcc. In this work, we use the variable-based violation measure (see [8]) which is the minimum number of variables whose values must be changed in order to satisfy the associated gcc constraint. Thus $\textsf{soft-gcc}(S, lb, ub, z, \mu)$ has a solution if $\exists\ A[S]$ such that $min(D_z) \leq \mu(A[S]) \leq max(D_z)$. Based on this definition the constraints s-SBC and s-TBC are formalized as follow.

s-SBC : $\forall t \in [h+1, \mathcal{H}]$, $\forall b \in \mathcal{B}' \subseteq \mathcal{B}$. Let $W^{SBC}_{S^t_b}$ be a $|\mathcal{B}'|$-ary soft-gcc constraint associated to block b at time t. The scope $S^t_b = \{x^t_{b,i} | i \in [1 \cdots \mathcal{N}_b]\}$.

$$W^{SBC}_{S^t_b} = \textsf{soft-gcc}(S^t_b, lb, ub, z, \mu) \quad (12)$$

s-TBC : $\forall b \in \mathcal{B}' \subseteq \mathcal{B}$, $\forall i \in \mathcal{N}_b$. Let $W^{TBC}_{S_{b,i}}$ be a $(\mathcal{H} - h)$-ary soft-gcc constraint associated to each landunit i. The scope $S_{b,i} = \{x^{h+1}_{b,i}, \cdots, x^{\mathcal{H}}_{b,i}\}$. Excepted the scope, $W^{TBC}_{S_{b,i}}$ is exactly defined as the global soft cardinality constraint defined for s-SBC.

5 Implementation

5.1 CAP instances description

We performed the experimentations by using four instances of the virtual farm presented in Fig. 2. Each instance corresponds to a new sampling of landunits. The number of landunits is increased from 15 to 120 (15, 30, 60, 120). For the CAP instance with 15 landunits, $\mathcal{N}_1 = \mathcal{N}_3 = 4, \mathcal{N}_2 = 2$ and $\mathcal{N}_4 = 5$ where $\mathcal{N}_i$ denotes the number of landunits in the block i. In this problem, sampling is done such that the plots (see Fig. 2) are also the landunits (12 ha per landunit). These landunits are gradually refined by splitting them into 2, 4 and 8 smaller ones, to respectively build the instances with 30, 60 and 120 landunits. These sampling are chosen to be representative of different farm sizes. The planning horizon is nine years. According to the minimum return time (*winter wheat* $rt(BH) = 2$, *spring barley* $rt(OP) = 3$, *maize* $rt(MA) = 2$ and *winter rape* $rt(CH) = 3$) the four last years are dedicated to the future while the five first are historic ones. We use the historic values presented in Fig. 2.

We should emphasis that there is no constraints or preferences between blocks as described in Section 2.2. Thus, we first focus on solving each block independently. The instances associated to the block 1 are B1-LU4, B1-LU8, B1-LU16, B1-LU32 respectively for 4, 8, 16, 32 landunits. For all these experimentations the costs associated to s-TOP, s-SBC and s-TBC are respectively $\delta_1 = 2$, $\delta_2 = 100$ and $\delta_3 = 10$. By doing so, we implicitly introduce a hierarchy among the soft constraints according to the criterion defined in the last paragraph of section 2.2. To fine-tune the weight of preceding effects k_p in the global cost function, we introduced a factor $\delta_4 = 10$ such that k_p are set to $\delta_4 * KP$. By doing so, the crop sequences that minimize the preceding effects are desired to be satisfied as much as possible.

Secondly, we add a new preference over all blocks in our original model. We define a new cost function $W^{SBC}_{S^t}$, extending the previous $W^{SBC}_{S^t_b}$ described in section 2.2 such that the annual global acreage of MA and BH over all blocks should be respectively within the range $[40, 72]$ ha and $[70, 100]$ ha. The CAP instances B1[1-4]-LU15(*), B[1-4]-LU30(*), B[1-4]-LU60(*) and B[1-4]-LU120(*) are associated to these new problems. The blocks are now interdependent and consequently the maximum arity of soft global cardinality constraints is equal to the total number of landunits. All of these instances are available in the cost function benchmark[1]. For each instance, the number of constraints is approximately equal to $\frac{5}{2} \times \mathcal{N} \times \mathcal{H} \pm 30$, where $\mathcal{N}$ denotes the number of landunits and $\mathcal{H}$ the planning horizon.

5.2 Analysis of the results

For solving the CAP, we use a Depth-First Branch and Bound (DFBB) algorithm implemented in the **Toulbar2** solver[2] (version 0.9.1) using default options. Columns $|\mathcal{X}|$ and $|\mathcal{W}|$ of Tab. 1 shows the number of variables and constraints for each instance.

The results presented in Tab. 1 are performed on a 2.27GHz Intel(R) Xeon(R) processor. Total CPU times are in seconds. We measure total times to find and prove optimality (column Time(s) of One optimal (DFBB)) starting with a relatively good upper bound (column UB). The initial upper bound has an important impact on performance. We chose its value empirically. Based on optimal values, we also measure total times to find all the optimal solutions (column Time(s) of All optimal (DFBB)) by setting the initial upper bound to the optimum (column Opt.) plus one.

While focusing on independent blocks, the best solution is got in less than a minute excepte for B1-LU32. The optimum is found and proved for all the instances. The differences between CPU times to find one optimal and all the optimal solutions is mainly due to the quality of the initial upper bound. The results found while introducing interdependence between blocks are also acceptable compared to the problem size. Indeed, the scope of some gcc and soft-gcc constraints is equal to the number of landunits (120 variables in the worse case). This may explain why the instance B[1-4]-LU120(*) is not closed after 48 hours.

[1] http://www.costfunction.org/benchmark?task=browseAnonymous&idb=33
[2] http://mulcyber.toulouse.inra.fr/projects/toulbar2

Table 1. An 0ptimal and all optimal solutions using DFBB

Instance of CAP	$\|\mathcal{X}\|$	UB	$\|\mathcal{W}\|$	Opt.	One optimal (DFBB)			All optimal DFBB			
					Time(s)	Nodes	BT	Time(s)	Nodes	BT	Nb.Sol
B1-LU4	36	1000	91	92	0.39	17	10	0.08	8	4	5
B1-LU8	72	2000	175	184	2.96	94	49	0.21	32	16	17
B1-LU16	144	4000	343	368	21.47	413	209	2.64	256	512	257
B1-LU32	288	6000	679	640	228	285	147	6.19	38	19	17
B2-LU2	18	1000	47	38	0.08	2	2	0.06	2	1	1
B2-LU4	36	2000	95	116	0.22	8	4	0.22	8	4	1
B2-LU8	72	4000	191	392	4.19	6	5	0.36	2	1	1
B2-LU16	144	6000	383	752	7.9	10	9	0.78	2	1	1
B3-LU4	36	1000	99	328	0.3	14	7	0.29	16	8	2
B3-LU8	72	2000	199	656	0.64	14	7	0.6	16	8	2
B3-LU16	144	4000	367	1312	1.51	18	9	1.37	16	8	2
B3-LU32	288	6000	703	2592	4.1	20	10	3.79	18	9	2
B4-LU5	45	1000	119	46	0.53	4	4	0.08	0	0	1
B4-LU10	90	2000	239	192	11.64	5	4	0.57	0	0	1
B4-LU20	180	4000	479	752	12.32	12	10	0.73	0	0	1
B4-LU40	360	6000	959	1504	39.33	23	19	1.97	2	1	1
B[1-4]-LU15(*)	135	2000	360	704	21.02	257	131	7.87	96	48	2
B[1-4]-LU30(*)	270	4000	712	1560	323.02	1029	521	155.9	498	249	12
B[1-4]-LU60(*)	540	4000	1384	3852	2412.97	1297	658	3697.23	2228	1114	136
B[1-4]-LU120(*)	1080	8000	2728	-	-	-	-	-	-	-	-

6 Conclusion

In this paper, we have modelled the crop allocation problem (CAP) using the Weighted CSP formalism. Contrary to existing approaches for solving such a problem, our proposition combines both the spatial and the temporal aspects of crop allocation. We explicitly described how the farmers' hard and soft constraints can be addressed as a global objective function optimization problem. The results have shown that on small and middle CAP, the *Toulbar2* solver can deliver relevant solutions in acceptable computational time. In the future, we will investigate the CUMULATIVE constraint for expressing more complex resource management and the COSTREGULAR constraint for mixing the return time and preceding effects, taking inspiration from the work done by [15].

References

1. J. Annetts and E. Audsley. Multiple objective linear programming for environmental farm planning. *Journal of the Operational Research Society*, 53(9):933–943, 2002.
2. J. Bachinger and P. Zander. ROTOR, a tool for generating and evaluating crop rotations for organic farming systems. *European Journal of Agronomy*, 26(2):130–143, 2007.

3. N. Beldiceanu, I. Katriel, and S. Thiel. Filtering algorithms for the same constraint. In *CPAIOR-04*, pages 65–79, 2004.
4. S. Dogliotti, W. A. H. Rossing, and M. K. van Ittersum. ROTAT, a tool for systematically generating crop rotations. *European Journal of Agronomy*, 19(2):239–250, 2003.
5. J. Dury, N. Schallers,F. Garcia, A. Reynaud and JE. Bergez. Models to support cropping plan and crop rotation decisions: a review. *Agronomy for Sustainable Development*, accepted, 2011.
6. T. El-Nazer and B. A. McCarl. The Choice of Crop Rotation: A Modelling Approach and Case Study. *American Journal of Agricultural Economics*, 68(1):127–136, 1986.
7. E. O. Heady. The Economics of Rotations with Farm and Production Policy Applications. *Journal of Farm Economics*, pages 645–664, 1948.
8. W.J. van Hoeve, G. Pesant,L.M. Rousseau. On global warming: Flow-based soft global constraints. *Journal of Heuristics*, pages 347–373, 2006.
9. S. Irnich and G. Desaulniers. *Shortest Path Problems with Resource Constraints*, chapter 2, pages 33–65. GERAD 25th Anniversary Series. Springer, 2005.
10. T. Itoh, H. Ishii, and T. Nanseki. A model of crop planning under uncertainty in agricultural management. *Int. J. of Production Economics*, 81-82:555–558, 2003.
11. W. K. Kein Haneveld and A. W. Stegeman. Crop succession requirements in agricultural production planning. *European Journal of Operational Research*, 166(2):406–429, 2005.
12. B. Leteinturier, J. Herman, F. D. Longueville, L. Quintin, and R. Oger. Adaptation of a crop sequence indicator based on a land parcel management system. *Agriculture, Ecosystems & Environment*, 112(4):324–334, 2006.
13. B. A. McCarl, W. V. Candler, D. H. Doster, and P. R. Robbins. Experiences with farmer oriented linear programming for crop planning. *Canadian Journal of Agricultural Economics/Revue canadienne d'agroeconomie*, 25(1):17–30, 1977.
14. P. Meseguer, F. Rossi, and T. Schiex. Soft Constraints Processing. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.
15. J.-P. Métivier, P. Boizumault, and S. Loudni. Solving nurse rostering problems using soft global constraints. In *CP-09*, pages 73–87, 2009.
16. G. Pesant. A regular language membership constraint for finite sequences of variables. In *CP-04*, pages 482–495, 2004.
17. J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *AAAI'96*, pages 209–215, 1996.
18. R. Sarker and T. Ray. An improved evolutionary algorithm for solving multi-objective crop planning models. *Computers and Electronics in Agriculture*, 68(2):191–199, 2009.

Decomposing Global Cost Functions

D. Allouche[1], C. Bessiere[2], P. Boizumault[3], S. de Givry[1], P. Gutierrez[4], S. Loudni[3], JP. Métivier[3], T. Schiex[1]

[1] UBIA, UR 875, INRA, F-31320 Castanet Tolosan, France
[2] LIRMM, Université de Montpellier, France
[3] GREYC, Université de Caen, France
[4] IIIA-CSIC, Campus Universitat Autònoma de Barcelona, 08193, Bellaterra, Spain

Abstract. Similarly to what has been done with Global Constraints in Constraint Programming, different results have been recently published on Global Cost Functions in weighted CSPs, defining the premises of a Cost Function Programming paradigm. In this paper, in the spirit of Berge-acyclic decompositions of global constraints such as Regular, we explore the possibility of decomposing Global Cost Functions in such a way that enforcing soft local consistencies on the decomposed cost function offers guarantees on the level of consistency enforced on the original global cost function. We show that an extension of Directional Arc Consistency to arbitrary arities and Virtual Arc Consistency offer specific guarantees. We conclude by preliminary experiments on WeightedRegular decompositions that show that decompositions may be very useful to easily integrate global cost functions in existing solvers with good efficiency.

.

Introduction

Graphical model processing is a central problem in AI. The optimization of the combined cost of local cost functions, central in the valued/weighted CSP frameworks [25], captures problems such as weighted MaxSAT, Weighted CSP or Maximum Probability Explanation in probabilistic networks. It has applications in *resource allocation*, *combinatorial auctions*, *bioinformatics*...

The main approach to solve such problems in the most general situation relies on Branch and Bound combined with dedicated lower bounds. Such lower bounds can be provided by enforcing soft local consistencies [5], leading to pruning as in Constraint Programming solvers. CP solvers are also equipped with global constraints which are often considered as crucial for solving large difficult problems. Dedicated algorithms for filtering such constraints have been introduced. For some classes of global constraints, among which the famous Regular constraint, it has been shown that using a direct decomposition of the constraint into

This work has been partially funded by the french "Agence nationale de la Recherche", reference ANR-10-BLA-0214.

a Berge-acyclic network of fixed arities constraints could lead to better efficiency, with a simpler implementation and without losing effectiveness in filtering.

The notion of global constraints has been recently extended to weighted CSP, defining Global Cost functions [26,19,18] with associated efficient filtering algorithms. In this paper, we consider the possible decomposition of global cost functions into Berge-acyclic networks and see if enforcing local consistency on the decomposition can lead to a filtering which is comparable to the filtering obtained by directly enforcing the same consistency on the original global cost function.

To give body to this notion of Berge-acyclic decomposable cost functions, we use the WeightedRegular cost function, which relies on a weighted finite automaton to define a cost function on assignments, considered as a regular language.

After some preliminaries introducing Cost Function Networks and Soft Local Consistencies, we define Cost function decomposition and show how it can be applied to the WeightedRegular cost function in Section 2. Section 3 then shows that enforcing soft local consistencies such as Directional Arc Consistency or Virtual Arc Consistency on Berge-acyclic decompositions is essentially equivalent to a direct application on the original global cost function. Finally, Section 4 reports preliminary experiments comparing the efficiency of decomposed vs. monolithic version of the WeightedRegular cost function used to model SoftRegular cost functions.

1 Preliminaries

A Cost Function Network (CFN) or weighted CSP is a pair (X, W) where $X = \{1, \ldots, n\}$ is a set of n variables and W is a set of cost functions. Each variable $i \in X$ has a finite domain D_i of values than can be assigned to it. A value a in D_i is denoted (i, a). The maximum domain size is d. For a set of variables $S \subseteq X$, D_S denotes the Cartesian product of the domain of the variables in S. For a given tuple of values t, $t[S]$ denotes the projection of t over S. A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D_S \mapsto [0, k]$ where k is a maximum integer cost (or ∞) used to represent forbidden assignments (expressing hard constraints). To faithfully capture hard constraints, costs are combined using the bounded addition defined by $\alpha \oplus \beta = \max(k, \alpha + \beta)$. Observe that the intolerable cost k may be either finite or infinite. A cost β may also be subtracted from a larger cost α using the operation $\ominus$ where $\alpha \ominus \beta$ is $(\alpha - \beta)$ if $\alpha \neq k$ and k otherwise. Without loss of generality, we assume that every network contains one unary cost function w_i per variable and a 0-arity (constant) cost function $w_\varnothing$. A tuple t_S is said to be valid iff $\forall i \in S, w_i(t[i]) < k$.

The associated hyper-graph of a CFN (X, W) is an hypergraph with one vertex per variable $i \in X$ and one hyperedge per scope S such that $\exists w_S \in W$. We consider CFN with connected hypergraphs. The intersection graph of an hypergraph has one vertex by hyperedge and an edge connects two vertices iff their associated hyperedges intersect. An hyper-graph is Berge acyclic iff hyperedges intersect by at most one vertex and its intersection graph is acyclic [1].

The central problem in CFN is to find an optimal *solution*: a complete assignment t minimizing the combined cost function $\bigoplus_{w_S \in W} w_S(t[S])$, with a cost strictly lower than k. This optimization problem has an associated NP-complete decision problem and restrictions to boolean variables and binary constraints are known to be APX-hard [22]. It federates a variety of famous problems including CSP, SAT, Max-SAT but also the *Maximum A posteriori Problem* (MAP) in Random Markov fields, the *Maximum Probability Explanation* (MPE) problem in Bayes nets [14] and quadratic pseudo-boolean optimization [3].

General exact methods for solving this minimization problem usually rely on branch and bound algorithms equipped with dedicated lower bounds. We focus in this paper on the incremental lower bounds provided by maintaining soft local consistencies at the arc level such as Directed Arc Consistency (DAC [6,17]) and Virtual Arc Consistency (VAC [5]).

DAC has been originally introduced on binary cost functions using the notion of strong support [5] and later extended to non binary cost functions in [24] and [19] with different definitions. These two definitions coincide on binary cost functions. In this paper, we use a simpler extension of DAC, and to avoid confusion, we call this variant T-DAC (for terminal DAC). Given a total order $\prec$ on variables, a binary CFN is said to be Terminal Directional Arc Consistent (T-DAC) w.r.t. $\prec$ iff for any cost function w_S, and for any value (i, a) of the maximum variable $i \in S$ according to $\prec$, there exists $t \in D_S, t[i] = a$ such that $w_i(a) = w_S(t) \bigoplus_{j \in S} w_j(t[j])$. The tuple t is a full support of (i, a) on w_S w.r.t. $\prec$. Note that either $w_i(a) = k$ and (i, a) does not participate in any solution or $w_i(a) < k$ and this implies that $w_S(t) \bigoplus_{j \in S, j \neq i} w_j(t[j]) = 0$.

Virtual Arc Consistency is a more recent local consistency property that establishes a link between a Cost Function Network $P = (X, W)$ and a Constraint Network denoted as $Bool(P)$ defined by the same set X of domain variables and such that every constraint in $Bool(P)$ is the result of the transformation of a cost function $w_S \in W$ into a constraint c_S with the same scope which forbids any tuple $t \in D_s$ such that $w_S(t) \neq 0$. A CFN P is said to be Virtually Arc Consistent iff the arc consistent closure of the constraint network $Bool(P)$ is non empty [5].

Enforcing soft local consistencies

Enforcing such soft local consistencies relies on so-called arc level *Equivalence Preserving Transformations* (EPTs) which apply to one cost function w_S [7]. Instead of just deleting domain values, EPTs may shift cost between w_S and the unary constraints $w_i, i \in S$ and therefore operate on a sub-network of P defined by w_S and denoted as $N_P(w_S) = (S, \{w_S\} \cup_{i \in S} w_i)$. The main EPT used by arc level soft local consistencies is described as Algorithm 1. This EPT shifts an amount of cost $|\alpha|$ between the unary cost function w_i and the cost function w_S. The direction of the cost move is given by the sign of α. The precondition guarantees that costs remain non negative in the resulting equivalent network.

Algorithm 1: The main cost shifting EPT used to enforce soft arc consistencies. The $\oplus, \ominus$ operations are extended to handle possibly negative costs as follows: for non negative costs α, β, we have $\alpha \ominus (-\beta) = \alpha \oplus \beta$ and for $\beta \leq \alpha$, $\alpha \oplus (-\beta) = \alpha \ominus \beta$.

```
1 Precondition: $-w_i(a) \leq \alpha \leq \min_{t \in D_S, t[\{i\}]=a}\{w_S(t)\}$;
2 Procedure Project($w_S, i, a, \alpha$)
3     $w_i(a) \leftarrow w_i(a) \oplus \alpha$;
4     foreach ($t \in D_S$ such that $t[\{i\}] = a$) do
5         $w_S(t) \leftarrow w_S(t) \ominus \alpha$;
```

To enforce T-DAC on a single cost function w_S, it suffices to first shift the cost of every unary cost function $w_i, i \in S$ inside w_S by applying Project$(w_S, i, a, -w_i(a))$ for every value $a \in D_i$. Let j be the maximum variable in S according to $\prec$, one can then apply Project(w_S, j, b, α) for every value (j, b) and $\alpha = \min_{t \in D_S, t[j]=b} w_S(t)$. Let t be a tuple where this minimum is reached. t is then a strong support for (j, b): $w_j(b) = w_S(t) \bigoplus_{i \in S} w_i(t[i])$. This support can only be broken if for some unary cost functions $w_i, i \in S, i \neq j$ and $w_i(a)$ increases for some value (i, a).

To enforce T-DAC on a the complete CFN (X, W), one can simply sort W according to the order of the maximum variable of every cost function according to $\prec$ and apply the previous process on each cost function, successively. When a cost function w_S is processed, all the cost functions whose maximum variable appears before the maximum variable of S have already been processed which guarantees that none of the established full support will be broken. Enforcing T-DAC is therefore in $O(ed^r)$ in time. Using the Δ data-structures introduced in [5], space can be reduced to $O(edr)$.

The most efficient algorithms for enforcing VAC [5] actually enforce an approximation of VAC called VAC$_\varepsilon$ with a time complexity in $O(\frac{ekd^r}{\varepsilon})$ and a space complexity in $O(edr)$. Alternatively, and considering the worst case where the intolerable cost k is finite, Optimal Soft Arc Consistency can be used to enforce VAC in $O(e^{6.5}d^{(3r+3.5)} \log M)$ time (where M is the maximum finite cost in the network).

2 Decomposing Global Cost Functions

Global constraints are usually described as families of constraints with a precise semantics parametrized by the number of variables they involve. Most of the usually considered global constraints allow for efficient local consistency enforcing (compared to the default GAC algorithm). The notion of global constraints has been extended to define Soft Global Constraints such as SoftAllDiff or SoftRegular [11]. These "soft" global constraints are not cost functions but classical global constraints defined over a set of variables which includes a dedicated "cost" variable representing the cost of the assignment of the remaining variables under the precise softened global constraint semantics. For several such constraints, efficient dedicated algorithm for enforcing Generalized Arc Consistency have been introduced [11].

Recently, different papers [26,19,18] have shown that it is possible to define Global Cost Functions as cost functions with a precise semantics parametrized by the number of variables they involve, together with efficient soft local consistency enforcing algorithms. Compared to the previous cost variable based approach, this new approach offers improved propagation thanks to the enhanced communication between cost functions enabled by the arc level EPTs used to enforce Soft AC [7], DAC and FDAC [6,17], EDAC [16], OSAC and VAC [5].

2.1 Decomposing Cost Functions

Similarly to constraints, cost functions may possibly decompose into a set of cost functions of smaller arities.

Definition 1. *A cost function z_T decomposes into a cost function network $(T \cup E, F)$ iff $\forall t \in D_T, z_T(t) = \min_{t' \in D_{T\cup E}, t'[T]=t} \bigoplus_{w_S \in F} w_S(t'[S])$.*

Clearly, if z_T appears in a CFN $P = (X, W)$ and decomposes into $(T \cup E, F)$, then the optimal solutions of P can be directly obtained by projecting the optimal solutions of the CFN $P' = (X \cup E, W \setminus \{z_T\} \cup F)$ on X.

For technical reasons, we introduce the notion of extra-minimal decompositions.

Definition 2. *A decomposition $(T \cup E, F)$ of z_T is said to be extra-minimal iff all variables in E are involved in at least two cost functions in F.*

This is done without loss of generality since from any decomposition, one can easily produce an extra-minimal decomposition: for any extra variable $i \in T$ which is involved in just one cost function $w_S \in F$, we can eliminate i from E, replace w_S by the cost function $f = \min_i w_S$ on $S \setminus \{i\}$ and get a network $(T \cup E \setminus \{i\}, F \cup \{f\} \setminus \{w_S\})$ which is an extra-minimal decomposition. This process removes extra variables and reduces scopes and therefore preserves Berge-acyclicity.

Example 1. Consider the soft All-different cost function with the so-called decomposition measure [11]: the cost of an assignment is equal to the number of pairs of variables taking the same value. This global cost function can be decomposed in a set of $\frac{n.(n-1)}{2}$ binary soft difference cost functions, each involving a different pair of variables. A soft difference cost function takes cost 1 iff the two involved variables are equal and 0 otherwise. In this case, no extra variable is required and the decomposition is therefore already extra-minimal.

As for global constraints, using decomposition may lead to more local reasoning which may be less effective. In all cases, it facilitates the implementation of the global cost function in solvers without requiring the cost function to be "projection safe" (ability to perform EPTs on the internal representation of global cost functions directly, as introduced in [19]).

2.2 From Regular to Weighted Regular

Initially introduced in [23], the $\textsc{Regular}_{\mathcal{A}}(i_1, \dots, i_n)$ global constraint authorizes a tuple $(v_1, ..., v_n)$ iff it is a string of the language defined by the Finite Automaton $\mathcal{A}$. A deterministic finite automaton (DFA) is defined by (Q,Σ,θ,q_0,F), where Q is is a finite set of states, Σ is a finite set of symbols (the alphabet), $\theta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of final (or accepting) states. A non-deterministic finite automaton (NFA) is defined by (Q,Σ,δ,q_0,F) where δ is a transition function from $\Sigma \times Q \to 2^Q$. The automaton starts in the initial state q_0. In state q, the automaton inputs the next symbol $s \in \Sigma$ and moves to a state in $\delta(s, q)$. The string is accepted iff there is a path that ends in a state $q \in F$. The whole set of strings accepted by a N/DFA $\mathcal{A}$ is the language recognized by it, noted $L(\mathrm{A})$. A regular language is a language which can be recognized by a DFA or a NFA.

The $\textsc{SoftRegular}^d_{\mathcal{A}}(i_1, \dots, i_n, z)$ global constraint can be directly softened using any distance d between strings. This global constraint authorizes a tuple $(v_1, ..., v_n, c)$ iff the minimum distance according to d between $(v_1, \dots, v_n)$ and a string of the language of $\mathcal{A}$ is c. Traditional distances between strings of the same length such as the Hamming distance (number of positions at which string differs) or the Edit distance (minimum number of substitutions, insertions and deletions needed to edit one string into the other) have been considered and dedicated global constraints with cost variables proposed in [11].

More recently, [8] has been considering using weighted automata as a more general way of expressing soft regular constraints. This has also been extended to CFG (Context Free Grammars) in [13]. We follow the same idea and consider the $\textsc{WeightedRegular}$ global *cost function*, defined through a weighted automaton.

Weighted Automata and Language A weighted version of a N/DFA was introduced in [12]. A weighted finite automaton (WFA) is a FA where the transition function δ is replaced by a transition cost function σ. In our WCSP context, this function will output cost in $[0, k]$: $\sigma : Q \times \Sigma \times Q \to [0, k]$. An additional "exit" cost function ρ encodes cost for exiting the automaton $\rho : F \to [0, k]$[5].

The weighted automaton starts in the initial state q_0. In state q, the automaton inputs the next symbol $s \in \Sigma$ and moves to a state q', paying a cost $\sigma(q, s, q')$. The string is accepted iff a state $q \in F$ is ultimately reached. The cost of such an accepting path is the sum of the cost of all the transitions used (including the final step reaching an element of F, defined by ρ). The cost associated with a string ℓ is defined as the minimum cost over all possible accepting paths. If a string cannot be derived, then its associated cost is just k (the intolerable cost).

The $\textsc{WeightedRegular}_{\mathcal{A}}(i_1, \dots, i_n)$ cost function is defined on a sequence of variables T from a weighted automaton $\mathcal{A}$ using domain values as symbols in the set Σ. The cost of an assignment is just the cost of the string defined by the assignment of T according to $\mathcal{A}$.

[5] The usual definition of weighted automata includes also an "entry" cost function [21]. We don't use it in this paper.

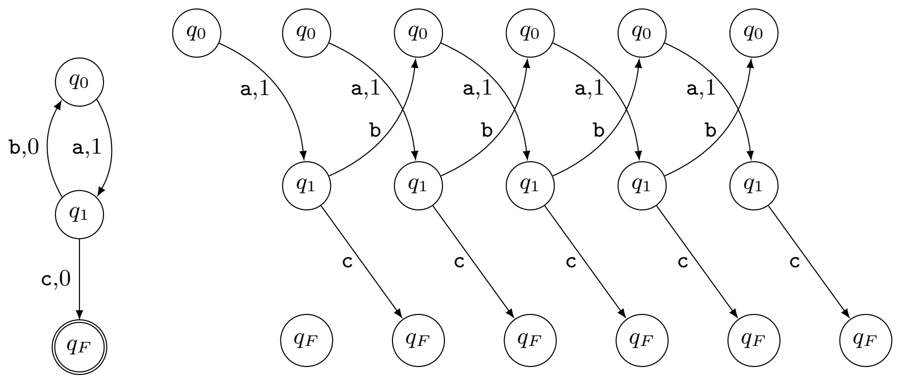
2.3 Decomposing Weighted Regular

We decompose the WeightedRegular using ternary cost functions encoding the weighted automaton and a sequence of extra state variables. This decomposition is similar in essence to the original decomposition of the Regular constraint but relies on cost functions. The decomposition is defined by:

- the original domain variables $i_1, \ldots, i_n \in T$,
- extra domain variables $s_0, \ldots, s_n$ representing automaton states. The domain of s_0 is just $\{q_0\}$, the domain of s_n is F. All other s_* variables have a domain equal to Q, the set of possible states.
- the set of ternary cost functions w_{s_{j-1}, i_j, s_j} which returns $\sigma(s_{j-1}, i_j, s_j)$.
- a unary cost function w_{s_n} on s_n directly defined by ρ.

By construction, the minimum cost that $\bigoplus_{j=1}^{n} w_{s_{j-1}, i_j, s_j} \oplus w_{s_n}$ can take is precisely the cost defined by $\text{WeightedRegular}_{\mathcal{A}}(i_1, \ldots, i_n)$.

This construction can be seen as a WCSP representation of the "unrolled" automaton transition graphs over n steps where each variable s_j represents the possible states at step j. The figure below illustrates this on a simple (deterministic) automaton for words $a(ba)^*c$ with each occurrence of a having cost 1 and a sequence of 6 variables (this automaton can only emit even length words whose cost is half the length). The unrolled automaton on the right does not mention costs for clarity (cost 0 everywhere except for edges emitting a with cost 1). Omitted edges represent intolerable k costs. It should be clear that one can associate one variable with each "column" of states, from the first to the last column. The additional variable i_j capture the possible characters emitted, and the triple has the associated emitting cost.



Similarly to what has been shown in cost variable decomposition using context free grammars [13], it is possible to encode Hamming and Edit distance based soft constraints using a weighted automaton and therefore a WeighteDRegular cost function.

Considering the Hamming distance, from an original DFA $\mathcal{A}$, we just derive a weighted automata with the same alphabet and states, with a constant zero exit function ρ and a transition cost function $\sigma(q, s, q')$ defined as:

- 0 whenever $q' \in \delta(s, q)$ in $\mathcal{A}$,
- 1 whenever $q' \notin \delta(s, q)$ in $\mathcal{A}$, but $\exists t \in \Sigma, t \neq s$, such that $q' \in \delta(t, q)$,
- the intolerable cost k otherwise.

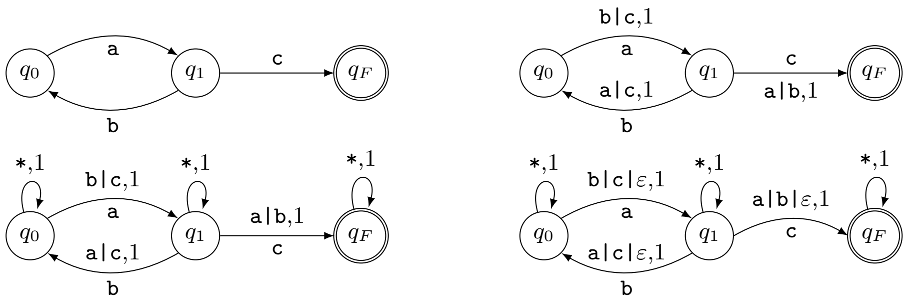


Fig. 1. From top-left to bottom-right, in reading order: (tl) the DFA for the language $a(ba)^*c$ (tr) the WFA encoding the Hamming distance to the previous automaton, (bl) the WFA allowing both for substitution and insertion (br) the WFA for the Edit distance. An arc is labelled by symbols,cost pairs where 0 costs are omitted and a '|' separated list of symbols is used to factorize several transitions with same source, destination and cost.

The Edit distance $d(s_1, s_2)$ of two words s_1 and s_2 is the smallest number of insertions, deletions, and substitutions required to change one word into another. It captures the fact that two words that are identical except for one extra or missing symbol should be considered close to one another. Considering the Edit distance from an original DFA $\mathcal{A}$, we derive a WFA with alphabet $\Sigma \cup \{\varepsilon\}$ and states Q with four kinds of transition cost functions:

1. copy of automaton $\mathcal{A}$: $\sigma(q, s, q')$=0 for each transition $q' \in \delta(s, q)$ in $\mathcal{A}$,
2. substitution: transition cost function is the same as for Hamming,
3. insertion: for each $q \in Q$ *s.t.* $q \notin \delta(s, q)$, $\sigma(q, s, q)$=1
4. deletion: to each transition $q' \in \delta(s, q)$ in $\mathcal{A}$ is associated an ε-transition[6] $q' \in \delta(\varepsilon, q)$ *s.t.* $\sigma(q, \varepsilon, q')$=1

Consider the DFA for the language $a(ba)^*c$ (Fig. 1-top-left). The WFA for substitution is depicted Fig. 1 (top-right), for substitution/insertion Fig. 1 (bottom-left) and for substitution/insertion/deletion Fig. 1 (bottom-right).

[6] An ε-transition is a transition using an empty symbol ε which does not emit anything. In practice, to derive a given word, a transition from state q to q' emitting ε means that we can directly go from q to q' without considering available symbols.

Unfortunately, the decomposition we presented in the beginning of this Section for WeightedRegular based on WFA without ε-transitions does not extend directly to WFA with ε-transitions, which are used for deletions. However, ε-transitions can be removed by computing the ε-closure of the WFA (see [21]). Then, the resulting WFA can be directly decomposed.

3 Local Consistency and Decompositions

Decomposing large arity constraints or cost functions into an equivalent combination of smaller arity components may sometimes be practically useful by itself: small arity may weaken propagation but may improve efficiency. Ultimately, this is a matter of compromise (especially in unstructured domains where there is no dedicated propagator for the considered cost function [9]).

However, for global cost functions, which have a precise semantics, it may be possible to define decompositions such that enforcing a given level of consistency on the decomposition offers a guarantee on the strength of the filtering compared to what would have been done using the original (non decomposed) cost function.

On classical constraint networks, different global constraints, such as the Regular global constraint, can be decomposed into a Berge-acyclic set of small arity constraints. By virtue of arc consistency, it is known that enforcing GAC on the set of Berge-acyclic constraint enforces GAC on the global constraint itself [1]. We show that a similar result can be obtained for cost functions and we illustrate this on the WeightedRegular cost function.

3.1 Directional AC

In this section, we will show that enforcing T-DAC on a decomposition of a cost function is equivalent to enforcing it on the original cost function itself, as far as the decomposition is Berge-acyclic.

We consider a decomposable global cost function z_T on the variables $T =$ with possible associated unary cost function $w_i, i \in T$. We assume that z_T can be decomposed in an extra-minimal Berge-acyclic cost function network $N = (T \cup E, F)$.

We now show that there exists a variable ordering on $T \cup E$ such that enforcing T-DAC on the decomposition is as strong as enforcing T-DAC on the original global cost function z_T.

Theorem 1. *In a CFN, if a global cost function z_T decomposes into an extra-minimal Berge-acyclic cost function network $N = (T \cup E, F)$ then there is an ordering on $T \cup E$ such that the unary cost function w_{i_n} on the last variable i_n produced by enforcing T-DAC on the subnetwork $(T, \{z_T\} \cup_{i \in T} w_i)$ is identical to the unary cost function w'_{i_n} produced by enforcing T-DAC on the decomposition $N = (T \cup E, F \cup_{i \in T} w_i)$.*

Proof. We first show that for any ordering of the variables, and any value (i_n, a) of the last variable i_n of T according to the ordering, enforcing T-DAC on the subnetwork $(T, \{z_T\} \cup_{i \in T} w_i)$ guarantees that the single value assignment (i_n, a) can be extended to a complete assignment t of T with cost $w_{i_n}(a)$. Let us consider t a full support of (i_n, a) on z_T. By definition $w_{i_n}(a) = z_T \bigoplus_{i \in T} w_i(t[i])$ which is precisely the cost of the complete assignment t in the subnetwork $(T, \{z_T\} \cup_{i \in T} w_i)$.

Conversely, we will now prove that the same property holds after enforcing T-DAC on the decomposition of z_T. The proof is not difficult but technical. Let $L = (F, I)$ be the intersection graph associated with the decomposition. L has one vertex per cost function in F and an edge $e \in I$ connects any two cost functions sharing a variable. Because of the Berge-acyclicity of N, L is acyclic (a forest). We can assume without loss of generality that L is connected (a tree). Indeed, if it is not connected, we can select two connected components and link them by adding one dummy constant zero binary cost function in F and just repeat this until L is connected. This preserves Berge-acyclicity and does not change the maximum arity in F.

We root L in any cost function that involves at least one original variable in T. We then perform a topological sort on L which successively selects a cost function with just one non selected neighbor in L (a leaf), choosing first cost functions whose intersection with this neighbor is an extra variable from E. Let $f_{S_1}, \ldots, f_{S_e}$ be the sequence of cost functions that are successively removed by the topological sort. Note that f_{S_i} intersects with $f_{S_{i+1}}$ by at most one variable (Berge acyclicity). This intersecting variable will be denoted as $\mathit{Root}(f_{S_i})$.

This ordering on cost functions can be extended to an order on variables as follows: we take each cost function f_{S_i} in the previous order and replace it by the sequence of all the variables in S_i which have not already been ordered. This sequence itself is ordered in such a way that the variable $\mathit{Root}(f_{S_i})$ that intersects with $f_{S_{i+1}}$ (if any) is ordered last. This produces an order $j_1, \ldots, j_m$ over variables in $T \cup E$.

First note that the variable j_m must be a variable of T. Indeed, since the decomposition is network minimal, all extra variables in f_{S_e} must appear in the intersection between scopes with previous cost functions. Because of the topological ordering chosen, if j_m is en extra variable, this means that f_{S_e} involves only extra variables which contradicts the choice of a root with a variable from T. Thus, $j_m \in T$ and $j_m = i_n$.

Assume now that $(T \cup E, F \cup_{i \in T} w_i)$ is made T-DAC consistent (which does not change scopes). Consider a value (i_n, a). If $w_{i_n}(a) = k$, then clearly any complete assignment extending this value has a cost of k and the property is proved. Otherwise, $w_{i_n}(a) < k$. Let t_e be a strong support of this value. We have $w_{i_n}(a) = w_{S_e}(t_e) \bigoplus_{i \in S_e} w_i(t_e[i])$ which proves that t_e is an assignment of S_e with the same cost $w_{i_n}(a)$ and such that $\forall i \in S_e, i \neq i_n), w_i(t_e[i]) = 0$. This is specifically true for any of the variables i in S_e that intersect with scopes of children cost functions, where the value $t_e[i]$ will have a zero unary cost. Since L is a tree, we can inductively use the same argument based on T-DAC to show that the tuple t_e can be extended to more variables with no increase in cost until the leafs of the tree are reached and all variables are assigned. This proves that the assignment (i_n, a) can be extended to complete assignment of $(T \cup E, F \cup_{i \in T} w_i)$ with cost $w_{i_n}(a)$. □

This result shows that directional consistency has enough power to handle Berge-acyclic decompositions in Weighted CSP without losing any propagation strength, provided a correct order is used for cost function propagation.

In practice, a CFN may contain different Berge-acyclic decomposable cost functions sharing variables and there may be no variable order which would be compatible with all the Berge-acyclic structures of these decomposed cost functions. In this case, a possibility would be to use so-called "Propagator groups" [15] which have been recently proposed for Berge-acyclic propagation in CN. Each propagator group is in charge of propagating one decomposition. For CFN, proper ordering between groups will likely be needed to avoid cycling.

3.2 Virtual AC

Virtual Arc Consistency offers a simple and direct link between CNs and CFNs which allows to directly lift classical CN's properties to CFNs, under simple conditions.

Consider a decomposable global cost function z_T on the variables T with possible associated unary cost functions $w_i, i \in T$. We assume that z_T can be decomposed in an extra-minimal Berge-acyclic cost function network $N = (T \cup E, F)$.

Theorem 2. *In a CFN, if a global cost function z_T decomposes into a Berge-acyclic cost function network $N = (T \cup E, F)$ then enforcing VAC on either $(T, \{z_T\} \cup_{i \in T} w_i)$ or on $(T \cup E, F \cup_{i \in T} w_i)$ yields the same lower bound $w_\varnothing$.*

Proof. Enforcing VAC on the CFN $P = (T \cup E, F \cup_{i \in T} w_i)$ does not modify the set of scopes and yields an equivalent problem P' such that $Bool(P')$ is Berge-acyclic, a situation where arc consistency is a decision procedure. We can directly make use of Proposition 10.5 of [5] which states that if a CFN P is VAC and $Bool(P)$ is in a class of CSPs for which arc consistency is a decision procedure, then P has an optimal solution of cost $w_\varnothing$.

Similarly, the network $Q = (T, \{z_T\} \cup_{i \in T} w_i)$ contains just one cost function with arity strictly above 1 and $Bool(Q)$ will be decided by arc consistency. Enforcing VAC will therefore provide a CFN which has also an optimal solution of cost $w_\varnothing$. The networks P and Q having optimal solutions of the same cost by definition of a decomposition, the result follows. □

4 Experimental Results

In this section, we intend to probe the possible interest of global cost function decompositions. These decompositions allow for a simple implementation, but it is also interesting to check if they can improve filtering performances or, at least, not degrade them too much.

We implemented the ternary encoding of the WeightedRegular cost function allowing to model a SoftRegular with Hamming distance. Since the monolithic propagator for the global SoftRegular cost function is already implemented in the WCSP solver `toulbar2` with associated (weak) E/FDGAC* filtering, this allows to compare the decomposition and monolithic versions.

Following [23], we generated random automata with $|Q|$ states and an alphabet size $|\Sigma|$. We randomly selected 30% of all possible pairs $(s, q_i) \in \Sigma \times Q$ and randomly chose a state $q_j \in Q$ to form a transition $\delta(s, q_i) = q_j$ for each such pair. The set of final states F is obtained by randomly selecting 50% of states in Q. All random samples use a uniform distribution.

From each automaton, we built two CFNs: one using a monolithic SoftRegular cost function using Hamming distance and another using the Berge-acyclic decomposition of the WeightedRegular cost function encoding the same cost function. To make the situation more realistic, we added to each of these problems the same set of random unary constraints, one per non-extra variable (with unary costs randomly chosen between 0 and 9). These problems have been solved using the CFN solver `toulbar2` (See `https://mulcyber.toulouse.inra.fr/projects/toulbar2`, version 0.9.4).

All preprocessing options of toulbar2 except filtering were turned off (option line `-o -e: -f: -dec: -h: -c: -d:`) and a DAC ordering compatible with the Berge-acyclic structure of the decomposition was used. The value ordering used chooses the existential EAC value first. The adaptive variable ordering heuristic is dom/wdeg. No initial upper bound is used. The same level of local consistency (namely (weak) EDGAC*, stronger than the T-DAC consistency considered in the paper and which therefore will also produce an optimal $w_\varnothing$ at the root) was used in all cases. We measured two times: (1) time for loading the problem and filtering the root node of the search tree and (2) total time for solving the CFN (including the previous time). The first time is informative on the filtering complexity while the second emphasizes the incrementality of the filtering algorithms. All the experiments were run on one 2.66 Ghz Intel Xeon CPU core with 64Gb RAM available with a time-limit of 5'. Times were averaged on 30 runs and samples reaching the time limit are considered as terminating in 5'. Table 1 shows the results.

Clearly, the decomposition brings an impressive progress in terms of efficiency. The results of these experiments should however be considered with care. The current implementation of the monolithic version of SoftRegular in `toulbar2` is probably far from optimized and, clearly, it has very limited incrementality. On the other hand, the core table filtering algorithms of `toulbar2` show excellent incrementality. These results could still be improved however: the ternary decomposition uses the same ternary cost function over and over in the problem. This is currently ignored by the solver. Taking this into account could considerably lower memory usage and file sizes.

It is also interesting but not trivial to compare the asymptotic complexities of the algorithms considered here. Enforcing EDGAC* on the decomposition is in $O(\max(nd, k).nd^3)$ [24].

n	$\|\Sigma\|$	$\|Q\|$	Monolithic		Decomposed	
			filter	solve	filter	solve
25	5	10	0.08	0.46	0.00	0.00
		20	0.20	1.17	0.01	0.01
		40	0.47	2.69	0.02	0.02
		80	1.63	9.55	0.08	0.08
25	10	10	0.42	2.36	0.00	0.00
		20	0.94	5.37	0.02	0.02
		40	2.31	13.36	0.06	0.06
		80	5.56	32.65	0.23	0.25
25	20	10	2.07	11.30	0.01	0.01
		20	5.13	27.95	0.04	0.04
		40	12.40	66.40	0.14	0.15
		80	30.34	164.7	0.51	0.55
50	5	10	0.30	3.39	0.00	0.00
		20	0.78	9.01	0.01	0.01
		40	2.41	28.58	0.04	0.05
		80	8.74	108.3	0.17	0.18
50	10	10	1.70	18.01	0.01	0.01
		20	3.86	41.58	0.03	0.04
		40	10.78	117.7	0.13	0.14
		80	260.2	260.2	0.48	0.53
50	20	10	8.44	84.88	0.03	0.03
		20	21.49	213.5	0.08	0.09
		40	300	300	0.29	0.31
		80	300	300	1.06	1.14

Table 1. Time in seconds to *filter* the CFN and to *solve* it to optimality. The CFNs encode a SoftRegular cost function representing the Hamming distance between a set of n variablesdefining a string and the language of a randomly generated DFA with $|\Sigma| = d$ symbols and $|Q|$ states using either the monolitic filtering algorithm of [18] or a ternary encoding of the equivalent WeightedRegular presented in Section 2.3, using table cost functions.

The asymptotic complexity for enforcing (weak) E/FDGAC* on the global SoftRegular cost function is not so simple to bound and it depends on the nature of the string distance and automaton used. For the Hamming distance and a deterministic automata, the underlying network has $O(|Q|.(n+1))$ vertices and $O(n(|\theta| + |Q|d)) = O(n.|Q|d)$ edges. Therefore, the complexity for just applying the successive shortest path minimum cost flow algorithm [4] on this network will be $O(|V|^2.|E|) = O(n^3.|Q|^3d)$ while the search for a shortest path using Bellman-Ford will be in $O(|V|.|E|) = O(n^2.|Q|^2d)$. According to [19,18], this means that the sole search for a full support will be in $O(n^3.|Q|^2.d(d+|Q|))$. Enforcing EDGAC* requires to simultaneously have simple, full and existential supports. Just considering full supports, this means that the complexity for enforcing EDGAC* exceeds $O(\max(nd, k).n^5.|Q|^2.d(d + |Q|))$. Assuming $\frac{|Q|}{|\Sigma|}$ is in $O(1)$, this is in $O(\max(nd, k).n^5.d^4)$ compared to the $O(\max(nd, k).nd^3)$ obtained through decomposition. This corroborates our experimental results but the strength of this comparison is largely conditionned by the tightness of the bounds presented in [19,18]. Another reason for the impressive speedups we observe may also lies in the non optimality of the global filtering algorithms for SoftRegular.

Conclusion

In this paper, we have extended the idea of constraint decomposition to cost functions occurring in CFNs. For cost functions having a Berge-acyclic decompositions, we have shown that even relatively simple filtering, at the directed arc consistency level, provide a comparable filtering when they are applied onto the decomposition or on the global cost function itself, provided a suitable variable ordering is used. For the stronger consistency Virtual AC filtering, the same result is obtained, without any requirement on variable ordering.

An example of decomposable cost function that has been considered in the paper is the WeightedRegular cost function. This cost function, based on weighted automata, allows to directly decompose the SoftRegular constraint using either the Hamming or the Edit distance[7].

The results presented in this paper are still preliminary. There are at least three directions that are worth exploring here. First, one should consider other decomposable cost functions beyond just WeightedRegular. Several related constraints, including ContextFreeGrammar, Among, . . . have Berge-acyclic decompositions and their cost functions variants likely have related Berge-acyclic decompositions, allowing for a simple extension of WCSP solvers to more extensive Cost Function Programming tools.

Beyond this, more experiments should be performed on more complex problems such as Nurse Rostering problems [20]. Finally, the strength and the simplicity of the result obtained for this local consistency plead for experiments using Virtual AC. This was not possible at the time this paper was written as the only implementations of VAC we know are restricted to binary cost functions.

Although restricted to Berge-acyclic decompositions, this work paves the way for a more general form of "structural decompositions" where global cost functions decompose into an acyclic structure of local cost functions combined with operators that could be different from $\oplus$, with bounded separator sizes (but not necessarily cardinality 1). For various operators, these global structurally decomposed cost functions could then be propagated efficiently through non serial dynamic programming (elimination) approaches.

References

1. Beeri, C., Fagin, R., Maier, D., M.Yannakakis: On the desirability of acyclic database schemes. Journal of the ACM 30, 479–513 (1983)
2. Bessiere, C., Hebrard, E., Hnich, B., Kiziltan, Z., Quimper, C., Walsh, T.: Reformulating global constraints: the slide and regular constraints. Abstraction, Reformulation, and Approximation pp. 80–92 (2007)
3. Boros, E., Hammer, P.: Pseudo-Boolean Optimization. Discrete Appl. Math. 123, 155–225 (2002)
4. Busacker, R., Gowen, P.: A procedure for determining minimal-cost network flow patterns. In: ORO Technical Report 15 (1961)
5. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence 174, 449–478 (2010)
6. Cooper, M.C.: Reduction operations in fuzzy or valued constraint satisfaction. Fuzzy Sets and Systems 134(3), 311–342 (2003)

[7] A result that could not be achieved in [2]. Note however that [13] also shows how the WeightedCFG soft constraint can encode Edit distance using ε-transitions.

7. Cooper, M.C., Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence 154(1-2), 199–227 (2004)
8. Demassey, S., Pesant, G., Rousseau, L.M.: A cost-regular based hybrid column generation approach. Constraints 11(4), 315–333 (2006)
9. Favier, A., de Givry, S., Legarra, A., Schiex, T.: Pairwise decomposition for combinatorial optimization in graphical models. In: Proc. of IJCAI'11. Barcelona, Spain (2011)
10. Gent, I.P. (ed.): Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings, Lecture Notes in Computer Science, vol. 5732. Springer (2009)
11. van Hoeve, W.J., Pesant, G., Rousseau, L.M.: On global warming: Flow-based soft global constraints. J. Heuristics 12(4-5), 347–373 (2006)
12. II, K.C., Kari, J.: Image compression using weighted finite automata. In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS. Lecture Notes in Computer Science, vol. 711, pp. 392–402. Springer (1993)
13. Katsirelos, G., Narodytska, N., Walsh, T.: The weighted grammar constraint. Annals OR 184(1), 179–207 (2011)
14. Koller, D., Friedman, N.: Probabilistic graphical models. MIT press (2009)
15. Lagerkvist, M.Z., Schulte, C.: Propagator groups. In: Gent [10], pp. 524–538
16. Larrosa, J., de Givry, S., Heras, F., Zytnicki, M.: Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In: Proc. of the 19^{th} IJCAI. pp. 84–89. Edinburgh, Scotland (Aug 2005)
17. Larrosa, J., Schiex, T.: Solving weighted CSP by maintaining arc consistency. Artif. Intell. 159(1-2), 1–26 (2004)
18. Lee, J., Leung, K.L.: A stronger consistency for soft global constraints in weighted constraint satisfaction. In: Fox, M., Poole, D. (eds.) AAAI. AAAI Press (2010)
19. Lee, J.H.M., Leung, K.L.: Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In: Boutilier, C. (ed.) IJCAI. pp. 559–565 (2009)
20. Métivier, J.P., Boizumault, P., Loudni, S.: Solving nurse rostering problems using soft global constraints. In: Gent [10], pp. 73–87
21. Mohri, M.: Edit-distance of weighted automata: General definitions and algorithms. International Journal of Foundations of Computer Science 14(6), 957–982 (2003)
22. Papadimitriou, C., Yannakakis, M.: Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43(3), 425–440 (1991)
23. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) CP. Lecture Notes in Computer Science, vol. 3258, pp. 482–495. Springer (2004)
24. Sánchez, M., de Givry, S., Schiex, T.: Mendelian error detection in complex pedigrees using weighted constraint satisfaction techniques. Constraints 13(1-2), 130–154 (2008)
25. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In: Proc. of the 14^{th} IJCAI. pp. 631–637. Montréal, Canada (Aug 1995)
26. Zytnicki, M., Gaspin, C., Schiex, T.: A new local consistency for weighted CSP dedicated to long domains. In: Proceedings of the 2006 ACM Symposium on Applied Computing (SAC). pp. 394–398. Dijon, France (Apr 2007)

Regularity Requirements in University Course Timetabling.

Broes De Cat, Christophe Machiels, Gerda Janssens and Marc Denecker
Department of Computer Science, K.U.Leuven, Belgium
{broes.decat, gerda.janssens, marc.denecker}@cs.kuleuven.be,
christophe.machiels@student.cs.kuleuven.be

K.U. Leuven

Abstract. Timetabling is the task of assigning sets of events to periods of time, taking into account resource-constraints and preferences among assignments. It is a well-studied field of research and is generally recognized to be a hard problem, both from the perspective of encoding it as from a computational point of view.
In recent years, there has been increased interest in combining efficient search algorithms with modelling languages capable of high-level representations of real-world domains. Research into such systems is conducted, a.o., in the fields of Constraint Programming and Knowledge Representation. In this paper, we investigate the use of the modelling language of first-order logic extended with constructs such as aggregates, types, definitions and arithmetic, and the IDP system, which implements model generation for this language, to the timetabling problem.
We show the feasibility of the approach and argue that there are important advantages both from the modelling point of view, leading to natural representation of the problem, and from the solving point of view, taking advantage of efficient automated search techniques like SAT and constraint propagation.

1 Introduction

Timetabling is a well-studied field of research and is generally known to be hard to solve, both from the perspective of encoding as from a computational point of view, belonging to the class of NP-hard constraint optimization problems. The problem consists of assigning time points to a set of events, taking into account constraints on resources associated with those events and preferences among assignments. University timetabling distinguishes between examination timetabling and course timetabling. In this paper we consider the latter.

Introductions to the basic elements of (automated) timetabling, approaches to solve the problem and surveys of course and examination timetabling can be found for example in [22]. Surveys of developments in timetabling can be found in [2,7,20,14]. Methods generally used to solve the problem are primarily based on three approaches: local search, e.g. [12], constraint logic programming, e.g. [21], and integer or mixed-integer programming, e.g. [23].

The requirements for course timetables can be quite different from one university to another, but usually the problem is *over-constrained.* A common approach is then to make a distinction between *hard* and *soft* constraints. The solution to the problem should satisfy all hard constraints and satisfy the soft constraints as much as possible. Therefore, soft constraints can also be seen as optimization objectives for the search algorithms.

In recent years, there has been increased interest in combining efficient search algorithms with modelling languages which allow high-level, natural representation of real-world applications. Research into such systems is conducted within e.g. the fields of Constraint Programming and Knowledge Representation.

At the Knowledge Representation and Reasoning group at the K.U. Leuven, our long-term goal is to build a *knowledge base system* (KBS) [9]. In such a system, knowledge about a domain of discourse is stored using a suitable logic and diverse tasks can be solved by applying various inference methods on that knowledge. Applied to the timetabling problem, such a KBS would contain the relevant knowledge about timetabling and the concrete data. By applying suitable forms of inference, diverse tasks can be accomplished: automatically generating timetables at the start of the year, automatic verification of handcrafted tables, revision of existing timetables as time progresses, . . . , which are all accomplished by reasoning on the same base of knowledge.

Our aim is to develop forms of inference for a KBS using FO(·) languages. Here FO(·) stands for the family of extensions of FO with modelling constructs useful for knowledge representation[1]. Examples are inductive definitions [8], aggregates, a type hierarchy and arithmetic. The IDP system is a step towards such a general system, implementing *model generation* for FO(·), a form of inference capable of generating models of a specification in FO(·).

The key features of a KBS are two-fold:

- The knowledge base language is truly *declarative*, being completely independent of any execution mechanism.
- The specific inference mechanisms are able to use the *latest techniques* in their respective fields, for example taking advantage of efficient automated search techniques like SAT and constraint propagation in model generation.

In this paper, we investigate the application of the IDP system, which supports an instance of FO(·), to solving the timetabling problem. The focus lies on solutions which are perceived to be *regular*: there should be some repeated pattern in the timetable over the course of weeks. In practice, non-regular schedules are almost never used as they complicate planning of other activities too much, both for students, teachers, rooms, . . .

The main contributions of this paper are, on the one hand, showing the applicability of soft constraints to handle regularity constraints in timetabling applications. On the other hand, demonstrating the value of high-level modelling

[1] ←Standard notation for extending FO with constructs of a class C is $FO(C)$. Well-known examples are FO(LFP) - FO extended with least-fixpoint constructs and FO(ID) - FO extended with an inductive definition construct.

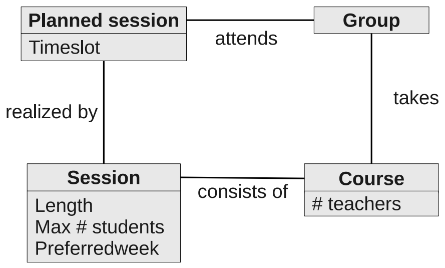


Fig. 1. A model illustrating the application domain of scheduling university courses.

and declarative problem solving in general and of the IDP system in particular for such applications. The paper is an extension of the work done in the master's thesis "Modelling of timetables" [15]. We also show experimental results by generating timetables for practicals for the first bachelor year of informatics students at the K.U. Leuven.

2 What Are the Timetable Requirements?

We will now introduce the basic concepts involved in course scheduling. An overview of the domain is presented in figure 1.

A *course* consists of a set of *sessions*, events with a specific content that make out (part of) the course's syllabus. A course is taught by a set of teachers (professors, assistants, ...) and taken by at least one *group* of students. A *time slot* is a period in time, characterized by a start and end point. A *timetable* consists of a set of *planned sessions*: instances of a session with an associated group and a time slot. For each session, the number of *planned sessions* depends on the number of teachers, the number of students taking the course, ...

This domain model will be used throught the rest of the paper. No mention was made of other relevant resource constraints, such as available rooms, available computers, ... Leaving them out at this stage is motivated by the well-known scheduling practice of dividing the problem in a sequence of increasingly concrete scheduling problems, where the result of the previous problem is used as input for the next. This reduces the complexity of the separate scheduling problems, but potentially leads to suboptimal solutions. The domain model also takes abstraction of the effective person teaching a certain class or which students are in which group. It is assumed that a teacher only teaches one course and students are in only one group.

Time slots are discretized by dividing a semester into a number of weeks and a day into a specified number of available slots (e.g. 8-9am, 9-10am, ...). A time slot is then represented by a week, a weekday and a start and end slot. A session has a number of (consecutive) slots it takes and a week in which it is be preferably taught. Sessions of the same course are also ordered.

Any valid timetable is subject to a set of hard and soft constraints. Our timetabling application is subject to the following hard constraints:

– A group cannot attend multiple planned sessions in the same time slot.
– For each combination of a group and a session of a course the group takes, one concrete session is planned (groups are assumed to be atomic).
– On holidays, no sessions are planned.
– The length of a session is taken into account.

The main focus in this paper is on the soft constraints involved in timetabling and specifically, generating timetables with a high degree of *regularity*. Informally, the table of some course is regular with respect to a group taking the course if its planned sessions take place on the same days and slots over the weeks and are taught to the same group of students. A complete timetable is then regular if it is regular for all courses and groups. Regularity is a very desirable property of timetables, for obvious reasons: both students and teachers can work along some fixed lines and can get to know each-other over time, improving cooperation. When naively generating timetables without enforcing such a constraint, it is almost never satisfied.

Next to regularity, the following other soft constraints are considered:

– Sessions should be planned in their preferred week. If this is not possible, planning them later than their preferred week is considered better than planning them early.
– Sessions should not be planned at noon or in the earliest and latest slots of each day. For Monday morning and Friday afternoon, this is even more important.
– Having contiguous periods of planned sessions for the same group is preferred to having free slots in between.

3 Reasoning on Logic Theories

3.1 FO(·)

For representing the knowledge about our problem domain in a declarative way, we use a logic based on full first-order logic, extended with constructs to make it better suited for representing knowledge in a natural way. In general, the class of those languages is be denoted by FO(·): they are based on FO, but extended by a number of additional language constructs. In this paper, we focus on a specific instantiation of FO(·), one which allows to express all concepts appearing in the timetabling domain in a natural fashion. This instantiation is the language which extends FO with inductive definitions, aggregate functions, partial functions, arithmetic and a hierarchical typing system (so it might be denoted FO(ID, Agg, Part, Arith, Type)). We will now introduce the language constructs relevant for the discussion at hand, for the remainder we refer the reader to [24]. Abusing notation, FO(·) will be used in the rest of the paper to refer to this instantiation whenever it is clear from the context.

A specification $< \Sigma, T, S >$ in FO(·) is a knowledge base consisting of the *vocabulary* Σ, the *theory* T and a *σ-structure* S for a subset σ of Σ. A vocabulary contains a hierarchy of types, a set of typed symbols (predicates and functions) and an infinite set of typed variables. A (partial) Σ-structure assigns interpretations to all types and a (partial) interpretation to the symbols of Σ.

Terms and formulas are defined by mutual recursion. A term is defined recursively as

- a (typed) variable is a term.
- if f is a function symbol and $t_1, \dots t_n$ are terms, then $f(t_1, \dots t_n)$ is a term.
- if agg is an *aggregate function symbol* and $\phi(\overline{x})$ is a formula with free variables $\overline{x}$, then $agg(\{\overline{x}|\phi(\overline{x})\})$ is a term.

An aggregate function symbol is interpreted by an *aggregate function*, a function mapping (multi)-sets of domain elements to a domain element. We consider the multi-set aggregates maximum, minimum, sum and count/cardinality, which all map sets of integers to respectively the maximum element, the minimum element, the sum and the number of elements of the set. These aggregate functions are denoted respectively as max, min, sum and $\#$.

A formula is defined recursively as

- if P is a predicate symbol and $t_1, \dots t_n$ are terms, then $P(t_1, \dots t_n)$ is a formula (denoted as an *atom*).
- if $\phi(x)$ and $\psi(y)$ are formulas, then applying the following (FO) connectives also yields formulas: $\forall x : \phi(x)$, $\exists x : \phi(x)$, $\phi(x) \land \psi(y)$, $\phi(x) \lor \psi(y)$, $\neg\phi(x)$.

A (possibly inductive) definition Δ consists of rules of the form $P(\overline{x}) \leftarrow \phi(\overline{x})$, with P a predicate symbol and $\phi(\overline{x})$ a logical formula. Inductive definitions follow evaluation according to the well-founded semantics [8].

An FO($\cdot$) theory consists of a set of formulas and a set of definitions Δ. A *model* of an FO($\cdot$) theory over a vocabulary Σ is a Σ-structure satisfying all formulas and all definitions.

In the rest of the paper, no recursively defined concepts are used, in which case a defined literal is true if and only if any of its rules are satisfied.

For any variable, its type is the type of the argument positions in which the variable appears [2]. The type of a quantified variable v can also be specified explicitly to be type t, denoted as $\forall v[t]$ or $\exists v[t]$.

Example 1. An example timetabling vocabulary might contain among others a group and a course type and a predicate linking students to the courses they take:

$$type\ group,\ type\ course,\ takes(group,\ course)$$

The theory can then express for example the constraint that each course has at least one group. This can be formalized in FO($\cdot$) by the formula

$$\forall\ c :\ \exists\ g : takes(g, c)$$

[2] ←The IDP system supports hierarchies of types and uses a type derivation system to infer the correct type of variables. Such hierarchies are not used in this paper.

The structure will among others contain an interpretation for the courses and the groups (both denoted here by their informal name) and the interpretation of the “takes” relationship:

$$\begin{aligned} group &= \{Bach1, Master2, \ldots\} \\ course &= \{Linear\ Algebra, Informatics, Organic\ Chemistry, \ldots\} \\ takes &= \{(Bach1, Informatics), (Master2, Organic\ Chemistry), \ldots\} \end{aligned}$$

3.2 Model Expansion

The IDP system implements one form of inference for FO($\cdot$), namely *finite domain model expansion* [18]. The task of model expansion is, given a specification $< \Sigma, T, S >$, with S a finite (partial) σ-structure ($\sigma \subseteq \Sigma$), to expand S to a model of T. Model expansion generalizes both model checking ($\sigma = \Sigma$) and model generation for a given finite domain (σ is empty). We denote the model expansion problem of expanding structure S to a model of theory T over Σ as $MX(< \Sigma, T, S >)$.

The task of *model minimization*, denoted $MM(< \Sigma, T, S >, O)$ is an extension of the model expansion task $MX(< \Sigma, T, S >)$. O represents a (pre-)order on the Σ-structures extending S. The task is then to find *optimal* models of T: models M such that no other model M' exists that $M' < M$ according to O.

Example 2. Consider the FO($\cdot$) specification $< \Sigma, T, S >$ and pre-order O shown below. Optimal models of this theory have a cost of 2, or equivalently, on each day exactly two planned sessions take place.

$$\begin{aligned} \Sigma &= \{type\ plannedsession,\ type\ day,\ on(plannedsession) : day,\ cost : int\} \\ T &= \{cost = max(\ \{\ \#(\{s | s \in plannedsession \wedge on(s) = d\})\ |\ d \in day\}\ ) \\ S &= \{plannedsession = \{1..10\},\ day = \{1..5\}, \ldots\} \end{aligned}$$

$$Order = M < M' \text{ iff } cost^{M} < cost^{M'}$$

The IDP system is a state-of-the-art model expansion system, as has been shown in the Answer Set Programming competition [10], where the task is to solve tasks by modelling them in a declarative framework and then applying automated inference techniques.

Model expansion inference in the IDP system is implemented by two main components:

- The *grounder* reduces an FO($\cdot$) theory to an equivalent propositional theory [24]. The grounder combines state-of-the-art grounding technology with a symbolic reasoning algorithm to derive which formulas of the propositional theory are certainly true or certainly false in models of that theory. This information is exploited to efficiently simplify the propositional theory while it is constructed.

- The *solver* [16] searches for models of the propositional theory generated by the grounder. The core of the solver is a SAT solver, i.e., a model generator for propositional logic. Currently, we use the SAT solver MiniSat [11]. Propagation techniques from CP and SAT-Modulo-Theories [19], as well as specialized algorithms [16], complement the SAT-based search to efficiently handle the additional language constructs like aggregates and inductive definitions.

Model minimization in IDP is implemented in a relatively standard way on top of model expansion with an *anytime branch-and-bound* algorithm. The minimization pre-order is specified as a logical term t and an order on the range type. Search is conducted in a *top-down* fashion by incrementally reducing the values t can still take. At the start of the search, no restrictions are imposed. When a model M is found with $t^M = n$, the constraint $t < n$ is added [3]. Another strategy would be to use bottom up search, starting with a minimal initial bound on t and incrementally loosening the bound when no model exists, but this has several drawbacks compared to the top-down strategy:

- Top-down search results in an anytime algorithm: at anytime during the search when the computation is interrupted, the last model found is the best one found. This is especially useful for timetabling problems as it is well-known that suboptimal solutions are often acceptable and the optimal solution often takes too long to find. With bottom-up search, the first model found is the optimal one.
- Experimental results have shown that performing a bottom-up search often converges slower to the optimal solution. One reason is that proving that no model exists is in many cases more expensive than finding a model.
- SAT-solvers infer additional constraints during search (implied by the theory but not explicitated) which further reduce the search space. Such learnt constraints remain trivially valid when the theory is incrementally strengthened (as in top-down search), but not when the theory is loosened (as in bottom-up search);

Considering the relation between modelling and performance, there are two main issues concerning the complexity of the IDP model expansion algorithm. Firstly, the size of the logical theory increases during the transformation into the ground format. This increase is exponential in the largest quantification depth of any formula in the theory. For example, given a formula $\forall x : \phi(x)$ or $\exists x : \phi(x)$, the grounding will contain n instantiations of $\phi(x)$, with n the domain size of the type of x. Consequently, the modelling approach should avoid using deep nesting of quantifications. Secondly, increasing the size of the domain, for example to increase the granularity of the time slots, increases both grounding size and solving time.

[3] ←Previous, less stringent constraints are removed.

4 Modelling the timetabling application

The approach we take to model timetabling is to cast it as an FO(·) model minimization problem $MM(< \Sigma, T, S >, O)$. Hard constraints are represented as logical formulas, the pre-order will reflect the influence of the soft constraints. We will then use IDP to solve the model minimization problem.

Before presenting how the constraints are modelled as a logical theory, we introduce the technique *reification* and how it applies here.

4.1 Modelling aspects

When generating a logical specification from a problem description, it is important to keep the number of variables in any constraint to a minimum. Next to reducing the size of the propositional theory (see previous section), it also increases the modularity, reuse and compactness of the specification. Consider for example the issue of modelling a time slot, which is identified by a week, a day, a start slot and an end slot. A predicate relating a planned session to its time slot might be modelled as $time(plannedsession, week, day, start, end)$, but using it would be awkward: we always have to quantify over all arguments. The technique *reification*, a well-known technique in knowledge representation, remedies this. Reification denotes the practice of creating an "artificial" identifier for a tuple of information. Then, the identifier can be used instead of the full tuple of information, effectively abstracting the information itself. This concept is well-known from a.o. database practice, where *keys* are used as identifiers to *records*, allowing to refer to any record by only using its key as opposed to the full record.

Example 3. Consider the constraint that the start slot has to precede the end slot. Using the predicate $time(plannedsession, week, day, start, end)$, this is expressed as:

$$\forall\ g\ w\ d\ s:\ time(ps, w, d, s, e) \Rightarrow s \leq e$$

Using reification, a new type t_{ps} is introduced which identifies a planned session. Its properties can be represented as binary symbols $group(t_{ps}) : group$, $start(t_{ps}) : slot$, ... The above-mentioned constraint is then expressed as

$$\forall\ s[t_{ps}] : start(s) \leq end(s)$$

There are several consequences to using reification. It increases *modularity* and *reuse* by providing an extra layer of abstraction. Also, it allows to *reduce* the size of logical formulas, by not having to quantify over spurious variables.

On the downside, an *upper bound* is necessary on the number of tuples that can be in the relationship. Otherwise the upper bound will equal the number of possible combinations of arguments and reification will not reduce the propositional theory. Secondly, as which identifier maps to which tuple is not defined (not knowing which tuples will be in the relationship) and as there might be too many identifiers (if the upper bound is an approximation), an important number of (additional) *symmetries* are introduced in the problem. Symmetry breaking constraints can be introduced to remedy this, by restricting which identifiers can map to which tuples, as shown in the following example.

Example 3. (continued) Given the previous reified planned session type t_{ps}, the number of symmetries can be (statically[4]) restricted by adding constraints:

$$\begin{aligned}
&\forall\ s_1[t_{ps}]\ s_2[t_{ps}] : s_1 < s_2 \Rightarrow \mathit{week}(s_1) \leq \mathit{week}(s_2)\\
&\forall\ s_1[t_{ps}]\ s_2[t_{ps}] : (s_1 < s_2 \wedge \mathit{week}(s_1) \leq \mathit{week}(s_2)) \Rightarrow \mathit{group}(s_1) \leq \mathit{group}(s_2)\\
&\qquad\qquad\qquad \ldots
\end{aligned}$$

If the upper bound is an approximation, a predicate $\mathit{used}(t_{pc})$ is introduced indicating which identifiers are effectively in use. Constraints of the form $\forall s : \phi(s)$ are then converted into $\forall s : \mathit{used}(s) \Rightarrow \phi(s)$. A similar transformation applies to existential quantification.

4.2 From specification to FO(·)

Applied to our setting, reification is used both for representing concepts from the domain and for all constraints. It can be applied to domain concepts (*course*, *group*, ...) that have an upper bound on the number of tuples in the relationship. For each such domain concept c, we create a type t_c for c (tuple identifiers). For each tuple of c known in advance, a unique identifier is created and its associations are modelled using that identifier. This set of identifiers is extended with *upperbound* unique identifiers, where *upperbound* is the maximum number of additional tuples in c. If *upperbound* is an approximation, any association which is a function has to be *partial* to allow for non-assigned identifiers.

Example 4. Consider the concept planned session, consisting of (among others) a week, day, start and end slot. Its association with a session is represented by the function $\mathit{sessionOf}(\mathit{week}, \mathit{day}, \mathit{start}, \mathit{end}) : \mathit{session}$. Assume that we know that $< \mathit{week}_{12}, \mathit{day}_2, \mathit{slot}_1, \mathit{slot}_2 >$ is a fixed planned session associated with session $\mathit{session}_5$. Furthermore, there will be at most 3 other planned sessions.

The concept planned session is reified into a type t_{ps}, consisting of the identifiers $\{a, b, c, d\}$ (1 known + 3 others). The properties are represented as the functions $\mathit{sessionOf}(t_{ps}) : \mathit{session}$, $\mathit{weekOf}(t_{ps}) : \mathit{week}$, $\mathit{dayOf}(t_{ps}) : \mathit{day}$, ... and the partial interpretation maps $\mathit{sessionOf}(a)$ to $\mathit{session}_5$, $\mathit{weekOf}(a)$ to week_{12}, $\mathit{dayOf}(a)$ to day_2, ...

Constraints often occur multiple times in the same specification, for example as sub-constraints of multiple, larger ones. Also, multiple timetabling tasks might differ only in which constraints they consider. In general, reuse of constraints should be easy. A modelling technique serving this purpose is to introduce additional symbols which represent whether a constraint is satisfied or not

[4] ←Symmetries can be broken *statically*, by adding extra constraints to the theory, or *dynamically*, by controlling the exploration of the search space by the search algorithm.

(or whether it should be satisfied or not). In the following, we demonstrate how to model hard and soft constraints and how to allow easy reuse.

A hard constraint can be seen as a condition which is required to be fulfilled. Using the created vocabulary, this condition is represented by a logical formula ϕ. An additional predicate c is introduced which represents the truth of ϕ. The hard constraint is then modelled by adding the rule $c \leftarrow \phi$ to the theory and the fact that c is true to the structure. This allows to switch the constraint on or off depending to solve various tasks by only changing the structure. Also, c can be used as an abstraction for the constraint itself in other constraints.

Example 5. The hard constraint that no sessions should be planned on a holiday, is represented as follows (the predicate $holiday(week, day)$ represents which days are holidays).

$$keepHolidaysFree \leftarrow \forall\ s[t_{ps}] : \neg holiday(weekOf(s), dayOf(s))$$

The associated structure asserts the reified atom: $keepHolidaysFree = \mathbf{true}$.

Any soft constraint can be seen as a condition and a cost for violating the condition (*cost*). Again, an additional predicate c is introduced which represents the truth of the condition. But instead of adding c to the structure, a *cost function* f_c is created which is defined to be 0 if the condition holds and equal to *cost* otherwise. Here, c is used to represent the condition, as shown in the next example.

Example 6. For each planned session, the cost is 1 if it is not planned in the preferred week of its session. This can be represented as:

$$\forall s :\ inPrefferedWeek(s) \leftarrow weekOf(s) = preferredWeekOf(sessionOf(s))$$
$$\forall s :\ cost_{inPrefferedWeek}(s) = 0 \Leftrightarrow inPrefferedWeek(s)$$
$$\forall s :\ cost_{inPrefferedWeek}(s) = 1 \Leftrightarrow \neg inPrefferedWeek(s)$$

As a last part of creating the logical specification, a constant $cost : int$ is created which represents the cost of the violation of all soft constraints. It is defined as the sum of the cost functions of all the soft constraints, over all their domain elements. For example, two cost functions, f_1 and f_2, with arities 1 and 2 respectively, result in the following global cost function:

$$cost = sum(\{y \mid \forall\ x :\ y = f_1(x)\} \cup \{z \mid \forall\ x\ y :\ z = f_2(x, y)\})$$

This cost is used as the pre-order on the interpretations and allows to minimize of the violation of the soft constraints.

In the next section we will discuss how to represent the constraint on the regularity of schedules. The other soft constraints are not discussed in detail, as they follow straightforwardly from the presented ideas.

4.3 Regularity Requirements

It turned out that formalizing the concept of regularity was quite difficult. The basic case is straightforward: if all sessions of a specific course for some group of students always take place on Thursday at 2pm, this is clearly regular. What about a timetable organising 2 sessions in odd weeks and 1 in even weeks? This probably also qualifies as quite regular. To decide whether a timetable is regular, we introduce the concept of *regularity patterns* (in short, denoted as patterns), which define which assignments of sessions to time slots qualify as regular. Such a pattern would consist of a set of rules that define which slots can be assigned to which sessions as a function of the weeks. The regularity pattern in the first example is the pattern which maps all sessions of the course to the slot on Thursdays at 2pm. In the second, it is the pattern which maps sessions n and $n + 1$ to an odd week week and $n + 2$ to the next (even) week.

Clearly, many such regular patterns are possible, leading to a three-step approach to generate regular timetables: *pattern selection* - *pattern instantiation* - *timetable generation*. Firstly, a subset of allowed patterns is fixed (*selection*). An example selection might be to only allow patterns which assign the same day and slot to a group and course combination, independent of the weeks. Secondly, a model minimization problem is solved that *instantiates* one pattern for each group and course combination. This comes down to choosing the effective slot(s) in which the group will preferably take sessions of the course. If (according to the pattern) each group and course are be assigned exactly one day and slot, we will denote this as the *default* session for that group and course. The final step then consists of *generating* the complete timetable. As a measure of regularity of the timetables, a cost is assigned to the "distance" between a timetable and the instantiated patterns.

In this paper, the allowed patterns are patterns which assign the same day and slot to a group and course combination, independent of the weeks. An obvious disadvantage to this approach is that a bad instantiation will lead to suboptimal timetables. This happens for example if there are more sessions to a course than weeks in the semester (no low-cost timetables will exist then). Experiments in this direction are part of future work, but a better pattern selection might be to guess how many default slots will be necessary for a course.

In the pattern instantiation step, the following hard constraints are used in the model minimization:

- No sessions are planned across noon.
- The number of planned sessions on the same timeslot cannot exceed the number of available teachers.
- No planned sessions should coincide for the same group.
- The session length is taken into account.

A new cost function is introduced for this minimization problem which reflects how *busy* a specific day and slot will be across the term. It maps a day and slot combination to a linear combination of

– The number of sessions of each course of which the default session is planned on that day and slot.
– The number of holidays on that day.
– Whether the slot is preferred (recall, e.g. Monday morning was not).

The total cost is then defined as a function of the costs of each day and slot. Preferably, the cost of a busy day should weigh more on the total cost than the cost of several less busy days (there is a preference of spreading the work over a week). As the IDP system cannot handle non-linear functions efficiently (modelling the total cost as e.g. a quadratic summation might seem appropriate), the total cost was defined as the maximum of the costs of each day and slot.

5 Experiments

Experiments were conducted by generating timetables for the scheduling of one term (13 weeks) of courses for the first bachelor year in informatics, 2010-2011 at the Katholieke Universiteit of Leuven. Courses consist of a series of lectures and a series of practicals. The dates of lectures are known in advance (they are scheduled by another university division) and were used as partial interpretation of the planned sessions. The lecture timetable is quite regular, except for some special cases like holidays. Each problem instance has 3 groups of students, which all take the same courses, and 2 teachers for each course.

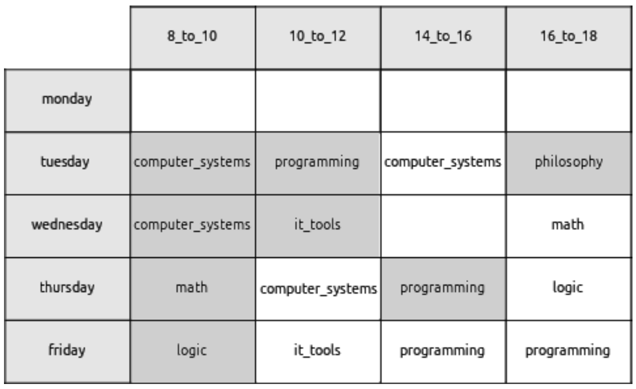
From this dataset, we conducted several experiments [5], by varying both the number of groups to schedule and the length of the term.

The time to instantiate the patterns is orders of magnitude below the time to find the full schedule (as expected). In figure 2, such an optimal default pattern is shown for one group of students. As expected, there are no overlapping default sessions, as there is room enough in between the lectures.

Originally, the symmetry breaking constraints, introduced in 4, were included in the experiments. Such constraints cause a blow-up in the propositional theory for large domains, as each domain element is related to all other domain elements. This blow-up effectively forced us to disable the symmetry breaking constraints for the larger data sets. As symmetry breaking constraints are important in efficiently proving that no better solutions exist, optimality for those data sets was never proven within the imposed time bounds. On the other hand, the last (suboptimal) solution found before timeout, even on those data sets, proved to be sufficiently regular for practical use.

Table 1 shows the total time it took to find an optimal timetable, including pattern instantiation and grounding. A timeout of 30 minutes was used. All cases in which the optimum was found were very regular schedules, indicating that the soft constraints were chosen quite well.

[5] ←The experiments were conducted using version 2.17 of the IDP system, which can be found on *http://dtai.cs.kuleuven.be/krr/software/download*. All datasets and the experimental setup can be found on *http://dtai.cs.kuleuven.be/krr/research/experiments*. Experiments were conducted on an Ubuntu 11.04 system, with 4Gb of RAM and a dual-core 2.53 GHz processor



	8_to_10	10_to_12	14_to_16	16_to_18
monday				
tuesday	computer_systems	programming	computer_systems	philosophy
wednesday	computer_systems	it_tools		math
thursday	math	computer_systems	programming	logic
friday	logic	it_tools	programming	programming

Fig. 2. A one-week timetable containing the generated default sessions for one group of the data set. Vertical axis denotes the days of the week, horizontal axis the time slots available each day. The grey positions denote the moments that a lecture takes place and white, labelled positions indicate the default time slot for the associated practicals.

# weeks	1 group	2 groups	3 groups	# weeks	1 group	2 groups	3 groups
1	0.1	1.1	5	8	15*	641*	###*
2	0.7	3.4	22	9	28*	1342*	###*
3	1.5	8.4	174	10	48*	1875*	###*
4	3.7	29	###	11	61*	###*	###*
5	3.1	54	###	12	88*	###*	###*
6	6.8*	180*	###*	13	130*	###*	###*
7	10*	285*	###*				

Table 1. The time taken to find an optimal timetable as a function of the number of groups and the length of the term. Disabled symmetry breaking constraints are indicated by $*$, ### indicates that optimality was not proven within the time bound.

6 Related and Future Work

Researchers from several communities such as operations research, metaheuristics [14] and constraint programming have been trying for some time to find general, workable solutions for timetabling applications.

Within the specific application of course timetabling, a number of general problem formulations have been proposed in [5], providing a generic way to compare timetabling systems. As part of future work, we will model those in FO($\cdot$) and compare the performance of IDP with state-of-the-art systems.

The link with graph colouring has inspired a lot of approximation algorithms that use metaheuristics to find "good enough" solutions, including genetic algorithms, tabu search and simulated annealing. Another line of research uses mixed integer programming and optimizes an objective function [23].

Also constraint programming has been used to solve timetabling problems: constraints model the knowledge and solutions are found by interleaving constraint propagation and search. Within the field of SAT, SAT technology has recently been used to tackle timetabling problems by casting them as maximum satisfiability problems and using max-SAT technology to find good solutions, see e.g. [1]. The approach taken in this paper is a hybrid of CP and SAT technology, by using a SAT-solver as backend, but using specialised propagation mechanisms for numeric constraints like sum and cardinality and for inductive definitions.

Within the constraint programming community, the treatment of soft constraints is supported by the semiring-based formalism [3,4]. The formalism extends the classical constraint notion by adding the concept of a structure representing the levels of satisfiability of the constraints.

One future task is to compare our approach with other high-level modelling languages and their associated systems, both in terms of ease of modelling and performance. Related languages are Zinc [17], based on Constraint Programming methodologies and Answer Set Programming [13], based on logic programs.

Furthermore, we will be looking at other instances of course timetabling, for example the master years of informatics, in which the students are able to choose courses on an individual basis (which increases the problem complexity). We also intend to deploy the system at the K.U. Leuven in the long run, after solving the scalability issues related to the intricate interleaving of the curricula.

7 Conclusion

In this paper we present an approach to solving the university timetabling problem, taking regularity constraints into account. The approach was based on modelling the problem in a declarative, high-level language and using an automatic reasoning mechanism to find solutions.

Using FO(·) as a modelling language proved to be adequate, allowing for a short (few hundred lines of code), adaptable and more importantly, natural representation of the problem, even in the presence of soft constraints. The experiments show that using the IDP system and its model minimization inference are capable of tackling the problem for practically usable problem sizes, after applying a number of optimizations to reduce the search space.

References

1. Roberto Asín Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. pages 42–56, 2010.
2. Armen S. Asratian and Dominique de Werra. A generalized class-teacher model for some timetabling problems. *European Journal of Operational Research*, 143(3):531–542, 2002.
3. Stefano Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *Lecture Notes in Computer Science*. Springer, 2004.
4. Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44:201–236, March 1997.
5. Alex Bonutti, Fabio De Cesco, Luca Di Gaspero, and Andrea Schaerf. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals of Operations Research*, pages 1–12, 2010.

6. Edmund K. Burke and Patrick De Causmaecker, editors. *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002, Gent, Belgium, August 21-23, 2002, Selected Revised Papers*, volume 2740 of *Lecture Notes in Computer Science*. Springer, 2003.
7. Edmund K. Burke and Sanja Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*, 140(2):266–280, 2002.
8. Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL)*, 9(2):Article 14, 2008.
9. Marc Denecker and Joost Vennekens. Building a knowledge base system for an integration of logic programming and classical logic. volume 5366 of *LNCS*, pages 71–76. Springer, 2008.
10. Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Miroslaw Truszczynski. The second answer set programming competition. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *LPNMR*, volume 5753 of *Lecture Notes in Computer Science*, pages 637–654. Springer, 2009.
11. Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
12. Luca Di Gaspero and Andrea Schaerf. Multi-neighbourhood local search with application to course timetabling. In Burke and Causmaecker [6], pages 262–275.
13. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.
14. Rhydian Lewis. A survey of metaheuristic-based techniques for university timetabling problems. *Or Spektrum*, 30:167–190, 2007.
15. Christophe Machiels. Modelleren van lessenroosters. Available at http://dtai.cs.kuleuven.be/krr/research/publications, 2011.
16. Maarten Mariën. *Model Generation for ID-Logic*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, February 2009.
17. Kim Marriott, Nicholas Nethercote, Reza Rafeh, Peter J. Stuckey, Maria Garcia de la Banda, and Mark Wallace. The design of the zinc modelling language. *Constraints*, 13(3):229–267, 2008.
18. David G. Mitchell and Eugenia Ternovska. A framework for representing and solving NP search problems. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 430–435. AAAI Press / The MIT Press, 2005.
19. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
20. Sanja Petrovic and Edmund Burke. University timetabling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis, chapter 45*. Chapman Hall/CRC Press, 2004.
21. Hana Rudová and Keith S. Murray. University course timetabling with soft constraints. In Burke and Causmaecker [6], pages 310–328.
22. Andrea Schaerf. A survey of automated timetabling. *Artif. Intell. Rev.*, 13(2):87–127, 1999.
23. Katja Schimmelpfeng and Stefan Helber. Application of a real-world university-course timetabling model solved by integer programming. *OR Spectrum*, 29:783–803, 2007. 10.1007/s00291-006-0074-z.
24. Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *Journal of Artificial Intelligence Research*, 38:223–269, 2010.

Heuristic Search for m Best Solutions with Applications to Graphical Models

Rina Dechter and Natalia Flerova

Bren school of Information and Computer Sciences
University of California Irvine

Abstract. The paper focuses on finding the m best solutions to a combinatorial optimization problems using Best-First or Branch-and-Bound search. We are interested in graphical model optimization tasks (e.g., Weighted CSP), which can be formulated as finding the m-best solution-paths in a weighted search graph. Specifically, we present m-A*, extending the well-known A* to the m-best problem, and prove that all A*'s properties are maintained, including soundness and completeness of *m-A**, dominance with respect to improved heuristics and most significantly optimal efficiency compared with any other search algorithm that use the same heuristic function. We also present and analyse *m-B&B*, an extension of a Depth First Branch and Bound algorithm to the task of finding the m best solutions. Finally, for graphical models, a hybrid of A* and a variable-elimination scheme yields an algorithm which has the best complexity bound compared with earlier known m-best algorithms.

1 Introduction

Depth-First Branch and Bound (*B&B*) and Best-First Search (BFS) are the most widely used search schemes for finding optimal solutions. In this paper we explore the extension of such search algorithms to finding the m-best optimal solutions. We apply such algorithms to optimization tasks over graphical models, such as weighted CSPs and most probable explanation (MPE) over probabilistic networks, arising in many applications, e.g, in procurement auction problems, biological sequence alignment and finding m most likely haplotype configurations.

Most of the paper's analysis focuses on Best-First Search whose behavior for the task of finding a single optimal solution is well understood. The algorithm is known to be sound and complete when guided by an admissible (i.e., lower bound for minimization task) heuristic evaluation function. Most significantly, it is efficiently optimal: any node it expands must be expanded by any other exact search algorithm having the same heuristic function, if both use the same tie breaking rule [4]. Best-First Search, and its most famous variant A*, are also known to require significant memory. The most popular alternative to BFS is Depth-First Branch and Bound, whose most attractive feature compared with BFS is that it can be executed with linear memory. Yet, when the search space is a graph, it can exploit memory to improve its performance, by flexibly trading space and time.

Highly efficient *B&B* and BFS algorithms for finding an optimal solution over graphical models were developed recently. These algorithms explore the AND/OR search-tree or the context-minimal AND/OR search graph of the graphical model [3], they use heuristic evaluation functions generated either by the mini-bucket scheme or through soft arc-consistency schemes [9, 12] and they proved to be most effective as demonstrated in recent evaluations [2]. Clearly, an extension of a BFS or *B&B* to the m-best task over a general search space graph for optimal path-finding tasks is applicable to graphical models when searching its AND/OR search space [3].

The main contribution of this paper (Section 3) is in presenting m-A*, an extension of the A* for finding the m-best solutions, and in showing that all its properties extend to the m-best case. In particular we prove that m-A* is sound and complete and is efficiently optimal. In Section 4 we discuss extensions of Branch and Bound to the m-best task. Subsequently, in Section 5, we discuss anchoring the algorithms for searching graphical models' AND/OR search spaces. We show that a hybrid of A* and a variable-elimination scheme, denoted BE-Greedy-m-BF, yields a best complexity algorithm compared with earlier work on graphical models. Preliminary empirical evaluation shed light on the algorithm's performance. We assume without loss of generality that the optimization task at hand is minimization.

2 Background

Let A be a general search algorithm for finding an optimal solution path over a search space defined implicitly by a set of states (the nodes in the graph), operators that map states to states having costs or weights (the directed weighted arcs), a starting state n_0 and a set of goal states. The task is to find the least cost solution path from the start node to a goal [10] where the cost of a solution path is the sum of the weights or the product of the weights on its arcs. The two primary search strategies for finding an optimal solution are Best-First Search (*BFS*), (e.g., A*) and Depth-First Branch-and-Bound search (*B&B*). In this paper we explore extensions of both schemes for finding the m-best solutions for any integer m. Best-first search (*BFS*) seems to be the most suitable algorithm to be extended to m-best task. It explores the search space using a heuristic evaluation function $f(n)$ which for every node n estimates the best cost solution path passing through n. The algorithm maintains a list of nodes that are candidates for expansions (often called "OPEN", or "frontier"). At each iteration a node in the frontier having a minimal f is selected for expansion, moved to another list (called "CLOSED", or "explored set") and its child nodes are places in OPEN, each associated with its own evaluation function. When a goal node is selected for expansion the algorithm terminates and outputs the solution found.

It is known that when $f(n)$ is a lower bound on the optimal cost path that goes through n the algorithm terminates with an optimal solution. The algorithm's properties were extensively studied [10, 11]. In particular, (up to some tie breaking rule) the algorithm is efficiently optimal. Namely, any node expanded by BFS is expanded by any other search algorithm guaranteed to find an optimal solution [4]. If provided with a consistent (also called monotonic) heuristic, the algorithm expands nodes in frontiers of monotonically increasing evaluation function and it expands every node just once [10]. This later property is of utmost importance because it frees the algorithm from the need to check for duplicates when the search space is a graph.

We focus on the well-known BFS variant called A^*, where the heuristic evaluation function is expressed as the sum $f(n) = g(n) + h(n)$ in which for any node n $g(n)$ is minimal cost from the root n_0 to n along the current path, and $h(n)$ underestimates $h^*(n)$, the optimal cost from n to a goal node. The implicit directed search space graph is $G = (N, E)$. We denote by $g_\pi(n)$ the cost from the root to n along path π and by $c_\pi(n_1, n_2)$ the cost from n_1 to n_2 along π. The heuristic function h is *consistent* iff $\forall$ n' successor of n in G, $h(n) \leq c(n, n') + h(n')$.

3 Best-first Search For the Best m Solutions

As was noted in [1], the extension of BFS algorithms, including A^*, to the m-best problem seems simple: rather then terminate with the first solution found, the algorithm continues searching until it generates m solutions. As we will show, these solutions are the m best ones. Specifically, the second solution found is the second optimal solution, the third is the third optimal one and so on. In the following subsections we present an extension of A^* to finding the m-best solutions and show that most of A^*'s nice properties are maintained in m-A^*.

3.1 Algorithm m-A*

Figure 1 provides a high level description of a tree-search variant which we call m-A*. The algorithm expands nodes in order of increasing value of f in the usual A^* manner. For simplicity we specify the algorithm under the assumption that h is consistent. The algorithm maintains separate paths to each copy of a node in the explored search tree, denoted by Tr. As we will show, this redundancy is not wasteful when the heuristic function is consistent.

We denote by C^*_i the i^{th} best solution cost, by $f^*_i(n)$ the cost of the i^{th} best solution going through node n, by $f_i(n)$ the evaluation function estimating $f^*_i(n)$ and by $g_i(n)$ and $h_i(n)$ the estimates of the i^{th} best costs from n_0 to n and from n to a goal, respectively. If the heuristic function is not consistent, step 7 should be revised to account for the possibility that new paths to n' may be encountered and expanded in a non-monotone cost order. We therefore should revise as follows:
7a. for any node n' that appears already more than m times in the union of OPEN or CLOSED, if $g(n')$ is strictly smaller than $g_m(n')$, the current m-best path to n', keep n' with a pointer to n and discard the earlier pointer.

```
Algorithm m-A*
Input: An implicit directed search graph $G = (N, E)$, with a start node $n_0$ and a set
of goal nodes $Goals$. A consistent heuristic evaluation function $h(n)$, parameter $m$,
OPEN=$\emptyset$. A tree $Tr$ which is initially empty.
Output: the m-best solutions.

 1. i =1 (i counts the current solution being searched for).
 2. Put the start node $n_0$ in OPEN, $f(n_0) = h(n_0)$. Make $n_0$ the root of $Tr$.
 3. If OPEN is empty, exit and return the solutions found so far.
 4. Remove a node, denoted $n$, in OPEN having a minimum $f$ (break ties arbitrarily,
    but in favor of goal nodes and deeper nodes) and put it in CLOSED.
 5. If $n$ is a goal node, output the current solution obtained by tracing back pointers
    from $n$ to $n_0$ (pointers are assigned in the following step). Denote this solution as
    $Sol_i$. If $i = m$ exit. else $i \leftarrow i + 1$, and go to step 3.
 6. Otherwise expand $n$, generating all its child nodes $Ch$. Compute $g(n') = g(n) +
    c(n, n')$ and $f(n') = g(n') + h(n')$, $\forall n' \in Ch$.
 7. If $n'$ appears in OPEN or CLOSED $m$ times, discard node $n'$, otherwise attach
    from each $n'$ in $Ch$ a pointer back to $n$ in $Tr$. Insert $n'$ into the right place in
    OPEN based on its $f(n')$.
 8. Go to step 3.
```

Fig. 1: Algorithm m-A*

Example 1. Consider example in Figure 2. We assume the task of finding the two shortest paths from node A to node G. Assuming the following heuristic values: h(A)=5, h(B)=4, h(C)=3, h(D)=2, h(E)=1, h(F)=1, h(G)=0, the cost of the best solution path $\{A, C, D, F, G\}$ is cost 6, the cost of the second best solution $\{A, B, D, F, G\}$ is cost 9. On the trace of the search tree (right) orange nodes were expanded and put on CLOSED, blue nodes remain on OPEN, magenta nodes are the goals. Nodes D, F, E and G are expanded twice and two duplicates of each of these nodes are retained.

Based on the known properties of A^* [10] we will establish the following corresponding properties of *m-A**:

1. Soundness and completeness: *m-A** terminates with the m-best solutions generated in order of their costs.
2. Optimal efficiency: *m-A** is optimally efficient in terms of node expansions, compared with any other search algorithm that is guaranteed to find the m-best solutions. Namely, any node that is surely expanded by *m-A** must be expanded by any other sound and complete algorithm.
3. Optimal efficiency for consistent heuristics: *m-A** is optimally efficient in terms of *number of node expansions* when the heuristic function is consistent. In this case, *m-A** expands each node at most m times.
4. Dominance: Given two heuristic functions h_1 and h_2, s.t. for every n $h_1(n) < h_2(n)$, $m\text{-}A^*_1$ will expand every node surely expanded by $m\text{-}A^*_2$, when $m\text{-}A^*_i$ is using heuristic h_i.

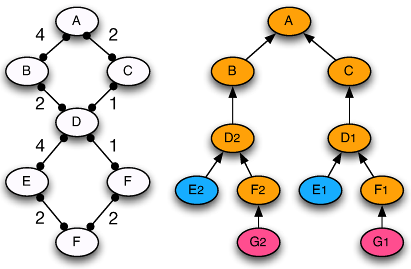


Fig. 2: The search graph of the problem (left) and the trace of the m-A^* for m=2. Node A is the start node, node G is the goal node. Orange represents the nodes that were expanded and put on CLOSED, blue nodes were generated, but never expanded (remain on OPEN list), magenta nodes are the goal nodes.

3.2 *m-A** is Sound and Complete

We know, that, if provided with an admissible heuristic, A^* will surely expand any node n, whose evaluation function f_π along a given path π from n_0 to n is strictly lower than the cost of the optimal solution, C^*. Namely, it surely expands every node n', such that $n' \in \pi_{n_0..n}$ and $f(n') < C^*$. We next show that this property can be extended straightforwardly to the m-best case.

Let's denote by SS_i the set of nodes expanded by *m-A** just before a goal node of the i^{th}-best solution was selected for expansion. By definition $SS_1 \subset SS_2, ... \subset SS_i, .. \subset SS_m$ and $C_1^* \leq C_2^*, ... \leq C_m^*$.

Since we maintain at most m live copies of a node in OPEN or CLOSED we must be sure that we never discard any of the viable m-best solution paths. Bounding the number of copies of a node is important for complexity reasons.

Proposition 1. *At any point before the algorithm generates the i^{th}-best solution there is always a node in OPEN along each of the j^{th}-best solution path for $j \in [i, ..., m]$.*

From the above proposition it follows that,

Proposition 2. *At any time before m-A* expands a goal node of the i^{th} best solution path, there is a node n' in OPEN satisfying $f(n') \leq C_i^*$.*

Proof. Assume that the search graph G has at least m solutions. Let's assume that all the $i-1$ best solutions were already generated. Namely, the goal nodes of the first $i-1$ solution paths were selected for expansion. At any time after that, there is a node n in OPEN that resides on an i^{th} best path π_i, because the i^{th}-best path is not discarded (Proposition 1). Let n' be the first OPEN node on π_i. Its evaluation function is, by definition, $f(n') = g_{\pi_i}(n') + h(n')$. Since π_i is an i^{th} best path, $g_{\pi_i}(n') + c_{\pi_i}(n', t) = C_i^*$, when t is the goal for path π_i. Since by definition $h(n') \leq c_\pi(n', t)$, we get that n' is in OPEN and $f(n') \leq C_i^*$.

We can conclude:

Theorem 1 (sound and completeness). *Algorithm m-A* generates the m-best solutions in order, namely, the i^{th} solution generated is the i^{th} best solution.*

Proof. By induction. We know that the first solution generated is the optimal one and, assuming that the first $i-1$ solutions generated are $i-1$ best in sequence, we will prove that the i^{th} one generated is the i^{th} best. If not, the i^{th} generated solution path, denoted by π' has a cost c and $c > C^*_i$. However, when the algorithm selected the goal t' along π', its evaluation function was $f(t') = g_{\pi'}(t') = c$, while there was a node in OPEN whose evaluation function is C^*_i or smaller that should have been selected. Contradiction.

3.3 *m-A** is Optimally Efficient

Algorithm A* is known to be optimally efficient [4]. Namely, any other algorithm that extends search paths from the root and uses the same heuristic information will expand every node that is surely expanded by A^*, i.e., $\forall\ n$, such that $f(n) < C^*$. This property can be extended to our *m-A** as follows:

Theorem 2 (m-optimal efficiency). *Any search algorithm which is guaranteed to find the m-best solutions, and which explores the same search graph as m-A* will have to expand any node that is surely expanded by* m-A**, if it uses the same heuristic function. Formally, it will have to expand every node n that lies on a path $\pi_{0..n}$ that is dominated by C^*_m, namely s.t., $f(n') < C^*_m\ \forall n' \in \pi_{0..n}$.*

Similarly to [4] we can show that any algorithm that does not expand a node n lying on a path $\pi_{0..n}$, whose evaluation function is dominated by C^*_m, (namely $\forall\ n'\ f(n') < C^*_m$), can miss one of the m-best solutions when applied to a slightly different problem, and therefore contradicts completeness.

Proof. Consider a problem having the search graph G and consistent heuristic h. Assume that node n is surely expanded by *m-A**, namely for some $j \le m$ node n lies on a path π dominated by C^*_j. Let B be an algorithm that uses the same heuristic h and is guaranteed to find the m best solutions. Assume that B does not expand n. We can create a new problem graph G' (Figure 3) by adding a new goal node t with $h(t) = 0$, connecting it to n by an edge having cost $c = h(n) + \delta$, where $\delta = 0.5(C^*_j - D)$, in where $D = \underset{n' \in E_j}{max} f(n')$, E_j is the set of nodes surely expanded by *m-A** before finding the j^{th} solution.

It is possible to show that the heuristic h is also admissible for the graph G' [4]. Since $\delta = 0.5(C^*_j - D)$, $C* = D - 2\delta$. By construction, the evaluation function of the new goal node is $f(t) = g(t) + h(t) = g(n) + c = g(n) + h(n) + \delta = f(n) + \delta \le D + \delta = C^*_j - \delta < C^*_j$. Since *m-A** will surely expand all nodes with $f(n') < C^*_j$ before finding the j^{th} solution, it will expand node t and discover the solution whose cost is bounded by $C^*_j - \delta$. On the other hand, algorithm B, that does not expand node n in the original problem, will be unable to reach node t and will only discover the solution with cost C^*_j, thus not returning the true set of m best solutions to the modified problem. Contradiction.

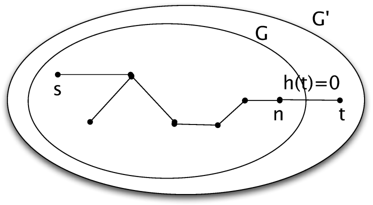


Fig. 3: The graph G' represents a new problem instance constructed by appending to node n a branch leading to a new goal node t.

3.4 *m-A** for Consistent Heuristics

If the heuristic function is consistent, whenever a node n is selected for expansion (for the first time) by A^* the algorithm had already found the shortest path to that node. We can extend this property as follows:

Theorem 3. *Given a consistent h, when* m-A* *selects a node n for expansion for the i^{th} time, then $g(n) = g^*_i(n)$, namely it has found the i^{th} best solution from s to n.*

Proof. By induction. For $i = 1$ the theorem holds [10]. Assume that it also holds for $i = (j-1)$. Let us consider the j^{th} expansion of n. Since we have already expanded n $(j-1)$ times and since by the induction hypothesis we have already found the $(j-1)$ distinct best paths to the node n, there can be two cases: either the cost of the j^{th} path to n is equal to the j^{th} best, i.e., $g(n) = g^*_i(n)$, or it is greater i.e., $g(n) = g^*_k(n), k > i$. If we assume the latter, then there exists some other path π from n_0 to n with cost $g_\pi(n) = g^*_j(n) < g_k(n)$ that has not been traversed yet. Since $f_\pi(n) = g_\pi(n) + h(n)$ and $f(n) = g(n) + h(n)$, it follows that $f_\pi(n) < f(n)$. Let n' be the latest node on π on OPEN, i.e., a node already generated but not yet expanded. It is known, that if the heuristic function is consistent, the values of f along any given path are non-decreasing and therefore $\forall\ n'$ on π, $f_\pi(n') \leq f_\pi(n) < f(n)$ and n' should have been expanded before node n leading to contradiction.

We can conclude that when h is consistent any node n will be expanded at most m times.

Corollary 1. *Given m-A* with a consistent h*

- *The maximum number of copies of the same node in OPEN or CLOSED can be bounded by m.*
- *The set $\{n|f(n) < C^*_m\}$ will surely be expanded (no need of dominance along the path).*

3.5 The Impact of m on the Expanded Search Space

The relation between the sizes of search space explored by m-A* for different levels of m is obviously monotonically increasing with m. We can provide the following characterization for the respective search spaces.

Proposition 3. *Given a search graph,*

1. *Any node expanded by i-A* is expanded by j-A* if $i < j$ and if both use the same tie-breaking rule.*
2. *The set $S(i,j)$ of nodes defined by $S(i,j) = \{n|C_i^* < f(n) < C_j^*\}$ will surely be expanded by j-A* and surely not expanded by i-A*.*
3. *If $C_i = C_i^*$, the number of nodes expanded is determined by the tie-breaking rule.*

As a result, the larger the discrepancy between the respective costs $C_j^* - C_i^*$ is, the larger would be the potential difference in the search spaces they explore. This, however, also depends on the granularity with which the values of a sequence of observed evaluation functions increase, which is related to the arc costs (or weights) of the search graph. If $C_i^* = C_j^* = C$, then the search space explored by i-A* and j-A* will be different only in the frontier of $f(n) = C$.

3.6 The Case of $h = h^*$ for m-A^*

Like A^*, m-A^* improves its performance if it has access to more accurate heuristics. In particular, when h_1 is strictly larger (and therefore more accurate) than h_2, every node surely expanded by m-A^* with h_2 before the j^{th} solution is uncovered will also be expanded by m-A^* with h_1 before the j^{th} solution is uncovered. The proof idea is identical to the A* case and is omitted. The case of the exact heuristic deserves a special notice. It is easy to show that,

Theorem 4. *If $h = h^*$ is the exact heuristic, then m-A^* generates solutions on j-optimal paths $1 \leq j \leq m$, only.*

Proof. Since the heuristic function is exact the f values in OPEN are expanded in sequence $C_1^* \leq C_2^* \leq ... \leq C_i^* ... \leq C_m^*$. All the nodes generated having $f = C_1^*$ are by definition on optimal paths (since $h = h^*$), all those generated who have $f = C_2^*$ must be on paths that can be second best and so on. (Notice that the best cost may differ only at some indices.)

When $h = h^*$ m-A* is clearly linear in the number of nodes having $f^* \leq C_m^*$ value. However, when the cost function has only a small range of values, there may be exponential number of solution paths having cost C_m^*. To avoid this exponential frontier we chose the tie-breaking rule of expanding deeper nodes first. This will result in a number of nodes expansions that is bounded by mn when n bounds the solution length.

In summary,

Theorem 5. *When m-A^* has access to $h = h^*$, then, if it uses a tie breaking rule in favor of deeper nodes, it will expand at most $\#N$ nodes, where $\#N = \sum_i \#N_i$ and $\#N_i$ is the length of the i-optimal solution path. Clearly $\#N \leq mn$.*

4 Branch and Bound for m-best Solutions

Straightforwardly extending the well-known *B&B* scheme, algorithm m-Branch-and-Bound (*m-B&B*) finds the m best solutions by exploring the search space in a depth first manner.

The algorithm maintains an ordered list of the m best solutions found so far. It prunes nodes whose evaluation function (a lower bound on the optimal solution passing through the node) is greater than U_m, current upper bound on the m^{th} best solution ($U_m = \infty$ until initial m solutions are found). At time of completion, *m-B&B* outputs an ordered list of the m best solutions.

In analyzing the complexity of *m-B&B* we assume that the underlying search space is a graph (as opposed to a search tree). The three main sources of complexity, compared with finding a single best solution, are: the expansion of the search space, the node processing overhead due to m best problem and the impact of the cost function on pruning the explored search space. We defer the analysis of the first two factors till Section 5.3, where we discuss the application of *m-B&B* to graphical models.

Similar to m-A*, the number of nodes expanded by *m-B&B* greatly depends on the values of the evaluation function and costs of the solutions. Since, as we have already shown, m-A* is superior to any exact search algorithm for m-best solutions, *m-B&B must* expand all the nodes that are *surely expanded* by m-A*, which, assuming consistent heuristic, is the set of nodes $E^*_m = \{n|f(n) < C^*_m\}$.

The order in which *m-B&B* uncovers solutions is problem-specific and has a big impact on the total number of node expantions. Let $\{U^1_m, ..., U^j_m\}$ be the sequence of upper bounds on the m^{th} best solution, at the time when *m-B&B* uncovered j solutions, ordered from the earliest bound obtained to the latest. These upper-bounds, that influence the node pruning, are decreasing until they coincide with C^*_m.

Proposition 4. *Given a consistent heuristic*

1. m-B&B *will expand all nodes such as* $E^*_m = \{n|f(n) < C^*_m\}$.

2. Initially, and until m-B&B *encounters the true* m^{th}*-best solution, it will expand also nodes* $F^*_m = \{n|f(n) > C^*_m\}$. *Subsequently, like m-A* it will only expand nodes for which* $f(n) \leq C^*_m$.

5 Applying *m*-A* and *m-B&B* to Graphical Models.

In this section we apply m-A* and *m-B&B* to combinatorial optimization tasks (COP) defined over graphical models. We focus more on the application of *m-B&B*, since it is more practical then m-A* due to flexible memory requirements.

5.1 Background on graphical models.

Consider a weighted constraint problem expressed as a **graphical model** $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \sum)$, where $\mathbf{F} = \{f_1, \ldots, f_r\}$ is a set of discrete functions, called costs, over real-valued variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ with discrete domains $\mathbf{D} = \{D_1, \ldots, D_n\}$. We aim to minimize the sum of all costs $\min_{\mathbf{X}} \sum_i f_i$. Closely related combinatorial optimization problem is Most Probable Explanation (MPE), where the task is to compute $\max_{\mathbf{X}} \prod_i f_i$, where the sum is replaced by a product. The set of function scopes implies a *primal graph* and, given an ordering of the variables, an *induced graph* (where, from last to first, each node's earlier neighbors are connected) with a certain *induced width* w^*.

Example 2. Figure 4a depicts the primal graph of a problem with six variables. Figure 4b shows its induced graph along the ordering $d = A, B, C, D, E, F$, $w^* = 2$.

AND/OR search space of a graphical model exploits the problem decomposition captured in the structure of the graph (see [3] for more details). The search space is defined using a **pseudo tree** of the graphical model. A pseudo tree of an undirected graph $G = (X, E)$ is a directed, rooted tree $\mathcal{T} = (X, E')$, such that every arc of G not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in T to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E.

AND/OR Search Trees. Given a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \sum)$ and a pseudo tree $\mathcal{T}$, the *AND/OR search tree* guided by $\mathcal{T}$ consists of alternating levels of OR and AND nodes. Figure 4d shows the AND/OR search tree for our running example, guided by the pseudo-tree in Figure 4c.

It was shown that, for a problem with N variables with domain size k and a pseudo tree $\mathcal{T}$ of height h, the size of the corresponding AND/OR search tree is $O(Nk^h)$. Alternatively the size can be bounded by $O(Nk^{w^* \log N})$, where w^* is the induced width of the problem graph along a depth-first traversal of $\mathcal{T}$ [3].

AND/OR Search Graphs. Identical subproblems, identified by their context (the partial instantiation that separates the subproblem from the rest of the network), can be merged, yielding the *context-minimal AND/OR search graph*, at the expense of using additional memory during search. Merging nodes of the AND/OR tree in Figure 4d yields the context-minimal AND/OR search graph in Figure 4e. The context-minimal AND/OR search graph has size $O(Nk^{w^*+1})$ (see [3] for details).

An optimization problem can be solved by searching the corresponding *weighted AND/OR search graph*, namely an AND/OR search graph, in which each edge from OR node to AND node has a weight derived from the set of cost functions F [3]. As a heuristic evaluation function these search algorithms use the mini-bucket heuristic that is known to be admissible and consistent, yielding algorithms AOBB (AND/OR Branch and Bound) and AOBF (AND/OR Best-First), described and extensively studied in [7, 9]. The solution to the problem is a subtree of the weighted AND/OR search graph.

5.2 Algorithm m-AOBF for graphical models.

As noted, *AOBF* (AND/OR Best First) is version of algorithm A^* that searches the weighted AND/OR context minimal graph in a best-first manner. It can be guided by any admissible heuristic. Extending AOBF to the m-best task as suggested by the general m-A* yields an algorithm which we call *m-AOBF*. All

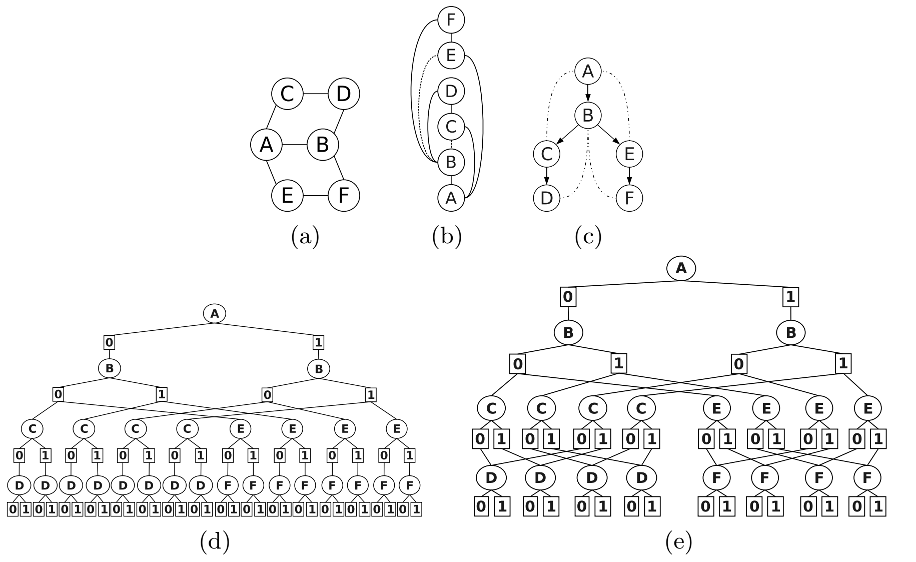


Fig. 4: Example primal graph with six variables (a), its induced graph along ordering $d = A, B, C, D, E, F$ (b), a corresponding pseudo tree (c), the resulting AND/OR search tree (d), and the context-minimal AND/OR search graph (e).

the properties of *m-A** extends to AND/OR search graphs. The algorithm may require duplicating some nodes at most m times (if there are m paths leading to them).

The complexity of m-best algorithms for graphical models can be analyzed by first characterizing the underlying search space that is being explored using graph parameters, as is common in graphical models. Subsequently, we can characterize the portion of the search space that is explored using the evaluation function, as we did earlier. So, since *m-AOBF* explores the AND/OR context-minimal graph (denoted CMG), and since we may duplicate some nodes at most m times we get:

Theorem 6. *The search space size explored by* m-AOBF *is bounded by* $O(N \cdot m \cdot k^{w*})$*, where* w^* *is the induced-width along the pseudo-tree ordering.*

Obviously this bound is loose since it does not consider the pruning power of the evaluation function.

Theorem 7. *Given an evaluation function* $f(n)$ *that underestimates the least cost solution tree that passes through the node n in the AND/OR context-minimal graph, algorithm* m-AOBF *surely expands the set of nodes* $\{n \in CMG | f(n) < C^*_m\}$*, where* C^*_m *is the cost of the* m^{th} *best solution, and a subset of nodes for which* $\{n \in CMG | f(n) = C^*_m\}$*, depending on the tie-breaking rule.*

5.3 Algorithm m-AOBB for Graphical models

Extending *AOBB* to *m-AOBB* is straightforward, mimicking the *m-B&B* while searching the AND/OR search space. The main difference between *m-AOBB* and *m-B&B* arises from the presence of AND nodes: at each AND node the m best solutions to the subproblems rooted in its children need to be combined and the best m out of the combination results need to be chosen. We distinguish between *m-AOBB* searching the AND/OR tree and the context minimal AND/OR graph. These two versions of *m-AOBB* differ in the size of underlying search space and the computational overhead per node.

The size of the AND/OR search tree is bounded by $O(Nk^h)$, where h is the height of the pseudo-tree. The primary overhead is due to the combination of the solutions to the children's subproblems of AND nodes which is $O(deg \cdot m \log m)$ time per AND node. Consequently, if the algorithm that searches the underlying AND/OR tree (with no caching) the run-time complexity is bounded by $O(m \cdot Nk^h \cdot deg \log m)$.

If the algorithm searches the context minimal AND/OR graph, (where identical subproblems are identified based on their context) *m-AOBB* need to cache the m best solutions rooted in some OR nodes, which requires additional memory, similar to graph-based AOBB [9]. Such caching introduces time overhead of $O(m \log m)$ per OR node. Since the size of an AND/OR search graph is bounded by $O(Nk^{w*})$ nodes and considering the AND node-related overhead, we can conclude that

Theorem 8. *The time complexity of m-AOBB which searches the underlying AND/OR search tree is bounded by $O(m \cdot Nk^h \cdot deg \log m)$. The time complexity of m-AOBB which searches the context-minimal graph is $O(N \cdot deg \cdot m \log mk^{w^*})$ and the space complexity is $O(N \cdot m \log mk^{w^*})$, where w^* is the induced-width of the ordering along the pseudo-tree that guides the search and h is its height.*

The above analysis considers only the impact of the m-best exploration on the size of the underlying search space and the overhead computation per node, but ignore the pruning power caused by the evaluation function that guides the search. Clearly the complexity analysis can be further refined by taking into consideration cost function, directly extending the results of Proposition 4 to *m-AOBB*.

5.4 Algorithm BE-Greedy-m-BF

Since an exact heuristic for graphical models can be generated by the bucket-elimination (BE) algorithm [7], we can use the idea suggested in Section 3.6, yielding algorithm which we call *BE-Greedy-m-BF*. The algorithm first generates the exact heuristics along an ordering using *BE* and subsequently applies *m-AOBF* using these exact heuristics. It turns out that the worst-case complexity of both algorithms when applied in sequence coincides with the best of the known m-best algorithms for graphical models. This is because the number of nodes expanded by m-A* when it uses the exact heuristic is bounded by $N \cdot m$, when N is the number of variables.

Theorem 9. *Let $B = (X, D, F)$ be a general graphical model. The complexity of* BE-Greedy-m-BF *(i.e., bucket-elimination followed by m-AOBF with exact heuristic) is $O(Nk^{w*} + N \cdot m)$ when N is the number of variables.*

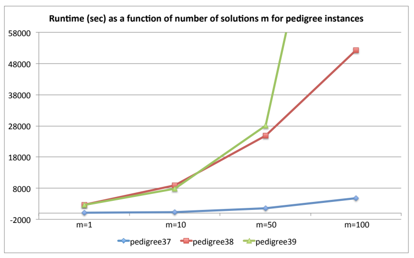
6 Earlier work on finding the m-best solutions.

The most influential work on finding the m-best solution was written by Lawler [8]. In the folowing years a great number of related methods were developed, which we do not describe here for lack of space. Particular areas of interest include the problem of finding k shortest paths, extensive references for which can be found in [5]. An overview of algorithms focused on graphical model is presented in [6].

7 Empirical demostrations

Instance	N	k	w*	h	m=1	m=10	m=50	m=100
pedigree1	298	4	15	59	1	32	464	1751
pedigree37	726	5	20	72	41	240	1537	4825
pedigree38	581	5	16	52	2700	8890	24894	t/o
pedigree39	953	5	20	77	2594	7907	28037	t/o
pedigree50	478	6	16	54	685	2835	21903	t/o
grid50-12-5	143	2	15	48	1	2	13	35
grid50-14-5	195	2	18	64	87	599	1995	3739
grid50-15-5	224	2	19	76	180	1382	5011	9591
grid75-16-5	256	2	21	73	539	3327	10917	20572
grid75-18-5	324	2	24	85	1147	4533	14078	23948
mm-03-0000	1220	2	18	43	43	580	4845	13381
mm-03-0001	1193	2	15	38	1	2	8	23
mm-03-0002	1193	2	15	39	1	3	17	48
mm-03-0004	1193	2	15	38	0	3	17	46
mm-03-0005	1193	2	15	38	0	5	25	70
mm-03-0011	1172	2	18	42	18	148	757	2016
mm-03-0014	1172	2	18	43	15	107	544	1488
mm-04-0012	2224	2	29	56	842	6859	31952	t/o
mm-04-0013	2224	2	29	58	375	4880	23854	t/o
mm-04-0014	2224	2	29	60	655	8080	36740	t/o
mm-04-0015	2224	2	29	57	818	13299	t/o	t/o

(a) Run-time of *m-AOBB* in seconds.



(b) Pedigree instances. Pedigree 39 could not be solved before the time-out of 12 hours, thus the runtime for m=100 is unavailable.

Fig. 5: The run-time of *m-AOBB* in seconds as a function of number of solutions m

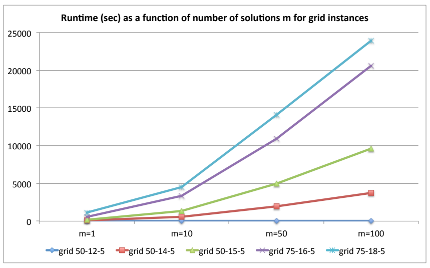
Due to our interest in extremely memory-intensive problems (e.g. genetic linkage analysis), we deem Branch and Bound-based methods with their flexible memory requirements the more promising in practice. In the following preliminary experiments we focused on evaluating how *m-AOBB* scales with number of solutions m.

Our implementation of *m-AOBB* is exploring an AND/OR search tree and not a graph. By making such design choice not only we achieve better space complexity, it also allows us to avoid the time overhead due to caching. On the other hand, the search space we are exploring is inheritently larger (see Section 5.3).

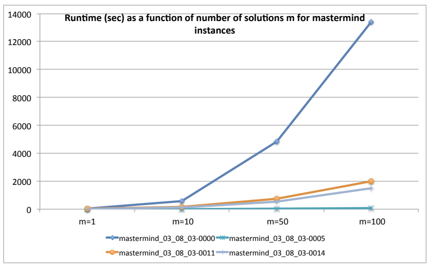
We evaluated the algorithm on maximum probability explanations (MPE) problems using 3 sets of instances taken from 2008 UAI evaluation: pedigrees, grids and mastermind instances. The parameters of the problems and run time results in seconds are presented in Table 5a.

We evaluated the algorithm for $m = [1, 10, 50, 100]$ solutions. The timeout was set to 12 hours. The memory limit for pedigree instances is 4 Gb, for grids and mastermind instances it is 10 Gb. In Figures 5b, 6a and 6b we see the dependence of the runtime on number of solutions m for a number of chosen instances from each set.

The theory states that the run-time of *m-AOBB* should scale with the number of solutions m as $(m \log_2 m)$. In practice, we see that it is not always the case and there exists a large discrepancy between the results. For some instances (e.g. grid 50-12-5, mm-03-001) the scaling of runtime with m is significantly better than what theory suggests, which can be attributed to sucessful pruning of a large part of the search space. However, for some instances (e.g. pedigree38, mm-04-0015) the runtime scales considerably worse. We explain the large scaling factor by the suboptimality of our current implementation, which might introduce overhead unaccounted for by theoretical analysis.



(a) Grids instances.



(b) Mastermind instances.

Fig. 6: The run-time of *m-AOBB* in seconds as a function of number of solutions m for grid and mastermind instances.

8 Conclusion

The paper shows how Best First search algorithm for finding one optimal solution can be extended for finding the m-best solutions. We provide theoretical analysis of soundness and completeness and show that m-A^* is optimally efficient compared with any other algorithm that searches the same search space using the same heuristic function.

We then extend our analysis to Depth First Branch and Bound and apply the algorithms to graphical models optimization tasks, provide worst-case bounds on the search space explored by the m-best algorithms and characterize the added pruning power associated with the heuristic evaluation function. Preliminary empirical evaluation of *m-AOBB* shows that the algorithm often scales with the number of solutions m significantly better than the theory suggests.

We also present *BE-Greedy-m-BF*, a hybrid of variable-elimination and Best First Search scheme, that, interestingly, has the best time complexity amongst m-best algorithms known to us.

References

1. E. Charniak and S.E. Shimony. Cost-based abduction and map explanation. *Artificial Intelligence*, 66(2):345–374, 1994.
2. A. Darwiche, R. Dechter, A. Choi, V. Gogate, and L. Otten. Results from the probablistic inference evaluation of UAI08, a web-report in http://graphmod.ics.uci.edu/uai08/Evaluation/Report. *In: UAI applications workshop*, 2008.
3. R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.
4. R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32:506–536, 1985.
5. David Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.
6. N. Flerova, R. Dechter, and E. Rollon. Bucket and mini-bucket schemes for m best solutions over graphical models. In *GKR'11 Workshop*, 2011.
7. K. Kask and R. Dechter. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence*, pages 91–131, 2001.
8. E.L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.
9. R. Marinescu and R. Dechter. AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 2009.
10. N. J. Nillson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, CA, 1980.
11. J. Pearl. *Heuristics: Intelligent Search Strategies.* Addison-Wesley, 1984.
12. T. Schiex. Arc consistency for soft constraints. *Principles and Practice of Constraint Programming (CP2000)*, pages 411–424, 2000.

Multi-Agent Preference Combination using Subjective Logic

Audun Jøsang

University of Oslo, Norway*
josang@mn.uio.no

Abstract. Situations where agents with different preferences try to agree on a single choice occur frequently. This must not be confused with fusion of evidence from different agents to determine the most likely correct hypothesis or actual event. Multi-agent preference combination assumes that each agent has already made up her mind, and is about determining the most acceptable decision or choice for the group of agents. This paper formalises and expresses preferences for a state variable in the form of subjective opinions over a frame, and then applies the belief constraint operator of subjective logic as a method for merging preferences of multiple agents into a single preference for the whole group. The model is expressive and flexible, and produces perfectly intuitive results.

1 Introduction

In situations where two or more agents need to make a selection among alternatives their preferences can be combined to derive the selection that best satisfies all agents. For example, person *A* might say: *"I like broccoli, but I dislike celery"* and person *B* might say: *"I like both of them"*. Assume that person *A* and person *B* are cooking a meal together, and they want to decide whether to include a particular ingredient, then inclusion of Broccoli is obvious because both like it. The inclusion of celery however is unclear because *A* and *B* have opposite preferences. In this case, cultural norms would play a role, as e.g. politeness, or the relative status or authority of *A* and *B*. If the preferences had been expressed as hard constraints, i.e. if *A* said *"For me celery is out of the question"* and *B* said *"For me celery is mandatory"* then it would seem that they simply can not cook the meal together.

In addition to having both positive and negative preferences, it is natural to also express indifference, stating that we neither have a positive nor a negative preference over a specific object. By continuing the above cooking example, person *A* might say: *"I'm indifferent to carrots"* and person *B* might say: *"I like carrots"*. Then the inclusion of carrots seems natural because *B* likes it and *A* is indifferent, i.e. the indifference of *A* lets *B* decide.

In this paper, we investigate how subjective opinions can be used to express preferences in general. In particular we analyse the applicability of the belief constraint operator, which in fact is an extension of Dempster's rule [11], for combining preferences of multiple agents about the same choice variable. The intuitive motivation behind our study is that preference can be represented as belief and that indifference can

* The work reported in this paper has been partially funded by UNIK.

be represented as uncertainty/uncommitted belief. Positive and negative preferences are considered as symmetric concepts, so they can be represented in the same way and combined using the same operator. A totally uncertain opinion has no influence and thereby represents the neutral element.

Our study focuses on multi-agent preferences over a single variable represented as the possible states in a frame. In future research we plane to extend our study by analysing multi-agent preferences over multiple variables, i.e. over multiple frames.

2 Related Work

The work presented here extends the fundamental idea of bipolar preferences wherein agents can express positive and/or negative preferences for a particular choice. Publications that focus on this principle include [1,2]. In [2] the soft constraint formalism based on semirings is used to model negative preferences, and a separate algebraic structure is used to model positive preferences. To model bipolar problems these two structures are linked and the highest negative preference is set to coincide with the lowest positive preference to model indifference. A combination operator is defined between positive and negative preferences to model preference compensation. In [1] uncertainty is modelled by the presence of uncontrollable variables. This means that the value of such variables is decided by "Nature" or by some other agent. A solution is then only assigned to controllable variables, not to uncontrollable ones. A typical example of uncontrollable variable, in the context of satellite scheduling, is a variable representing the time when clouds will disappear. Although the value for such uncontrollable variables can not be chosen directly, the plausibility of the values in their domains can be expressed. The plausibility information, which is not bipolar, is expressed by probability distributions.

Possibility theory applied to preference combination has also be investigated e.g. in [3,10]. The main idea in [10] is to represent preferences (or respective certainty degrees) as a possibility distribution over labelings (choices). Such a distribution then induces a possibility measure and a necessity measures over constraints. With this formalism constraints can be expressed as bounds on possibility or necessity defining a set of possibility distribution among labelings. One can then define a set of "most possible" labelings satisfying these bounds. The main idea in [3] formalizes the notion of possibilistic constraint satisfaction problems (CSP) that allows the modeling of uncertainly satisfied constraints. Necessity-valued constraints then express the respective certainty degrees of each constrain.

3 Subjective Logic Basics

Our model follows the same ideas as those of the models mentioned in Sec.2 above, namely to express and combine positive and negative preferences as well as indifference/uncertainty. However, the model of subjective logic is quite different from the models of the previous approaches. Subjective opinions simultaneously express positive and negative preferences as well as indifference/uncertainty, thereby avoiding the complexity of integrating multiple formalisms to express the various aspects of preferences. A subjective opinion expresses preferences over possible states of a frame, which constitutes a multi-polar preference model. Preference combination based on subjective logic can be interpreted as a form of majority voting where the weight of each agent's vote is inversely proportional to the indifference/uncertainty of that agent's preference. A totally indifferent/uncertain opinion then carries no weight and represents the neutral element. Subjective logic thus provides the basis for a very general preference combination model.

A subjective opinion is a composite function that consists of belief masses, uncommitted belief mass (uncertainty) and base rates, and that can also indicate the belief source or owner. The main idea behind our study is to interpret belief mass as preference, and uncommitted belief mass as indifference. Base rates can be interpreted as average preferences in the population.

3.1 The Reduced Powerset of Frames

A state space of mutually exclusive states is called a *"frame of discernment"* or *"frame"* for short. Let X be a frame of cardinality k. In this study the possible states in the frame represent the preference variable, i.e. agents can express preferences over states in the frame. It is assumed that the goal of the multi-agent preference is to select a single state from the frame as the most preferred state for the group of agents.

Belief mass (preference) is distributed over the reduced powerset of the frame denoted as $\mathscr{R}(X)$. More precisely, the reduced powerset $\mathscr{R}(X)$ is defined as:

$$\mathscr{R}(X) = 2^X \setminus \{X, \emptyset\} = \{x_i \mid i = 1 \ldots k,\ x_i \subset X\} , \tag{1}$$

which means that all proper subsets of X are elements of $\mathscr{R}(X)$, but X itself is not in $\mathscr{R}(X)$. The emptyset $\emptyset$ is also not considered to be a proper element of $\mathscr{R}(X)$.

An agent can thus express preference for singleton states as well as for subsets containing multiple singletons. Assigning belief mass to a singleton or to a subset is interpreted as positive preference for that singleton or subset, and as negative preference for their complements in the frame. This can be considered as model for expressing multi-polar preferences, and thereby extends the idea of bipolar preferences described in [1,2]. By not assigning all the belief mass to singletons or subsets the agent can express indifference, i.e. the level of indifference is equal to the amount of uncommitted belief mass.

The cardinality of of $\mathscr{R}(X)$ is computed as $\kappa = |\mathscr{R}(X)| = (2^k - 2)$, i.e. the reduced powerset has only $(2^k - 2)$ elements because it is assumed that X and $\emptyset$ are not elements of $\mathscr{R}(X)$. The first k elements of $\mathscr{R}(X)$ have the same index as the corresponding singletons of X. The remaining elements of $\mathscr{R}(X)$ are grouped in classes according to the number j of singletons they contain. The class is then called "class j", meaning that all elements belonging to class j have cardinality j. The actual number of elements belonging to each class is determined by the Choose Function $\mathrm{C}(\kappa, j)$ which dictates the number of ways that j out of κ singletons can be chosen. The Choose Function, equivalent to the binomial coefficient, is defined as:

$$\mathrm{C}(\kappa, j) = \binom{\kappa}{j} = \frac{\kappa!}{(\kappa - j)!\, j!} . \tag{2}$$

Within a class each element is indexed after the order of the lowest indexed singletons from X that it contains. For example in case of the frame $X = \{x_1, x_2, x_3, x_4\}$, class 1 has 4 elements, and class 2 has 6 elements, which together makes 10 elements. The first element of class 3 therefore has index 11. Table 1 defines the index and class of all the elements of $\mathscr{R}(X)$ according to this scheme in case of $|X| = 4$.

	Singleton selection per element													
Singletons x_4				*			*		*	*		*	*	*
Singletons x_3			*			*		*		*	*		*	*
Singletons x_2		*			*			*	*		*	*		*
Singletons x_1	*				*	*	*				*	*	*	
Element Index:	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Element Class:	1				2						3			

Table 1. Index and class of elements of $\mathscr{R}(X)$ in case $|X| = 4$.

Class-1 elements are the original singletons from X, i.e. we can state the equivalence $(x_i \in X) \Leftrightarrow (x_i$ is a class-1 element in $\mathscr{R}(X))$. The frame $X = \{x_1, x_2, x_3, x_4\}$ does not figure as an element of $\mathscr{R}(X)$ in Table 1 because excluding X is precisely what makes $\mathscr{R}(X)$ a reduced powerset.

3.2 Belief Distribution over the Reduced Powerset

Subjective logic allows various types of belief mass distributions over a frame X, which in this study is interpreted as a *preference mass distribution*. The distribution vector can be additive (i.e. sum = 1) or sub-additive (i.e. sum $<$ 1), and it can be restricted to elements of X or it can include proper subsets of X. A belief mass on a proper subset of X is equivalent to a belief mass on an element of $\mathscr{R}(X)$. In case of sub-additive belief mass distribution, (i.e. sum $<$ 1) the complement is defined as uncommitted belief mass, which in this study is interpreted as *indifference mass*. An additive belief mass distribution means that there is no uncommitted mass. In general, the belief vector $\vec{b}_X$ specifies the distribution of belief masses over the elements of $\mathscr{R}(X)$, and the uncommitted mass denoted as u_X represents the uncertainty about the probability expectation value, as will be explained below. The sub-additivity of the belief vector and the complement property of the uncommitted mass (uncertainty) are expressed by Eq.(3) and Eq.(4) below:

$$\text{Belief sub-additivity: } \sum_{x_i \in \mathscr{R}(X)} \vec{b}_X(x_i) \leq 1 \ , \ \vec{b}_X(x_i) \in [0,1] \tag{3}$$

$$\text{Belief and uncertainty additivity: } u_X + \sum_{x_i \in \mathscr{R}(X)} \vec{b}_X(x_i) = 1 \ , \ \vec{b}_X(x_i), u_X \in [0,1] \ . \tag{4}$$

An element $x_i \in \mathscr{R}(X)$ is a *focal element* when its belief mass is non-zero, i.e. when $\vec{b}_X(x_i) > 0$. The frame X is not considered to be a focal element, even when $u_X > 0$.

3.3 Base Rates over Frames

The concept of base rates is central in the theory of probability, and also in subjective logic. Given a frame of cardinality k, the default base rate of for each singleton in the frame is $1/k$, and the default base rate of a subset consisting of n singletons is n/k. In other words, the default base rate of a subset is equal to the number of singletons in the subset relative to the cardinality of the whole frame. A subset also has default *relative base rates* with respect to every other fully or partly overlapping subset of the frame.

In practical situations base rates are normally different from the default values. When modelling preferences, base rates can express average preferences in the population. The base rate function is denoted as a so that $a(x_i)$ represents the base rate of element $x_i \in X$. The base rate function is formally defined below.

Definition 1 (Base Rate Function). Let X be a frame of cardinality k, and let $\vec{a}_X$ be the function from X to $[0,1]^k$ satisfying:

$$\vec{a}_X(\emptyset) = 0, \quad \vec{a}_X(x_i) \in [0,1] \quad \text{and} \quad \sum_{i=1}^{k} \vec{a}_X(x_i) = 1 \ . \tag{5}$$

Then $\vec{a}_X$ is a base rate distribution over X.

Two different observers can share the same base rate vectors. However, it is obvious that two different observers can also assign different base rates to the same frame, in addition to assigning different beliefs to the frame. This naturally reflects different views, analyses and interpretations of the same situation by different observers. Base rates can thus be partly objective and partly subjective.

Events that can be repeated many times are typically frequentist in nature, meaning that the base rates for these often can be derived from statistical observations. For events that can only happen once, the analyst must often extract base rates from subjective intuition or from analyzing the nature of the phenomenon at hand and any other relevant evidence. However, when no specific base rate information is known, the default base rate of the singletons in a frame must be defined to be equally partitioned between them. More specifically, when there are k singletons in the frame, the default base rate of each element is $1/k$. For this study, the base rates are interpreted as average preferences in the population.

The usefulness of base rate function emerges from its application as the basis for probability projection. Because belief mass can be assigned to any subset of the frame it is necessary to also represent the base rates of such subsets. This is defined below.

Definition 2 (Subset Base Rates). Let X be a frame of cardinality k, and let $\mathscr{R}(X) = 2^X \setminus \{X, \emptyset\}$ be its reduced powerset of cardinality $\kappa = (2^k - 2)$. Assume that a base rate function $\vec{a}_X$ is defined over X according to Def.1. Then the base rates of the elements of the reduced powerset $\mathscr{R}(X)$ are expressed according to the powerset base rate function $\vec{a}_{\mathscr{R}(X)}$ from $\mathscr{R}(X)$ to $[0,1]^{\kappa}$ expressed below:

$$\vec{a}_{\mathscr{R}(X)}(\emptyset) = 0 \quad \text{and} \quad \vec{a}_{\mathscr{R}(X)}(x_i) = \sum_{\substack{x_j \in X \\ x_j \subseteq x_i}} \vec{a}_X(x_j) \ , \ \ \forall x_i \in \mathscr{R}(X) \ . \tag{6}$$

Note that $x_j \in X$ means that x_j is a singleton in X, so that the subset base rate in Eq.(6) is the sum of base rates on singletons $x_j \in x_i$. Trivially, it can be seen that when $x_i \in X$ then $\vec{a}_{\mathscr{R}(X)}(x_i) \equiv \vec{a}_X(x_i)$, meaning that $\vec{a}_{\mathscr{R}(X)}$ simply is an extension of $\vec{a}_X$. Because of this strong correspondence between $\vec{a}_{\mathscr{R}(X)}$ and $\vec{a}_X$ we will simply denote both base rate functions as $\vec{a}_X$. Because belief masses can be assigned to fully or partially overlapping subsets of the frame it is necessary to also derive relative base rates of subsets as a function of the degree of overlap with each other. This is defined below.

Definition 3 (Relative Base Rates). Assume frame X of cardinality k where $\mathscr{R}(X)$ is its reduced powerset of cardinality $\kappa = (2^k - 2)$. Assume the base rate function $\vec{a}_X$ defined over X according to Def.2. Then the base rates of an element x_i relative to an element x_j is expressed by the relative base rate function $\vec{a}_X(x_i/x_j)$ expressed below:

$$\vec{a}_X(x_i/x_j) = \frac{\vec{a}_X(x_i \cap x_j)}{\vec{a}_X(x_j)} \ , \ \ \forall \, x_i, x_j \in \mathscr{R}(X) \ . \tag{7}$$

4 Opinion Classes

An opinion is a composite function consisting of the belief mass vector $\vec{b}_X$, uncommitted belief mass u_X and the base rate vector $\vec{a}_X$, and can also indicate ownership whenever required. A subjective opinion is normally denoted as ω_X^A where A is the opinion owner, also called the subject, and X is the target frame to which the opinion applies [5]. An alternative notation is $\omega(A:X)$. There can be different classes of opinions, of which *hyper opinions* are the most general. *Multinomial opinions* and *binomial opinions* are specific sub-types. More specific opinion classes are DH opinion (Dogmatic Hyper), UB Opinion (Uncertain Binomial) etc. The six main opinion classes defined in this way are listed in Table 2 below, and are described in more detail in the next sections.

	Binomial Binary frame Focal element $x \in X$	**Multinomial** n-ary frame Focal elements $x \in X$	**Hyper** n-ary frame Focal elements $x \in \mathscr{R}(X)$
Uncertain $u > 0$	**UB opinion** Beta pdf	**UM opinion** Dirichlet pdf over X	**UH opinion** Dirichlet pdf over $\mathscr{R}(X)$
Dogmatic $u = 0$	**DB opinion** Scalar probability	**DM opinion** Probabilities on X	**DH opinion** Probabilities on $\mathscr{R}(X)$

Table 2. Opinion classes with equivalent probabilistic representations

The propositions of a frame are assumed to be exhaustive and mutually disjoint. For binary frames the opinion is binomial. In case the frame is larger than binary, and only singletons of X (i.e. class-1 elements of $\mathscr{R}(X)$) are focal elements, then the opinion is multinomial. In case the frame is larger than binary and there are focal elements of any class of $\mathscr{R}(X)$), then it is a *hyper opinion*. In case of uncommitted belief mass, i.e. $u_X > 0$, it is called an *uncertain opinion* which expresses degrees of indifference. In case $u_X = 0$ it is called a *dogmatic opinion* which represents dogmatic preferences.

The six entries in Table 2 also mention the equivalent probability representation of opinions, e.g. as Beta pdf, Dirichlet pdf or as a distribution of scalar probabilities over elements of X or $\mathscr{R}(X)$ [8]. This offers a frequentist interpretation of subjective opinions and preferences, and provides a method for deriving opinions and preferences from statistical data. Alternatively it is possible to map subjective opinions and preferences to Beta pdfs or Dirichlet pdfs, for further processing and analysis within classical statistical frameworks. The detailed description of the equivalence between opinions and probability density functions is outside the scope of this paper.

4.1 Binomial Opinions

A special notation is used for representing opinions over binary frames. A general n-ary frame X can be considered binary when seen as a binary partitioning consisting of one of its proper subsets x and the complement $\bar{x}$.

Definition 4 (Binomial Opinion). *Let $X = \{x, \bar{x}\}$ be either a binary frame or a binary partitioning of an n-ary frame. A binomial opinion about the truth of state x is the ordered quadruple $\omega_x = (b,d,u,a)$ where:*

b :	**belief**	*belief mass in support of x being true (preference for x),*
d :	**disbelief**	*belief mass in support of x being false (negative preference for x),*
u :	**uncertainty**	*the amount of uncommitted belief mass (indifference about x),*
a :	**base rate**	*the* a priori *probability of x (average preference for x).*

These components satisfy $b+d+u=1$ and $b,d,u,a \in [0,1]$. The characteristics of various binomial opinion classes are listed below. A binomial opinion:

where $b=1$	is equivalent to binary logic TRUE (hard positive constraint),
where $d=1$	is equivalent to binary logic FALSE (hard negative constraint),
where $b+d=1$	is equivalent to a traditional probability (preference),
where $b+d<1$	expresses degrees of uncertainty (indifference), and
where $b+d=0$	expresses total uncertainty (indifference).

Binomial opinions can be represented on an equilateral triangle as shown in Fig.1.

A point inside the triangle represents a (b,d,u) triple. The belief, disbelief, and uncertainty-axes run from one edge to the opposite vertex indicated by the b_x axis, d_x axis and u_x axis labels. The base rate[1], is shown as a point on the base line, and the probability expectation, E_x, is formed by projecting the opinion point onto the base, parallel to the base rate director line. The opinion $\omega_x = (0.2,\ 0.5,\ 0.3,\ 0.6)$ with expectation value $\mathrm{E}_x = 0.38$ is shown as an example.

[1] Also called *relative atomicity* in case of default base rates [5]

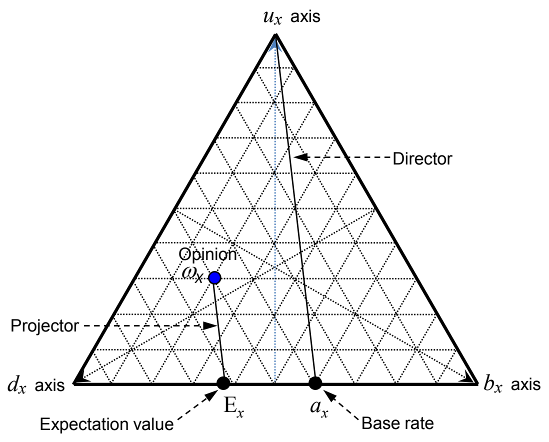


Fig. 1. Opinion triangle with example opinion

Binomial opinions with $u \geq 0$ are called UB opinions (Uncertain Binomial), whereas binomial opinions with $u = 0$ are called DB opinions (Dogmatic Binomial). DB opinions are equivalent to classical probabilities. In case the opinion point is located at one of the three vertices in the triangle, i.e. with $b = 1$, $d = 1$ or $u = 1$, the reasoning with such opinions becomes a form of three-valued logic that is an extension of Kleene logic [4]. Because the three-valued arguments of Kleene logic do not contain base rates, probability expectation values can not be derived from Kleene logic arguments. The conjunction of multiple Kleene logic arguments is therefore incompatible with multiplication of probabilities or opinions [7], and is inconsistent in general because the conjunction of an infinity of UNKNOWN arguments produces UNKNOWN in Kleene logic, whereas realistically it should converge towards FALSE. Eq.(8) defines the probability projection of a binomial opinion on proposition x:

$$\mathrm{E}_x = b + au\,. \tag{8}$$

In case the opinion point is located at the left or right bottom vertex in the triangle, i.e. with $d = 1$ or $b = 1$ and $u = 0$, the opinion is equivalent to boolean TRUE or FALSE, and is called an AB (Absolute Binomial) opinion. Reasoning with AB opinions is an extension of reasoning within binary logic.

4.2 Multinomial Opinions

An opinion on a frame X that is larger than binary and where the set of focal elements is restricted to class-1 elements is called a multinomial opinion. The uncommitted belief mass, which can be interpreted as uncertainty mass on the frame X, represents indifference in the present model. A UM (Uncertain Multinomial) opinion has $u_X > 0$, and a DM (Dogmatic Multinomial) has $u_X = 0$. Multinomial opinions on ternary frames can be presented as a point inside a tetrahedron, as shown in Fig.2.

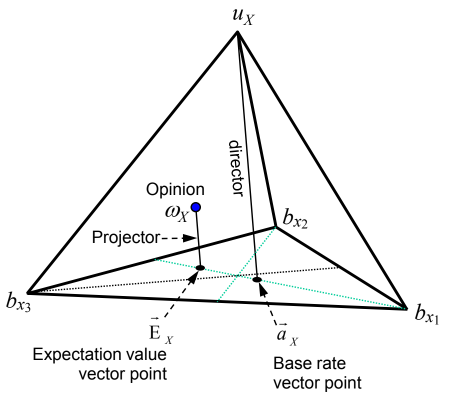


Fig. 2. Opinion tetrahedron with example opinion

The vertical elevation of the opinion point inside the tetrahedron represents the uncertainty mass in Fig.2. The distances from each of the three triangular side planes to the opinion point represents the respective belief mass values. The base rate vector $\vec{a}_X$ is indicated as a point on the base plane. The line that joins the tetrahedron summit and the base rate vector point represents the director. The probability expectation vector point is geometrically determined by drawing a projection from the opinion point parallel to the director onto the base plane.

A multinomial opinion thus contains $(2k+1)$ parameters. However, given Eq.(4) and Eq.(5), multinomial opinions only have $(2k-1)$ degrees of freedom.

In general, the triangle and tetrahedron belong to the *simplex* family of geometrical shapes. Multinomial opinions on frames of cardinality k can in general be represented as a point in a simplex of dimension $(k+1)$. The probability projection of multinomial opinions is expressed by Eq.(9) below:

$$\vec{\mathrm{E}}_X(x_i) = \vec{b}_X(x_i) \; + \; \vec{a}_X(x_i)\, u_X \; , \quad \forall\, x_i \in X \; . \tag{9}$$

The probability projection of multinomial opinions expressed by Eq.(9) is a generalisation of the probability projection of binomial opinions expressed by Eq.(8).

4.3 Hyper Opinions

An opinion on a frame X of cardinality $k > 2$ where any element $x \in \mathscr{R}(X)$ can be a focal element is called a hyper opinion. The special characteristic if this opinion class is thus that possible focal elements $x \in \mathscr{R}(X)$ can be overlapping subsets of the frame X. The frame X itself can have uncertainty mass assigned to it, but is not considered as a focal element. Definition 5 below not only defines hyper opinions, but also represents a general definition of subjective of opinions. In case $u_X \neq 0$ it is called a UH opinion (uncertain hyper opinion), and in case $u_X = 0$ it is called a DH opinion (dogmatic hyper opinion).

Definition 5. Hyper Opinion
Assume X be to a frame where $\mathscr{R}(X)$ denotes its reduced powerset. Let $\vec{b}_X$ be a belief vector over the elements of $\mathscr{R}(X)$, let u_X be the complementary uncertainty mass, and let $\vec{a}$ be a base rate vector over the frame X, all seen from the viewpoint of the opinion owner A. The composite function $\omega_X^A = (\vec{b}_X, u_X, \vec{a}_X)$ is then A's hyper opinion over X.

The belief vector $\vec{b}_X$ has $(2^k - 2)$ parameters, whereas the base rate vector $\vec{a}_X$ only has k parameters. The uncertainty parameter u_X is a simple scalar. A hyper opinion thus contains $(2^k + k - 1)$ parameters. However, given Eq.(4) and Eq.(5), hyper opinions only have $(2^k + k - 3)$ degrees of freedom.

Hyper opinions represent the most general class of opinions. It is challenging to design meaningful visualisations of hyper opinions because belief masses are distributed over the reduced powerset with partly overlapping elements. It can be seen that for a frame X of cardinality $k = 2$ a multinomial and a hyper opinion both have 3 degrees of freedom which is the same as for binomial opinions. Thus both multinomial and hyper opinions collapse to binomial opinions in case of binary frames.

The integration of the base rates in opinions allows the probability projection to be independent from the internal structure of the frame. The probability expectation of hyper opinions is a vector expressed as a function of the belief vector, the uncertainty mass and the base rate vector.

Definition 6 (Probability Projection of Hyper Opinions).
Assume X to be a frame of cardinality k where $\mathscr{R}(X)$ is its reduced powerset of cardinality $\kappa = (2^k - 2)$. Let $\omega_X = (\vec{b}_X,\ u_X,\ \vec{a}_X)$ be a hyper opinion on X. The probability projection of hyper opinions is defined by the vector $\vec{\mathrm{E}}_X$ from $\mathscr{R}(X)$ to $[0,1]^\kappa$ expressed as:

$$\vec{\mathrm{E}}_X(x_i) = \sum_{x_j \in \mathscr{R}(X)} \vec{a}_X(x_i/x_j)\, \vec{b}_X(x_j) \;+\; \vec{a}_X(x_i)\, u_X\ , \quad \forall\, x_i \in \mathscr{R}(X)\ . \qquad (10)$$

5 The Belief Constraint Operator

The belief constraint operator described here is an extension of Dempster's rule which in Dempster-Shafer belief theory is often presented as a method for fusing evidence from different sources [11]. Many authors have however demonstrated that Dempster's rule is not an appropriate operator for evidence fusion [12], and that it is better suited as a method for combining constraints [8], which is also our view.

Assume two opinions ω_X^A and ω_Y^B over the frame X. The superscripts A and B are attributes that identify the respective belief sources or belief owners. These two opinions can be mathematically merged using the belief constraint operator denoted as "$\odot$" which in formulas is written as: $\omega_X^{A\&B} = \omega_X^A \odot \omega_X^B$. Belief source combination denoted with "&" thus represents opinion combination with "$\odot$". The algebraic expression of the belief constraint operator for subjective opinions is defined next.

Definition 7 (Belief Constraint Operator).

$$\omega_X^{A\&B} = \omega_X^A \odot \omega_X^B = \begin{cases} \vec{b}^{A\&B}(x_i) = \frac{Har(x_i)}{(1-Con)}, & \forall\, x_i \in \mathscr{R}(X),\, x_i \neq \emptyset \\ u_X^{A\&B} \quad = \frac{u_X^A u_X^B}{(1-Con)} & \\ \vec{a}^{A\&B}(x_i) = \frac{\vec{a}^A(x_i)(1-u_X^A)+\vec{a}^B(x_i)(1-u_X^B)}{2-u_X^A-u_X^B}, & \forall\, x_i \in X,\, x_i \neq \emptyset \end{cases} \tag{11}$$

The term $Har(x_i)$ represents the degree of *Harmony*, or in other words overlapping belief mass, on x_i. The term *Con* represents the degree of belief *Conflict*, or in other words non-overlapping belief mass, between ω_X^A and ω_X^B. These are defined below:

$$\begin{aligned} Har(x_i) &= \vec{b}^A(x_i)u_X^B + \vec{b}^B(x_i)u_X^A + \textstyle\sum_{y\cap z=x_i} \vec{b}^A(y)\vec{b}^B(z), \qquad \forall\, x_i \in \mathscr{R}(X)\,. \\ Con \quad &= \textstyle\sum_{y\cap z=\emptyset} \vec{b}^A(y)\vec{b}^B(z)\,. \end{aligned} \tag{12}$$

The purpose of the divisor $(1-Con)$ in Eq.(11) is to normalise the derived belief mass, or in other words to ensure belief mass and uncertainty mass additivity. The use of the belief constraint operator is mathematically possible only if ω^A and ω^B are not totally conflicting, i.e., if $Con \neq 1$.

The belief constraint operator is commutative and non-idempotent. Associativity is preserved when the base rate is equal for all agents. Associativity in case of different base rates requires that all preference opinions be combined in a single operation which would require a generalisation of Def.7 for multiple agents, i.e. for multiple input arguments, which is relatively trivial. A totally indifferent opinion acts as the neutral element for belief constraint, formally expressed as:

$$\text{IF } (\omega_X^A \text{ is totally indifferent, i.e. with } u_X^A = 1) \text{ THEN } (\omega_X^A \odot \omega_X^B = \omega_X^B)\,. \tag{13}$$

Having a neutral element in the form of the totally indifferent opinion is very useful when modelling situations of preference combination.

6 Examples

6.1 Expressing Preferences with Subjective Opinions

Preferences can be expressed e.g. as soft or hard constraints, qualitative or quantitative, ordered or partially ordered etc. It is possible to specify a mapping between qualitative verbal tags and subjective opinions which enables easy solicitation of preferences [9]. Table 3 describes examples of how preferences can be expressed.

All the preference types of Table 3 can be interpreted in terms of subjective opinions, and further combined by considering them as constraints expressed by different agents. The examples that comprise two binary frames could also have been modelled with a quaternary product frame with a corresponding 4-nomial product opinion. In fact

Example & Type	Opinion Expression	
"Ingredient x is mandatory"	Binary frame	$X = \{x, \overline{x}\}$
Hard positive	Binomial opinion	$\omega_x : (1, 0, 0, \frac{1}{2})$
"Ingredient x is totally out of the question"	Binary frame	$X = \{x, \overline{x}\}$
Hard negative	Binomial opinion	$\omega_x : (0,\ 1,\ 0,\ \frac{1}{2})$
"My preference rating for x is 3 out of 10	Binary frame	$X = \{x, \overline{x}\}$
Quantitative	Binomial opinion	$\omega_x : (0.3,\ 0.7,\ 0.0,\ \frac{1}{2})$
"I prefer x or y, but z is also acceptable"	Ternary frame	$\Theta = \{x, y, z\}$
Qualitative	Trinomial opinion	$\omega_\Theta : (b(x,y) = 0.6,\ b(z) = 0.3,$ $u = 0.1,\ a(x,y,z) = \frac{1}{3})$
"I like x, but I like y even more"	Two binary frames	$X = \{x, \overline{x}\}$ and $Y = \{y, \overline{y}\}$
Positive rank	Binomial opinions	$\omega_x : (0.6,\ 0.3,\ 0.1,\ \frac{1}{2}),$ $\omega_y : (0.7,\ 0.2,\ 0.1,\ \frac{1}{2})$
"I don't like x, and I dislike y even more"	Two binary frames	$X = \{x, \overline{x}\}$ and $Y = \{y, \overline{y}\}$
Negative rank	Binomial opinions	$\omega_x : (0.3,\ 0.6,\ 0.1,\ \frac{1}{2}),$ $\omega_y : (0.2,\ 0.7,\ 0.1,\ \frac{1}{2})$
"I'm indifferent about x, y and z"	Ternary frame	$\Theta = \{x, y, z\}$
Neutral	Trinomial opinion	$\omega_\Theta : (u_\Theta = 1.0,\ a(x,y,z) = \frac{1}{3})$
"I'm indifferent but most people prefer x"	Ternary frame	$\Theta = \{x, y, z\}$
Neutral with bias	Trinomial opinion	$\omega_\Theta : (u_\Theta = 1.0,\ a(x) = 0.6,$ $a(y) = 0.2, a(z) = 0.2)$

Table 3. Example preferences and corresponding subjective opinions

product opinions over product frames could be a method of simultaneously considering preferences over multiple variables, and this will be the topic of future research.

Default base rates are specified in all but the last example which indicates total indifference but with a bias which expresses the average preference in the population. Base rates are useful in many situations, such as for default reasoning. Base rates only have an influence in case of significant indifference or uncertainty.

6.2 Going to the Cinema, 1st Attempt

Assume three friends, Alice, Bob and Clark, who want to see a film together at the cinema one evening, and that the only films showing are *Black Dust* (*BD*), *Grey Matter* (*GM*) and *White Powder* (*WP*), represented as the ternary frame $\Theta = \{BD,\ GM,\ WP\}$. Assume that the friends express their preferences in the form of the opinions of Table 4.

Alice and Bob have strong and conflicting preferences. Clark, who only does not want to watch *Black Dust*, and who is indifferent about the two other films, is not sure whether he wants to come along, so Table 4 shows the results of applying the preference combination operator, first without him, and then including in the party.

By applying the belief constraint operator Alice and Bob conclude that the only film they are both interested in seeing is *Grey Matter*. Including Clark in the party does not change that result because he is indifferent to *Grey Matter* and *White Powder* anyway, he just does not want to watch the film *Black Dust*.

		Preferences of:			Results of preference combinations:	
		Alice	Bob	Clark	(Alice & Bob)	(Alice & Bob & Clark)
		ω^A_Θ	ω^B_Θ	ω^C_Θ	$\omega^{A\&B}_\Theta$	$\omega^{A\&B\&C}_\Theta$
$b(BD)$	=	0.99	0.00	0.00	0.00	0.00
$b(GM)$	=	0.01	0.01	0.00	1.00	1.00
$b(WP)$	=	0.00	0.99	0.00	0.00	0.00
$b(GM \cup WP)$	=	0.00	0.00	1.00	0.00	0.00

Table 4. Combination of film preferences

The belief mass values of Alice and Bob in the above example are in fact equal to those of Zadeh's example [12] which was used to demonstrate the unsuitability of Dempster's rule for fusing beliefs because it produces counterintuitive results. Zadeh's example describes a medical case where two medical doctors express their opinions about possible diagnoses, which typically should have been modelled with the averaging fusion operator [6], not with Dempster's rule. In order to select the appropriate operator it is crucial to fully understand the nature of the situation to be modelled. The failure to understand that Dempster's rule does not represent an operator for cumulative or averaging belief fusion, combined with the unavailability of the general cumulative and averaging belief fusion operators for many years (1976[11]-2010[6]), has often led to inappropriate applications of Dempster's rule to cases of belief fusion [8]. However, when specifying the same numerical values as in [12] in a case of preference combination such as the example above, the belief constraint operator which is a simple extension of Dempster's rule is very suitable and produces perfectly intuitive results.

6.3 Going to the Cinema, 2nd Attempt

In this example Alice and Bob soften their strong preference by expressing some indifference in the form of $u = 0.01$, as specified by Table 5. Clark has the same opinion as in the previous example, and is still not sure whether he wants to come along, so Table 5 shows the results without and with his preference included.

		Preferences of:			Results of preference combinations:	
		Alice	Bob	Clark	(Alice & Bob)	(Alice & Bob & Clark)
		ω^A_Θ	ω^B_Θ	ω^C_Θ	$\omega^{A\&B}_\Theta$	$\omega^{A\&B\&C}_\Theta$
$b(BD)$	=	0.98	0.00	0.00	0.490	0.000
$b(GM)$	=	0.01	0.01	0.00	0.015	0.029
$b(WP)$	=	0.00	0.98	0.00	0.490	0.961
$b(GM \cup WP)$	=	0.00	0.00	1.00	0.000	0.010
u	=	0.01	0.01	0.00	0.005	0.000
$a(BD)$	=	0.6	0.6	0.6	0.6	0.6
$a(GM) = a(WP)$	=	0.2	0.2	0.2	0.2	0.2

Table 5. Combination of film preferences with some indifference and with non-default base rates

The effect of adding some indifference is that Alice and Bob should pick film *Black Dust* or *White Powder* because in both cases one of them actually prefers one of the films, and the other finds it acceptable. Neither Alice nor Bob prefers *Gray Matter*, they only find it acceptable, so it turns out not to be a good choice for any of them. When taking into consideration that the base rate $a(BD) = 0.6$ and the base rate $a(WP) = 0.2$, the preference expectation values according to Eq.(10) are such that:

$$\mathrm{E}^{A\&B}(BD) > \mathrm{E}^{A\&B}(WP)\ . \tag{14}$$

More precisely, the preference expectation values according to Eq.(10) are:

$$\mathrm{E}^{A\&B}(BD) = 0.493\ , \qquad \mathrm{E}^{A\&B}(WP) = 0.491\ . \tag{15}$$

Because of the higher base rate, *Black Dust* also has a higher expected preference than *White Powder*, so the rational choice would be to watch *Black Dust*.

However, when including Clark who does not want to watch *Black Dust*, the base rates no longer dictates the result. In this case Eq.(10) produces $\mathrm{E}^{A\&B\&C}(WP) = 0.966$ so the obvious choice is to watch *White Powder*.

6.4 Not Going to the Cinema

Assume now that the Alice and Bob express totally conflicting preferences as specified in Table 6, i.e. Alice expresses a hard preference for *Black Dust* and Bob expresses a hard preference for *White Powder*. Clark still has the same preference as before, i.e he does not want to watch *Black Dust* and is indifferent about the two other films.

	Preferences of:			Results of preference combinations:	
	Alice	Bob	Clark	(Alice & Bob)	(Alice & Bob & Clark)
	ω^A_Θ	ω^B_Θ	ω^C_Θ	$\omega^{A\&B}_\Theta$	$\omega^{A\&B\&C}_\Theta$
$b(BD)$ =	1.00	0.00	0.00	Undefined	Undefined
$b(GM)$ =	0.00	0.00	0.00	Undefined	Undefined
$b(WP)$ =	0.00	1.00	0.00	Undefined	Undefined
$b(GM \cup WP)$ =	0.00	0.00	1.00	Undefined	Undefined

Table 6. Combination of film preferences with hard and conflicting preferences

In this case the belief constraint operator can not be applied because Eq.(11) produces a division by zero. The conclusion is that the friends will not go to the cinema to see a film together. The test for detecting this situation is when *Con* = 1 in Eq.(12). It makes no difference to include Clark in the party because a conflict can not be resolved by including additional preferences. However it would have been possible for Bob and Clark to watch *White Powder* together without Alice.

7 Conclusion

The flexibility of subjective logic makes it simple to express positive and negative preferences within the same framework, as well as indifference/uncertainty. This paper describes how subjective logic can be used to express preferences over a variable represented as the possible states in a frame, and how the belief constraint operator, which is an extension of Dempster's rule, can be applied for combining preferences of multiple agents in order to determine the most preferred choice for the whole group. Because preference can be expressed over arbitrary subsets of the frame this is in fact a multi-polar model for expressing and combining preferences. Even in the case of no overlapping focal elements the belief constraint operator provides a meaningful answer, namely that the preferences are incompatible.

Multi-agent preference combination with subjective logic assumes that individual preferences have been predefined. Future research will focus on applying subjective logic for determining subjective preferences of each agent e.g. in situations with multiple criteria, and on combining preferences from multiple agents over different variables in the form of different frames.

References

1. Stefano Bistarelli, Maria Silvia Pini, Francesca Rossi, and K. Brent Venable. Uncertainty in bipolar preference problems. In *Proceedings of the 13th international conference on Principles and practice of constraint programming*, CP'07, pages 782–789, 2007.
2. Stefano Bistarelli, Maria Silvia Pini, and K. Brent Venable. Positive and negative preferences. In *Proceedings of the 7th International Workshop on Preferences and Soft Constraints*, Sitges, Spain, October 2005.
3. Didier Dubois, Hélène Fargier, and Henri Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.
4. M. Fitting. Kleene's three-valued logics and their children. *Fundamenta Informaticae*, 20:113–131, 1994.
5. A. Jøsang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, June 2001.
6. A. Jøsang, J. Diaz, and M. Rifqi. Cumulative and Averaging Fusion of Beliefs. *Information Fusion*, 11(2):192–200, 2010. doi:10.1016/j.inffus.2009.05.005.
7. A. Jøsang and D. McAnally. Multiplication and Comultiplication of Beliefs. *International Journal of Approximate Reasoning*, 38(1):19–51, 2004.
8. A. Jøsang and S. Pope. Dempster's Rule as Seen by Little Coloured Balls (in press). *Computational Intelligence Journal*, 2011. Available from http://folk.uio.no/josang/.
9. Simon Pope and Audun Jøsang. Analsysis of Competing Hypotheses using Subjective Logic. In *Proceedings of the 10th International Command and Control Research and Technology Symposium (ICCRTS)*. United States Department of Defense Command and Control Research Program (DoDCCRP), 2005.
10. Thomas Schiex. Possibilistic constraint satisfaction problems or "how to handle soft constraints ?". In *In Proc. 8th Conf. of Uncertainty in AI*, pages 269–275, 1992.
11. G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.
12. L.A. Zadeh. Review of Shafer's A Mathematical Theory of Evidence. *AI Magazine*, 5:81–83, 1984.

Domain permutation reduction for Valued CSPs

Maher Helaoui[1] and Wady Naanaa[2]

[1] Pôle de Recherche en Informatique du Centre (PRINCE), Institut Supérieur d'Informatique et des Techniques de Communication, Hammam-Sousse, Tunisia
maher.helaoui@gmail.com
[2] Faculty of sciences, University of Monastir, Tunisia
Wady.Naanaa@fsm.rnu.tn

Abstract. Several combinatorial problems can be formulated as Valued Constraint Satisfaction Problems (VCSPs) where constraints are defined through the use of valuation functions to reflect degrees of coherence. Despite the NP-hardness of solving VCSPs in general, tractable versions can be obtained by forcing the allowable valuation functions to have specific features. This is the case, for instance, of VCSPs with submodular valuation functions [16].
In this paper, we are concerned with a problem that generalizes submodular binary VCSP, called *permuted submodular binary* VCSP. The latter problem is obtained by independently applying permutations on the domains of submodular binary VCSP. We show that instances in the permuted binary VCSP class can be identified and solved in polynomial time provided that the employed value domain is bounded and the valuation functions are prime.

1 Introduction

Constraint Satisfaction Problems (CSPs) provide a general and convenient framework to model and solve numerous combinatorial problems including planning and scheduling. In the standard CSP framework, the constraints are defined by crisp relations, which specify the consistent combinations of values. However, in real-world situations, one may need to express various degrees of consistency in order to reflect the specificity of the problem at hand. The valued constraint satisfaction problems (VCSPs) approach [18] is intended to model such situations. Basically, a VCSP consists of a set of variables taking values in discrete sets called *domains*. A valued constraint is defined through the use of a valuation function, which associates a degree of desirability to each combination of values. The problem is to find an assignment of values to variables from their respective domains with a finite and optimal global valuation. Finding such an assignment or proving that none exists is known to be an NP-hard task [3].

The computational complexity of finding the optimal solution to a VCSP has been studied in many works and several classes of tractable VCSPs have been identified and solved [4, 11, 3]. Recall that a problem is said to be tractable if and only if there exists a polynomial-time algorithm that solves it. Binary VCSP with submodular binary valuation functions is one of these tractable class. By expressing VCSP instances with submodular binary functions as the problem of finding a minimum weighted cut of a weighted directed graph, it is possible to solve them only in $O(n^3d^3)$ steps, where n is the number of variables and d is the size of the largest value domain [2].

Nonetheless, VCSPs resulting from modeling real situations are rarely limited to submodular functions. In such cases, can we proceed in a more efficient manner than an exhaustive search while keeping solution optimality? Is there any mean to exploit submodularity in a less restrictive context? This paper is intended to contribute to providing positive answers to these questions. More precisely, we are concerned with a problem generalizing submodular binary VCSP, called *permuted submodular binary* VCSP. The latter problem is obtained by independently applying permutations on the value domains of submodular binary VCSP.

In [17], the author proposed a polynomial-time algorithm for identifying permuted submodular binary VCSPs which works only with finite valuations and assumes that the operator employed to aggregate local valuations into a global one is strictly monotonic. Our aim is to avoid these two limitations, because infinite valuations is the more natural mean to express unsatisfiability and this latter notion is essential in modeling real-world problems. Furthermore, the valuations needed to encode certain problems are bounded by a constant, which imposes a non strictly monotonic aggregation operator. This occurs, for instance, when modeling fuzzy or probabilistic systems.

Our approach is inspired by the one proposed in [9], which consists in transforming a crisp CSP instances into instances in the max-closed tractable CSP class [10]. If the identification of the transformable instances and the transformation process are tractable then the overall solution process for the permuted max-closed CSP is tractable. In our case, the target tractable VCSP is the submodular binary VCSP. The transformation process consists in performing permutations on the value domains of the instances to be solved, so that the resulting instances fall into the submodular binary VCSP class. The transformed instances are polynomially solved by dedicated algorithms [2, 16], and then a solution to the original instances can be deduced by applying the reverse permutations to the obtained solution. Of course, only instances in the permuted submodular binary VCSP class could be transformed in this way.

In this paper, we show that instances in the permuted submodular binary VCSPs class can be polynomially identified and transformed into instances in the submodular binary VCSPs class provided that the involved value domain is bounded and the valuation functions are prime. In compensation, the proposed approach does not impose restrictive conditions on the valuations used or on the way these valuations are aggregated.

The paper is organized as follows: the next section introduces some definitions and notations. Section 3, is devoted to specifying the particular VCSPs we are concerned with. In Section 4, we describe how the domain permutation theory proposed in [9] is extended in order to efficiently solve the VCSPs specified in Section 3. Some related works are discussed in Section 5. We conclude in Section 6.

2 Definitions and notations

In the valued CSP framework (VCSP) [18], the set of possible valuations E is assumed to be a totally ordered set with a minimum ($\perp$) and a maximum ($\top$) element, equipped with a single monotonic binary operation $\oplus$ known as *aggregation*. These assumptions can be gathered in a valuation structure that can be specified as follows:

Definition 1. *A valuation structure is defined as a tuple $S = (E, \oplus, \preceq)$ such that:*

- *E is a set of valuations;*
- *$\preceq$ is a total order on E;*
- *$\oplus$ is a binary commutative, associative and monotonic operator.*

An aggregation operator $\oplus$ is said to be *strictly monotonic* if for all α, β, γ in E such that $\alpha \prec \beta$ and $\gamma \neq \top$, we have $\alpha \oplus \gamma \prec \beta \oplus \gamma$.

Contrary to many widely used valuation structures, the valuation structure employed in this paper does not suppose a strictly monotonic aggregation operator. This is a crucial advantage, because this makes the proposed approach effective in dealing with many specific VCSP like Weighted CSP (WCSP) and Fuzzy CSP (FCSP).

Once the valuation structure is specified, we define the valued constraint satisfaction problem (VCSP) we are concerned with as follows:

Definition 2. *A valued constraint satisfaction problem (VCSP) instance is defined by a tuple (X, D, C, S) such that:*

- *X is a finite set of variables;*
- *D is a finite set called the domain of the instance;*
- *$S = (E, \oplus, \preceq)$ is a valuation structure which matches Definition 1.*
- *C is a set of valued constraints. Each valued constraint c is an ordered pair (σ, ϕ) where $\sigma \subseteq X$ is the scope of c and ϕ is a function from $D^{|\sigma|}$ to E.*

The *arity* of a valued constraint is the size of its scope. The arity of a problem is the maximum arity over all its constraints. In this work, we are mainly concerned with binary VCSPs, that is, VCSPs with unary and binary constraints only. Furthermore, the scopes of binary constraints are assumed to be ordered tuples. Hence, the constraint $c = (\langle x_i, x_j \rangle, \phi)$ differs from $c^T = (\langle x_j, x_i \rangle, \phi^T)$, where ϕ^T, the transpose[3] of ϕ, is such that $\phi^T(u,v) = \phi(v,u)$, for all $u, v \in D$. Of course, if c or c^T is in C then both of them is in C.

The valuation of an assignment t to a subset of variables $V \subseteq X$ is obtained by

$$\Phi_P(t) \quad = \quad \sum_{(\sigma,\phi)\in\vec{C},\sigma\subseteq V} \phi(t \downarrow \sigma) \tag{1}$$

where $t \downarrow \sigma$ denotes the projection of t on the variables of σ and $\vec{C}$ is the subset $\{(\langle x_i, x_j\rangle, \phi) \in C \mid i < j\}$. Hence, an overall optimal solution for a VCSP instance P on n variables is an n-tuple t such that $\Phi_P(t)$ is finite and minimal over all possible n-tuples.

In this paper, we are also led to cope with (crisp) constraint satisfaction problem (CSP), which is a VCSP with a valuation set E limited to $\{0, \infty\}$. In this special case and since only solutions with finite costs are relevant, we can replace every valuation function with a relation that exactly contains those tuples that have zero cost. The valuation structure S is, thereby, not necessary any more. Consequently, a CSP can be defined by a triple (X, D, C), where every (crisp) constraint is defined by a scope and a relation specifying the tuples of values allowed by the constraint.

Since only one valuation set (E) is used throughout the paper, we will simply write $\mathcal{F}_D$ to designate the set of all functions from D^2 to E.

The valuation functions employed in defining the VCSPs considered in this paper are elements of $\mathcal{F}_D$. These elements can be limited to specific subsets of $\mathcal{F}_D$ in order to obtain tractable VCSPs. Restricting the functions employed in defining a VCSP is captured in the notion language. Hence, a valued binary constraint language over domain D is any subset of $\mathcal{F}_D$. For any valued constraint language $\mathcal{G}_D \subseteq \mathcal{F}_D$, we will refer to the set of all VCSP instances with valuation functions in $\mathcal{G}_D$ by VCSP($\mathcal{G}_D$).

We also use the bijective binary CSP, a tractable binary CSP class which can be defined as follows. Let R be a binary relation.

- R is said to be functional on its first argument if and only if whenever $\langle u, v\rangle$ and $\langle u', v\rangle$ are in R, we have $u = u'$.
- R is said to be functional on its second argument if and only if whenever $\langle u, v\rangle$ and $\langle u, v'\rangle$ are in R, we have $v = v'$.
- R is bijective if and only if it is functional on both of its arguments.

[3] We use the term *transpose* because, when binary functions are stored in matrices, the matrix associated to ϕ^T is the transpose of the matrix associated to ϕ.

Denote by $\textsc{funct}1_V$ (resp. $\textsc{funct}2_V$) the (crisp) constraint language of all binary relations over domain V that are functional on their first (resp. second) argument and by $\textsc{biject}_V$ the valued constraint language of all bijective binary relations over domain V. Hence, CSP($\textsc{biject}_V$) will refer to the set of all binary CSP instances with relation in $\textsc{biject}_V$.

3 Modularity related VCSPs

To specify the VCSPs we are concerned with, we need to introduce the following two predicates. Let ϕ be any function of $\mathcal{F}_D$, we associate to ϕ the quaternary predicates mod_ϕ and submod_ϕ that are defined as follows

$$\begin{aligned} \text{mod}_\phi(u,u',v,v') &\overset{\text{def}}{\Leftrightarrow} & \phi(u,u') \oplus \phi(v,v') &= \phi(u,v') \oplus \phi(v,u') \\ \text{submod}_\phi(u,u',v,v') &\overset{\text{def}}{\Leftrightarrow} & \phi(u,u') \oplus \phi(v,v') &\preceq \phi(u,v') \oplus \phi(v,u') \end{aligned}$$

where u, v, u', v' are any elements of D.

Definition 3. *A binary function $\phi \in \mathcal{F}_D$ is* submodular *if and only if submod$_\phi(u,u',v,v')$ holds for all $u, v, u', v' \in D$ such that $u < v$ and $u' < v'$.*

Example 1. For any $m \in \mathbb{N}^* \cup \{+\infty\}$ and any integer $\lambda \geq -2$, consider the valuation structure $S_{m,\lambda} = (E, \oplus, \leq)$ where the valuation set is

$$E = \{0, \ldots, (2+\bar{\lambda})(m-1)^2\}$$

where $\bar{\lambda} = \max(0, \lambda)$ and the aggregation operator $\oplus$ is defined by

$$\alpha \oplus \beta = \min(\alpha + \beta, (2+\bar{\lambda})(m-1)^2)$$

The $\oplus$ operator is not always strictly monotonic since, if $m \geq 3$, then

$$(2+\bar{\lambda})(m-2)^2 < (2+\bar{\lambda})(m-1)^2$$

and

$$(2+\bar{\lambda})(2m-3) < (2+\bar{\lambda})(m-1)^2$$

but

$$(2+\bar{\lambda})(m-2)^2 \oplus (2+\bar{\lambda})(2m-3) = (2+\bar{\lambda})(m-1)^2 \oplus (2+\bar{\lambda})(2m-3)$$

Consider the family of binary functions f_λ defined on the domain $D = \{0, \ldots, m-1\}$ by

$$f_\lambda(u,v) = u^2 + v^2 + \lambda uv$$

We show that f_λ is submodular if and only if $\lambda \leq 0$.

Let $u, v, u', v' \in D$ such that $u < v,\ u' < v'$. By observing that $\lambda uu' + \lambda vv' \leq \lambda uv' + \lambda vu'$ for all $\lambda \leq 0$, we obtain

$$\begin{aligned} f_\lambda(u, u') \oplus f_\lambda(v, v') &= (u^2 + u'^2 + \lambda uu') \oplus (v^2 + v'^2 + \lambda vv') \\ &= \min(u^2 + u'^2 + \lambda uu' + v^2 + v'^2 + \lambda vv', (2 + \bar{\lambda})(m-1)^2) \\ &\leq \min(u^2 + u'^2 + \lambda uv' + v^2 + v'^2 + \lambda vu', (2 + \bar{\lambda})(m-1)^2) \\ &\leq f_\lambda(u, v') \oplus f_\lambda(v, u') \end{aligned}$$

On the other hand, if $\lambda > 0$ then we have $\neg\mathrm{mod}_{\phi_\lambda}(0, 0, 1, 1)$, since

$$f_\lambda(0, 0) \oplus f_\lambda(1, 1) = 2 + \lambda > 2 = f_\lambda(0, 1) \oplus f_\lambda(1, 0)$$

Hence, if $\lambda > 0$ then f_λ is not submodular. □

Submodular functions of any arity are widely studied because they are involved in many tractable discrete optimization problems [14, 15, 19, 20, 22].

Denote by SUBMOD$_D$ the language of all submodular functions of $\mathcal{F}_D$. Hence, VCSP(SUBMOD$_D$) will designate the class of VCSPs with valuation functions in SUBMOD$_D$. This VCSP class will be called the class of submodular VCSP instances.

Definition 4. *Let ϕ be in $\mathcal{F}_D$. Two values $u, v \in D$ are said to be* modular *with regard to (w.r.t) ϕ if and only if $\mathit{mod}_\phi(u, u', v, v')$ holds for all $u', v' \in D$.*

Example 2. Consider the family of binary function f_λ defined in Example 1. All the values of D are pairwise modular w.r.t f_0 since, for all $u, v, u', v' \in D$, we have

$$\begin{aligned} f_0(u, u') \oplus f_0(v, v') &= (u^2 + u'^2) \oplus (v^2 + v'^2) \\ &= \min(u^2 + u'^2 + v^2 + v'^2, (2 + \bar{\lambda})(m-1)^2) \\ &= (u^2 + v'^2) \oplus (v^2 + u'^2) \\ &= f_0(u, v') \oplus f_0(v, u') \end{aligned}$$

□

Definition 5. *Let ϕ be in $\mathcal{F}_D$. ϕ is said to be* prime *if and only if there is no modular values in D w.r.t. ϕ.*

Example 3. Consider again the family of binary function f_λ defined in Example 1. Then f_λ is prime for all $m \geq 2$ and $\lambda \neq 0$. Indeed, we have $\neg\mathrm{mod}_{f_\lambda}(u, 0, v, 1)$ for all $u, v \in D,\ u < v$, since

$$
\begin{aligned}
f_\lambda(u,0) \oplus f_\lambda(v,1) &= u^2 \oplus (v^2 + 1 + \lambda v) \\
&= u^2 + v^2 + 1 + \lambda v \\
&\neq u^2 + v^2 + 1 + \lambda u \\
&\neq (u^2 + 1 + \lambda u) \oplus v^2 \\
&\neq f_\lambda(u,1) \oplus f_\lambda(v,0)
\end{aligned}
$$

The latter equalities and inequalities hold because $u^2+v^2+1+\lambda v \leq (2+\bar{\lambda})(m-1)^2$ and $u^2+v^2+1+\lambda u \leq (2+\bar{\lambda})(m-1)^2$. □

Denote by $\textsc{prime}_D$ the language of all prime functions of $\mathcal{F}_D$. VCSP($\textsc{prime}_D$) will designate the class of VCSPs with valuation functions in $\textsc{prime}_D$. This VCSP class will be called the class of prime VCSP instances.

4 Domain permutation for VCSP

The VCSP($\textsc{submod}_D$) class highlighted in the previous section is essential to our present work because it is known to be tractable [2]. In other respects, in [9], the authors proposed a theory relying on domain permutations whose aim is to transform (crisp) CSP instances into CSP instances over a tractable constraint language, namely the max-closed constraint language [10].

In this section, we study the problem of discovering domain permutations that transform binary VCSP instances into instances in the VCSP($\textsc{submod}_D$) class. We call the set of all binary VCSP instances for which such a transformation exists *permuted submodular binary VCSP*. The tool that will be used to discover the required permutations is the domain permutation reduction theory [9], which is adapted and extended to cope with valued constraints.

4.1 Definitions

We begin by presenting the definitions needed for applying the domain permutation reduction theory to the VCSP framework.

Denote by Π_D the set of all permutations of D. Let $\langle \pi, \pi' \rangle$ be an ordered pair of permutations of D, that is, $\langle \pi, \pi' \rangle \in \Pi_D^2$ and let (v, v') be an ordered pair of D^2. Applying $\langle \pi, \pi' \rangle$ to (v, v') results in the ordered pair $\langle \pi, \pi' \rangle (v, v') = (\pi(v), \pi'(v'))$ and applying $\langle \pi, \pi' \rangle$ to a binary function $\phi \in \mathcal{F}_D$, yields the binary function $\langle \pi, \pi' \rangle \phi$ defined by

$$\langle \pi, \pi' \rangle \phi(v, v') = \phi(\pi^{-1}(v), \pi'^{-1}(v')), \qquad \text{for all } (v, v') \in D^2 \tag{2}$$

Definition 6. *Let $P = (X, D, C, S)$ be a VCSP instance. A* domain permutation *for P is a mapping Π assigning to each variable $x \in X$ a permutation from Π_D.*

– *For any scope $\sigma = \langle x_i, x_j \rangle$, we define $\Pi(\sigma) = \langle \Pi(x_i), \Pi(x_j) \rangle \in \Pi_D^2$.*
– *For any valued constraint $(\sigma, \phi) \in C$, we define $\Pi(\sigma, \phi) = (\sigma, \Pi(\sigma)\phi)$, where $\Pi(\sigma)\phi$ is defined by (2).*
– *The permuted constraint set is given by $\Pi(C) = \{\Pi(\sigma, \phi) \mid (\sigma, \rho) \in C\}$.*
– *The permuted instance is defined by $\Pi(P) = (X, D, \Pi(C), S)$*

Let P be a VCSP instance. If there exists a domain permutation Π for P such that $\Pi(P)$ is in VCSP($\textsc{submod}_D$) then we say that P is *reductible* to $\textsc{submod}_D$. Our aim is to determine whether a given VCSP instance is deductible to $\textsc{submod}_D$. This problem is called *the reduction problem into* $\textsc{submod}_D$ and resolving it efficiently is the key to extending the VCSP($\textsc{submod}_D$) tractable class. To express this problem in the constraint satisfaction framework, we need to adapt the notion of a *lifted relation*, (see Definition 20 of [9]), in order to handle valued constraints.

Definition 7. *We define the relation lifting any function $\phi \in \mathcal{F}_D$ into $\textsc{submod}_D$, $\rho(\phi, \textsc{submod}_D)$, to be the following binary relation over Π_D:*

$$\rho(\phi, \textsc{submod}_D) = \{\langle \pi, \pi' \rangle \in \Pi_D^2 \mid \langle \pi, \pi' \rangle\, \phi \in \textsc{submod}_D\}$$

Example 4. Consider again the family of binary functions f_λ defined in Example 1. We focus on the prime functions, then we assume that $\lambda \neq 0$. Furthermore, for the simplicity of the example, we take $m = 2$, then $D = \{0, 1\}$. There is two permutations on D: the identity permutation: $\iota = (0, 1)$, and the reverse permutation: $\tau = (1, 0)$.

We saw that the f_λ's are prime and submodular if and only if $\lambda < 0$. If we apply the reverse permutation to the domains of both variables of a prime and submodular function, we get a prime and submodular function. Conversely, if we apply the reverse permutation to exactly one of the domains of a prime and submodular function, we loose submodularity. Then the relation lifting the function $f_\lambda,\ \lambda < 0$ in $\textsc{submod}_D$, $\rho(f_{\lambda<0}, \textsc{submod}_D)$, is

$$\rho^- = \{\langle \iota, \iota \rangle, \langle \tau, \tau \rangle\}$$

On the other hand, the functions $f_\lambda,\ \lambda > 0$ are called *supermodular*. They are closely related to submodular functions since they verify the inequality obtained by inverting the comparison operator in the definition of the submodularity predicate. It is easy to transform a supermodular binary function on $D = \{0, 1\}$ into submodular one by means of permutations. It suffices to apply the identity permutation to the domain of one of the variables and apply the reverse permutation to the domain of the other variable. Then, the relation lifting the function $f_\lambda,\ \lambda > 0$ in $\textsc{submod}_D$, $\rho(f_\lambda > 0, \textsc{submod}_D)$, is

$$\rho^+ = \{\langle \iota, \tau \rangle, \langle \tau, \iota \rangle\}$$

□

4.2 Reducing prime VCSPs

Our aim is to show that the reduction problem into $\textsc{submod}_D$ is tractable if the considered VCSP instance is defined on a bounded value domain and valuation function in the $\textsc{prime}_D$ valued constraint language.

Definition 8. *The lifted (crisp) constraint language of* $\textsc{prime}_D$ *into* $\textsc{submod}_D$ *is* $\mathcal{L}(\textsc{prime}_D, \textsc{submod}_D) = \{\rho(\phi, \textsc{submod}_D) \mid \phi \in \textsc{prime}_D\}$.

Definition 9. *Let* $P = (X, D, C, S)$ *be a VCSP instance. We define the lifted instance for* P *into* $\textsc{submod}_D$, $\mathcal{L}(P, \textsc{submod}_D)$, *to be the (crisp) CSP instance* $(X, \Pi_D, \{(\sigma, \rho(\phi, \textsc{submod}_D)) \mid (\sigma, \phi) \in C\})$.

Example 5. Consider the binary VCSP instance $P = (X, D, C, S)$, on the variables $X = \{x_1, x_2, x_3, x_4\}$, with the value domain $D = \{0, 1\}$, the valuation structure used is the one described in Example 1, except that we fix m to 2. All the possible binary constraints are present in the instance (see Fig. 1-Left). Their valuation functions are defined as follows:

$$\phi_{i,j} = \begin{cases} f_{-1} & \text{if } i \text{ and } j \text{ have the same parity} \\ f_1 & \text{otherwise} \end{cases}$$

where f_1 and f_{-1} are define as in Example 1. It follows that $c_{1,3}$ and $c_{2,4}$ are submodular while $c_{1,2},\ c_{2,3},\ c_{3,4}$ and $c_{1,4}$ are supermodular[4].

The lifted instance for P into $\textsc{submod}_D$ is, therefore, a crisp CSP on four variables with value domain $\Pi_D = \{\iota, \tau\}$. It contains all the possible binary constraints on four variables (see Fig 1-Right) and, in accordance with Example 4, the relations constraining the different pairs of variables are defined as follows

$$\rho_{i,j} = \begin{cases} \rho^{-} & \text{if } i \text{ and } j \text{ have the same parity} \\ \rho^{+} & \text{otherwise} \end{cases}$$

Note that the resulting CSP instance is consistent since the assignment $(\iota, \tau, \iota, \tau)$ satisfies all the constraints. This means that the initial VCSP instance is reductible to $\textsc{submod}_D$. □

Lemma 1. $\mathcal{L}(\textsc{prime}_D, \textsc{submod}_D) \subseteq \textsc{Funct1}_{\Pi_D}$.

Proof. We have to prove that every relation in $\mathcal{L}(\textsc{prime}_D, \textsc{submod}_D)$ is functional on its first argument. To this end, we assume that there exists ρ in $\mathcal{L}(\textsc{prime}_D, \textsc{submod}_D)$ such that ρ is not functional on its first argument and proceed to get a contradiction. If ρ is not functional on its first argument then there must exist $\langle \pi_1, \pi \rangle, \langle \pi_2, \pi \rangle \in \rho$ such that $\pi_1 \neq \pi_2$. This implies that there exists $u, v \in D, u < v$ such that

[4] In fact, the $c_{i,j}$'s such that $i > j$ are also present in the VCSP.

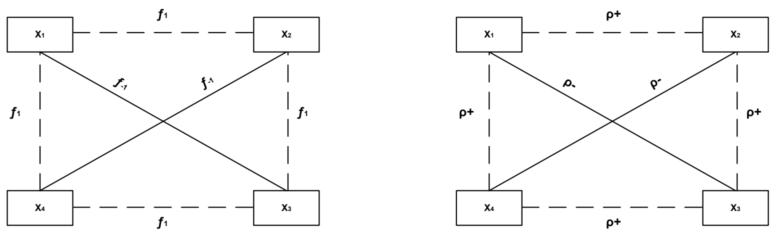


Fig. 1. The constraint graph of the four-variables VCSP instance given in Example 5 (Left) and the constraint graph of its lifted CSP instance (Right).

$$(\pi_1(u) < \pi_1(v) \wedge \pi_2(u) > \pi_2(v)) \ \vee \ (\pi_1(u) > \pi_1(v) \wedge \pi_2(u) < \pi_2(v))$$

Assume, without loss of generality, that the left-hand side of the disjunction holds. On the other hand, $\rho \in \mathcal{L}(\textsc{prime}_D, \textsc{submod}_D)$ implies that there exists $\phi \in \textsc{prime}_D$ such that $\rho = \rho(\phi, \textsc{submod}_D)$. Furthermore, since $\phi \in \textsc{prime}_D$, there must exist $u' < v'$ such that $\neg\text{mod}_\phi(u, u', v, v')$, otherwise, u and v would be modular and therefore ϕ would not be prime. Here, we distinguish two cases:

- $\pi(u') < \pi(v')$: let $\phi_1 = \langle \pi_1, \pi \rangle \, \phi$. Since $\langle \pi_1, \pi \rangle \in \rho$, we must have $\phi_1 \in \textsc{submod}_D$, and then $\text{submod}_{\phi_1}(\pi_1(u), \pi(u'), \pi_1(v), \pi(v'))$ must be true. By (2), we obtain $\text{submod}_\phi(u, u', v, v')$. Similarly, let $\phi_2 = \langle \pi_2, \pi \rangle \, \phi$. Since $\langle \pi_2, \pi \rangle \in \rho$, we must have $\phi_2 \in \textsc{submod}_D$, and then $\text{submod}_{\phi_2}(\pi_2(v), \pi(u'), \pi_2(u), \pi(v'))$. By (2), we obtain $\text{submod}_\phi(v, u', u, v')$. But $\text{submod}_\phi(u, u', v, v')$ and $\text{submod}_\phi(v, u', u, v')$ yield $\text{mod}_\phi(u, u', v, v')$, thus a contradiction.
- $\pi(u') > \pi(v')$: we can proceed in the same manner as in the first case to deduce the same contradiction.

Hence, every relation in $\mathcal{L}(\textsc{prime}_D, \textsc{submod}_D)$ is functional on its first argument. □

Lemma 1 is the first stage for proving that the lifted instance into $\textsc{submod}_D$ of a VCSP($\textsc{prime}_D$) instance is built from a particular constraint language, namely, the $\textsc{biject}_{\Pi_D}$ constraint language. To derive this result, we need to establish the following two identities.

Lemma 2. *Let $\phi \in \mathcal{F}_D$ and let $\pi, \pi' \in \Pi_D$, then $(\langle \pi, \pi' \rangle \, \phi)^T = \langle \pi', \pi \rangle \, \phi^T$.*

Proof. For all $u, v \in D$, we have [5]

$$\begin{aligned}
(\langle \pi, \pi' \rangle \, \phi)^T(u, v) &= \langle \pi, \pi' \rangle \, \phi(v, u) \\
&= \phi(\pi^{-1}(v), \pi'^{-1}(u)) \\
&= \phi^T(\pi'^{-1}(u), \pi^{-1}(v)) \\
&= \langle \pi', \pi \rangle \, \phi^T(u, v)
\end{aligned}$$

□

Lemma 3. *For all $\phi \in \mathcal{F}_D$, we have $\rho(\phi, \textsc{submod}_D) = \rho(\phi^T, \textsc{submod}_D)^T$.*

Proof. Exploiting the fact that the transpose of a submodular binary function is also submodular and using the identity given in Lemma 2, we obtain

$$\begin{aligned}
\rho(\phi, \textsc{submod}_D) &= \{\langle \pi, \pi' \rangle \in \mathit{\Pi}_D^2 \mid \langle \pi, \pi' \rangle \, \phi \in \textsc{submod}_D\} \\
&= \{\langle \pi, \pi' \rangle \in \mathit{\Pi}_D^2 \mid (\langle \pi, \pi' \rangle \, \phi)^T \in \textsc{submod}_D\} \\
&= \{\langle \pi, \pi' \rangle \in \mathit{\Pi}_D^2 \mid \langle \pi', \pi \rangle \, \phi^T \in \textsc{submod}_D\} \\
&= \{\langle \pi', \pi \rangle \in \mathit{\Pi}_D^2 \mid \langle \pi', \pi \rangle \, \phi^T \in \textsc{submod}_D\}^T \\
&= \rho(\phi^T, \textsc{submod}_D)^T
\end{aligned}$$

□

Theorem 1. *If $P \in \mathrm{VCSP}(\textsc{prime}_D)$ then $\mathcal{L}(P, \textsc{submod}_D) \in \mathrm{CSP}(\textsc{biject}_{\mathit{\Pi}_D})$.*

Proof. Let $P = (X, D, C, S)$ be in $\mathrm{VCSP}(\textsc{prime}_D)$. We have to prove that every constraint $\rho(\phi, \textsc{submod}_D)$ of $\mathcal{L}(P, \textsc{submod}_D)$ is in $\textsc{biject}_{\mathit{\Pi}_D}$.

Let $(\sigma, \phi) \in C$. Since P is in $\mathrm{VCSP}(\textsc{prime}_D)$, ϕ must be in $\textsc{prime}_D$. By Lemma 1, this guarantees that $\rho(\phi, \textsc{submod}_D)$ is in $\textsc{funct1}_{\mathit{\Pi}_D}$. On the other hand, $(\sigma, \phi) \in C$ implies $(\sigma^T, \phi^T) \in C$, where σ^T is σ in the reverse order. It follows that ϕ^T is also in $\textsc{prime}_D$. Again by Lemma 1, this guarantees that $\rho(\phi^T, \textsc{submod}_D) \in \textsc{funct1}_{\mathit{\Pi}_D}$, and then $\rho(\phi^T, \textsc{submod}_D)^T \in \textsc{funct2}_{\mathit{\Pi}_D}$. According to Lemma 3, we obtain $\rho(\phi, \textsc{submod}_D) \in \textsc{funct2}_{\mathit{\Pi}_D}$, and since we have already proved that $\rho(\phi, \textsc{submod}_D) \in \textsc{funct1}_{\mathit{\Pi}_D}$, we get $\rho(\phi, \textsc{submod}_D) \in \textsc{biject}_{\mathit{\Pi}_D}$. Hence, the result. □

In [21], the authors proposed a $O(|X|^2|D|)$ algorithm for solving bijective binary CSP. Hence, the lifted CSP instance resulting from the reduction into $\textsc{submod}_D$ of a $\mathrm{VCSP}(\textsc{prime}_D)$ instance, can be solved in $O(|X|^2|\mathit{\Pi}_D|)$ steps. Unfortunately, this time complexity is not polynomial on the size of the VCSP instance to be reduced, because $|\mathit{\Pi}_D| = |D|!$ is factorial, unless $|D|$ is in $O(1)$, which implies that $|D|!$ is also in $O(1)$. Hence, assuming a bounded value domain, Theorem 1 suggests that the reduction problem into $\textsc{submod}_D$ is $O(|X|^2)$

[5] We assume that the transpose operation has more priority than applying permutations to functions.

solvable for binary VCSP($\textsc{prime}_D$) instances.

Next, we show that it is tractable to determine whether a given binary VCSP instance is prime or not. This can be done by checking whether there is any modular pair of value with regard to any valuation function used in the instance. This can be achieved in $O(|X|^2)$ since the number of valued binary constraints is in $O(|X^2|)$ and we assumed that $|D| \in O(1)$. Hence the identifiability problem for the class of permuted submodular binary VCSP with a bounded domain and prime valuation functions is tractable.

For the solution algorithm, we can use the one proposed in [2] which performs in $O(|X|^3)$ on binary VCSP with a bounded domain. Finally, to retrieve a solution to the initial VCSP instance, we need to compute the reverse of every domain permutation obtained by solving the lifted CSP instance. This can be done in $O(|X|)$ steps. Hence, the overall solution process for permuted submodular binary VCSP with a bounded domain and prime valuation functions is tractable.

4.3 Domain permutation for VCSP decomposition

The approach described above may fail to find a set of domain permutations that transforms a given binary VCSP instance into an instance whose valuation functions are all submodular. This simply occurs when the instance at hand is not in the permuted binary VCSP class. Nevertheless, the idea of domain pemutation could be employed to derive a problem decomposition schema that can be integrated into a solution algorithm in order to accelerate problem solving. We have already presented such a decomposition schema [13], except that, in the latter work, the target valued constraint language was that of modular binary functions. Intuitively, a decomposition schema based on the superclass of submodular functions is more attractive.

In what follows, we briefly describe a decomposition schema whose main is to decompose any binary VCSP instance into a set of subproblems that involve submodular valuation functions only. This schema can be summarized is the following two steps:

- As a first step, chose an arbitrary ordering [6] for the value domain considered at the root of the search tree, keep this domain unchanged and move to the other domains.

- At each non-root node of the search tree, reduce the current value domain so that the remaining values can be ordered in such a way that renders every valuation function connecting the current variable to a preceding one

[6] In the present section, we rather use the more generic term *ordering* instead of *permutation*.

submodular. The ordering of the current domain is (in part) deduced from those of the preceding domains, because the latter orderings may induce the relative ordering of many value pairs in the current domain. Ideally, a total ordering of the current value domain is induced. In that case, the current domain is not reduced, i.e., it keeps all its values. But, due to domain reductions and constraint propagations, it may occur that the induced ordering is only partial, even if all the valuation functions are initially prime. In such a situation, the partial order should be (arbitrarily) completed into a total one. Conversely, the orderings of the preceding domains may induce incompatible orderings on some of the value pairs of the current domain. In such a situation, the current domain must be reduced so that only one value from each problematic pair is left in the domain. In the extreme case, the current domain will be reduced to a single value to meet the submodularity requirement.

The execution of these two steps produces, at each leaf of the search tree, a submodular and then tractable subproblem that can be solved by the algorithm proposed in [2].

We have already carried out many experiments with a solution algorithm that integrates the decomposition schema described above to a standard solution algorithm for weighted CSPs: MAC-star [12]. We have also compared the performances of the resulting solution algorithm to those of the standard MAC-star and the version of MAC-star proposed in [13]. These experiments showed that the version using the present decomposition schema is very competitive with the two others[7].

5 Related work

In [9], the authors proposed a theory whose aim is to extend the max-closed CSP tractable class. Their approach consists in discovering permutations which, when applied to every value domain, result in a max-closed CSP instance. If such permutations exist, the permuted instance is solved in polynomial time and then a solution to the original instance is deduced by simply inverting the permutations. The primary advantage of the approach is that the search for domain permutations is itself expressed as a CSP referred to as the *lifted constraint instance*. In some cases, the latter CSP has the advantage of being tractable. This is the case for bounded arity CSP with Boolean domain. Nonetheless, the authors have proved that the reduction problem into the max-closed constraint language is NP-complete even for binary CSPs with three-valued domains.

The idea of domain permutation was also used by E. Chen *et al* in view of transforming binary CSPs into the connected row convex (CRC) tractable

[7] We have not reported the details of the algorithm and the experimentation in the present paper for lack of space.

class [1]. But it appeared that their algorithm recognizes only a subset of the binary CSP instances that can be transformed into the CRC class. In fact, it has been proved that the permuted connected row convex identification problem is intractable for domains of size four or more [9].

Some soft local consistency algorithms, like optimal soft arc consistency (OSAC) [6] and virtual arc consistency (VAC) [7], can solve submodular VCSPs even on non-binary VCSPs. However, the aggregation operator employed by these algorithms is assumed to be strictly monotonic and the permutations that transforms the input VCSP instance into a submodular instance cannot be obtained.

In [17] Schlesinger proposed a polynomial-time algorithm which identifies permuted submodular binary VCSPs. Contrary to the soft local consistency algorithms, the present one computes the permutations that make the instance at hand submodular. This algorithm, however, works only on VCSPs with finite valuations and uses a strictly monotonic aggregation operator: the arithmetic sum.

6 Conclusion

In this work, we showed that permuted submodular binary VCSP, with possibly infinite valuations and a non strictly increasing aggregation operator, can be polynomially identified if we assume that the size the value domains are bounded by a constant and that the valuation functions are prime. For VCSP instances in this class, the proposed approach provides an explicit permutation for every value domain.

References

1. Chen E., Zhang Z., Wang X., Aihara K. An algorithm for fast recognition of connected row-convex constraint networks. In *JFPLC 2001, The Tenth International French Speaking Conference on Logic and Constraint Programming*, 43–58, April 2001.
2. Cohen, D. A., Cooper, M. C., Jeavons, P. G., Krokhin, A. A.: A maximal tractable class of soft constraint satisfaction. *Journal of Artificial Intelligence Research* 22 1–22, 2004.
3. Cohen, D. A., Cooper, M. C., Jeavons, P. G.: Generalizing submodularity and horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theoretical Computer Science* 401, 36–51, 2008.
4. Creignou, N., Khanna, S., Sudan, M.: Complexity Classification of Boolean Constraint Satisfaction Problems. In *SIAM Monographs on Discrete Mathematics and Applications*, 7 SIAM, 2001.
5. Cooper, M. C.: Arc consistency for soft constraints. *Artificial Intelligence* 154 199–227, 2004.

6. Cooper, M. C. Minimization of locally-defined submodular functions by Optimal Soft Arc Consistency. *Constraints* 13 (4), 2008.
7. Cooper M. C., de Givry S., Sanchez M., Schiex T., Zytnicki M. Virtual arc consistency for weighted csp. In *Proceeding of AAAI-08* Chicago, IL, 2008a.
8. Cooper, M. C., de Givry S., Sanchez M., Schiex T., Zytnicki M., Werner T. Soft arc consistency revisited. *Artificial Intelligence* 174, 449–478, 2010.
9. Green, M., Cohen, D. Domain permutation reduction for constraint satisfaction problems. *Artificial Intelligence*, 172, 1094–1118, 2008.
10. Jeavons, P.G., Cooper, M.C. Tractable constraints on Ordered domains. *Artificial Intelligence*, 79 (2) : 327339, 1995.
11. Khatib, L., Morris, P., Morris, R., Rossi, F. Temporal constraint reasoning with preferences. In *Proceedings of the* 17^{th} *International Joint Conference on Artificial Intelligence IJCAI-01* Seattle, USA (2001) 322–327, 2001.
12. Larrosa, J., Schiex, T. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159 (2004) 126.
13. Naanaa, W., Helaoui, M., Ayeb, B. A decomposition method for valued CSPs In *Soft'10 - 10th Workshop on Preferences and Soft Constraints*, 2010.
14. Narayanan, H. Submodular Functions and Electrical Networks, North-Holland, Amsterdam, 1997.
15. Nemhauser, G., Wolsey, L. Integer and Combinatorial Optimization, *John Wiley and Sons*, 1988.
16. Orlin J.B. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118 (2) 237–251, 2009.
17. Schlesinger, D. Exact Solution of Permuted Submodular MinSum Problems, In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, number 4679–2007 in LNCS, 28–38, 2007.
18. Schiex, T., Fargier, H., Verfaillie, G.: Valued constraint satisfaction problems: hard and easy problems. In *Proceedings of the* 14^{th} *IJCAI* Montréal, Canada, 631–637, 1995.
19. Schrijver, A. A combinatorial algorithm minimizing submodular functions in strongly polynomial time, *Journal of Combinatorial Theory*, Series B 80 (2) 346–355, 2000.
20. Topkis, D. Supermodularity and Complementarity, *Princeton University Press*, 1998.
21. Zhang, Y., Yap, R. H. C., Jaffar,.J : Functional Elimination and 0/1/All Constraints. In *AAAI* IAAI, 175–180, 1999.
22. Zivny, S., Jeavons, P.G. Classes of submodular constraints expressible by graph cuts. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming, CP'08*, in *Lecture Notes in Computer Science*, 5202, Springer, 2008.

Sorted-Pareto Dominance: an extension to the Pareto Dominance relation and its application in Soft Constraints

Conor O'Mahony and Nic Wilson

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

Abstract. The Pareto dominance relation, widely used in a number of decision making areas, can suffer from a lack of discriminatory power, which may result in a large number of undominated or optimal solutions. By strengthening this relation, we can narrow down this optimal set further, which is desirable e.g., for presenting a smaller number of more interesting solutions to a decision maker. This paper looks at a particular strengthening of the Pareto dominance relation, called Sorted-Pareto dominance, giving a semantics for the relation, and explores some possible usages within a Soft Constraints setting for providing a set of optimal solutions to a Soft Constraints problem. We also look at some further possible usages of the relation, with a view to future work and providing some empirical results.

1 Introduction

The notion of Pareto optimality originated in social welfare and economic theory, and the Pareto dominance relation is widely used in that field and many other related decision fields, such as collective decision and voting theory (where it is also known as 'unanimity' [13]), decision making under uncertainty (where it is also known as 'dominance' [3]), and multi-criteria decision making and optimisation [5, 10]. In a general decision-making context, a decision Pareto dominates another if it is strictly preferred in at least one aspect of the decision (where an aspect can be: a criterion in multi-criteria decision-making; a state of the world or a scenario in decision making under uncertainty; a voter in collective decision making, etc.) and at least as good as the other in all other aspects [13].

A problem with the Pareto dominance relation is its lack of discriminatory power, as many comparisons between pairs of decisions do not result in dominance, and this leads to a large number of undominated (also called 'Pareto optimal') solutions. Therefore, it is desirable to look at relations that extend the Pareto dominance relation, where the extending relation has more power when comparing decisions, which leads to a smaller set of undominated solutions that are still Pareto optimal.

In this paper we look at an extension to the Pareto dominance relation called Sorted-Pareto dominance. Section 2 describes a general decision making framework and the Pareto dominance relation. Section 3 defines the Sorted-Pareto relation and gives a semantics for the relation, while Section 4 introduces a general Soft Constraints problem framework, along with a Sorted-Pareto instance, and looks at a depth first branch and bound algorithm for solving such problems. Section 5 looks at various extensions to Sorted-Pareto, and Sections 6 and 7 look at related and possible future work in this area.

2 Preliminaries

First we describe a simple setup for decision making. Let $\mathcal{A}$ represent a set of decisions, and let $\mathcal{S} = \{1, \dots, m\}$, represent a finite set of decision aspects. Let α_i represent an evaluation, on a totally ordered scale T, of decision α in aspect i, where the scale T is ordered by $\geq$. This induces a relation $\succcurlyeq_i$ on $\mathcal{A}$ defined as, for all $\alpha, \beta \in \mathcal{A}$, $\alpha \succcurlyeq_i \beta$, if and only if, $\alpha_i \geq \beta_i$, i.e., α is as least as good as β in aspect i. Let $\succ_i$ represent the strict (or asymmetric) part of $\succcurlyeq_i$, and let $\equiv_i$ represent the indifference (or symmetric) part of $\succcurlyeq_i$.

An extension $\succcurlyeq_*$ to a relation $\succcurlyeq$, is a new relation such that, if α is preferred to β according to the original relation, then it is still preferred according to the extension, i.e., $\alpha \succcurlyeq \beta \Rightarrow \alpha \succcurlyeq_* \beta$.

2.1 Pareto Dominance

We now describe the various preference relations associated with Pareto dominance [13], along with some of the classifications of Pareto optimality.

Weak-Pareto Dominance: For all $\alpha, \beta \in \mathcal{A}$, α weak-Pareto dominates β, written as $\alpha \succcurlyeq_P \beta$, if and only if, for all $i \in \{1, \dots, m\}$, $\alpha \succcurlyeq_i \beta$. This is also called "Pareto preference or indifference".

Pareto Dominance: For all $\alpha, \beta \in \mathcal{A}$, α Pareto dominates β, written as $\alpha \succ_P \beta$, if and only if, for all $i \in \{1, \dots, m\}$, $\alpha \succcurlyeq_i \beta$, and there exists $j \in \mathcal{S}$ such that $\alpha \succ_j \beta$. Equivalently, this can be defined in terms of weak-Pareto dominance, as, $\alpha \succ_P \beta$, if and only if, $\alpha \succcurlyeq_P \beta$ and $\beta \not\succcurlyeq_P \alpha$.

Strict-Pareto Dominance: For all $\alpha, \beta \in \mathcal{A}$, α strict-Pareto dominates β, written as $\alpha \succ_{PS} \beta$, if and only if, for all $i \in \{1, \dots, m\}$, $\alpha \succ_i \beta$.

Pareto Indifference/Equivalence: For all $\alpha, \beta \in \mathcal{A}$, α is Pareto-indifferent to β, written as $\alpha \equiv_P \beta$, if and only if, for all $i \in \{1, \dots, m\}$, $\alpha \equiv_i \beta$. Equivalently, this can be defined in terms of weak-Pareto dominance, as, $\alpha \equiv_P \beta$, if and only if, $\alpha \succcurlyeq_P \beta$ and $\beta \succcurlyeq_P \alpha$.

A decision $\alpha \in \mathcal{A}$ is Pareto non-dominated (or Pareto optimal/efficient), if and only if, it is not Pareto dominated by any other decision, i.e., there is no $\beta \in \mathcal{A}$ such that $\beta \succ_P \alpha$. The set of these decisions, which we will denote as $PND(\mathcal{A})$, is often called the Pareto frontier. A decision is weakly Pareto non-dominated (or weak-Pareto optimal/efficient), if and only if, it is not strictly-Pareto dominated by any other decision, i.e., there is no $\beta \in \mathcal{A}$ such that $\beta \succ_{PS}$

α. We will denote the set of these decisions as $WPND(\mathcal{A})$. The set $PND(\mathcal{A})$ is a subset of $WPND(\mathcal{A})$ since $\alpha \succ_{PS} \beta \Rightarrow \alpha \succ_P \beta$.

3 Sorted-Pareto Dominance

We now define the Sorted-Pareto dominance relation, which is an extension to the Pareto dominance relation. This is based on an ordering defined in [11]. Firstly, in addition to the above notation, we have the following. A *sorted-permutation* $\alpha^{\uparrow}$ of a decision α is a reordering of the evaluations of the aspects of the decision in ascending order, i.e., $\alpha^{\uparrow} = (\alpha_{(1)}, \ldots, \alpha_{(m)})$, such that, $\alpha_{(1)} \leq \ldots \leq \alpha_{(m)}$.

Definition 1 (Weak Sorted-Pareto Dominance:). *For all $\alpha, \beta \in \mathcal{A}$, α Weak Sorted-Pareto dominates β, written as $\alpha \succcurlyeq_S \beta$, if and only if, $\alpha^{\uparrow} \succcurlyeq_P \beta^{\uparrow}$, i.e., $\alpha_{(i)} \geq \beta_{(i)}$, for all $i \in \{1, \ldots, m\}$.*

Definition 2 (Sorted-Pareto Dominance:). *For all $\alpha, \beta \in \mathcal{A}$, α Sorted-Pareto dominates β, written as $\alpha \succ_S \beta$, if and only if, $\alpha^{\uparrow} \succ_P \beta^{\uparrow}$, i.e., $\alpha_{(i)} \geq \beta_{(i)}$, for all $i \in \{1, \ldots, m\}$ and there exists $j \in \{1, \ldots, m\}$ such that $\alpha_{(j)} > \beta_{(j)}$. Equivalently, this can be defined in terms of Weak Sorted-Pareto dominance, as, $\alpha \succ_S \beta$, if and only if, $\alpha \succcurlyeq_S \beta$ and $\beta \not\succcurlyeq_S \alpha$.*

Definition 3 (Strict Sorted-Pareto Dominance:). *For all $\alpha, \beta \in \mathcal{A}$, α Strict Sorted-Pareto dominates β, written as $\alpha \succ_{SS} \beta$, if and only if, $\alpha^{\uparrow} \succ_{PS} \beta^{\uparrow}$, i.e., $\alpha_{(i)} > \beta_{(i)}$, for all $i \in \{1, \ldots, m\}$.*

Definition 4 (Sorted-Pareto Equivalence:). *For all $\alpha, \beta \in \mathcal{A}$, α is Sorted-Pareto equivalent to β, written as $\alpha \equiv_S \beta$, if and only if, $\alpha_{(i)} \equiv \beta_{(i)}$, for all $i \in \{1, \ldots, m\}$. Equivalently, this can be defined in terms of Weak Sorted-Pareto dominance, as, $\alpha \equiv_S \beta$, if and only if, $\alpha \succcurlyeq_S \beta$ and $\beta \succcurlyeq_S \alpha$.*

A decision $\alpha \in \mathcal{A}$ is Sorted-Pareto non-dominated, if and only if, it is not Sorted-Pareto dominated by any other decision, i.e., there is no $\beta \in \mathcal{A}$ such that $\beta \succ_S \alpha$. We will denote this set as $SPND(\mathcal{A})$. We will see that the set of Sorted-Pareto non-dominated decisions $SPND(\mathcal{A})$ is a subset of the Pareto non-dominated decision $PND(\mathcal{A})$, therefore we have a smaller set of undominated solutions that are still Pareto optimal. Let us look at an example.

Example 1. Consider, for example, a trivial group decision making problem, where the evaluation occurs of two decisions, $\mathcal{A} = \{\alpha, \beta\}$, by three different decision makers, $\mathcal{S} = \{1, 2, 3\}$, on the ordered scale $T = \{superb, good, ok, poor, bad\}$ (where of course $superb \geq good \geq ok \geq poor \geq bad$). Suppose decision α is evaluated as $(ok, poor, superb)$ and decision β is evaluated as $(poor, good, superb)$. According to Pareto dominance, we can see that $\alpha \not\succcurlyeq_P \beta$, and $\beta \not\succcurlyeq_P \alpha$, i.e., neither decision dominates each other, and therefore both α and β are in the set of undominated decisions, i.e., $PND(\mathcal{A}) = \{\alpha, \beta\}$, (so either could be chosen as the actual decision). Now, if we use Weak Sorted-Pareto dominance instead of Pareto dominance, then the sorted permutations of the decisions are

$\alpha^\uparrow = (\mathit{poor}, \mathit{ok}, \mathit{superb})$ and $\beta^\uparrow = (\mathit{poor}, \mathit{good}, \mathit{superb})$, and we can see that $\alpha^\uparrow \not\succcurlyeq_P \beta^\uparrow$ and $\beta^\uparrow \succcurlyeq_P \alpha^\uparrow$, i.e., $\alpha \not\succcurlyeq_S \beta$ and $\beta \succcurlyeq_S \alpha$, i.e., β dominates α using the Weak Sorted-Pareto relation. Therefore, in this instance, β is the only undominated decision, i.e., $SPND(\mathcal{A}) = \{\beta\}$, and therefore it could make sense to choose decision β over α.

The Sorted-Pareto class of relations can be useful in decision making situations where the evaluations of the aspects are using the same scale, for example in the situation where the aspects correspond to different voters or experts, or even different criteria that may use the same scale. However, even in situations where evaluations of aspects may not be on the same scale, then there are methods for modifying or normalising the different scales, e.g., using [11], so that the scales are commensurate. We assume that each of the aspects are of equal importance, however we look further at dealing with situations where this is not the case in Section 5.

3.1 Some Properties of Weak Sorted-Pareto Dominance

We now look at some properties of the Weak Sorted-Pareto dominance relation. $\succcurlyeq_S$ is reflexive and transitive, (i.e., it is a preorder), and it extends the Pareto dominance relation.

Proposition 1. *$\succcurlyeq_S$ is a preorder: it is transitive, i.e., $\forall \alpha, \beta, \gamma \in \mathcal{A}$, if $\alpha \succcurlyeq_S \beta$ and $\beta \succcurlyeq_S \gamma$, then $\alpha \succcurlyeq_S \gamma$; and reflexive, i.e., $\forall \alpha \in \mathcal{A}$, $\alpha \succcurlyeq_S \alpha$.*

Proof. Suppose $\alpha \succcurlyeq_S \beta$ and $\beta \succcurlyeq_S \gamma$. Therefore, by definition of $\succcurlyeq_S$, $\alpha^\uparrow \succcurlyeq_P \beta^\uparrow$ and $\beta^\uparrow \succcurlyeq_P \gamma^\uparrow$. By transitivity of $\succcurlyeq_P$, $\alpha^\uparrow \succcurlyeq_P \gamma^\uparrow$, which by definition of $\succcurlyeq_S$, implies that $\alpha \succcurlyeq_S \gamma$. Therefore $\succcurlyeq_S$ is transitive.

For any $\alpha \in \mathcal{A}$, consider its sorted permutation, $\alpha^\uparrow = (\alpha_{(1)}, \ldots, \alpha_{(m)})$. Since T is a totally ordered scale, then for any scale value $v \in T$, $v \geq v$, which means for any $\alpha_{(i)} \in \alpha^\uparrow$, $\alpha_{(i)} \geq \alpha_{(i)}$, i.e., for all $i \in \{1, \ldots, m\}$, $\alpha_{(i)} \geq \alpha_{(i)}$. Therefore, by definition of $\succcurlyeq_P$, $\alpha^\uparrow \succcurlyeq_P \alpha^\uparrow$, which by definition of $\succcurlyeq_S$ implies that $\alpha \succcurlyeq_S \alpha$. Therefore $\succcurlyeq_S$ is reflexive. ∎

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

Proof. Suppose $\alpha \succcurlyeq_P \beta$. Therefore, by definition of $\succcurlyeq_P$, $\alpha_i \geq \beta_i$, for all $i = 1, \ldots, m$. Consider any $j \in \{1, \ldots, m\}$. $\beta_{(j)}$ is the j-th smallest element of β, so there are at least $(m - j)$ components β_i of β with $\beta_i \geq \beta_{(j)}$. If $\beta_i \geq \beta_{(j)}$, then $\alpha_i \geq \beta_{(j)}$, since $\alpha_i \geq \beta_i$ by definition of $\succcurlyeq_P$. Therefore, there are at least $(m - j)$ components α_i of α with $\alpha_i \geq \beta_{(j)}$, in particular $\alpha_{(j)} \geq \beta_{(j)}$. Therefore, $\alpha^\uparrow \succcurlyeq_P \beta^\uparrow$, which by definition of $\succcurlyeq_S$, implies that $\alpha \succcurlyeq_S \beta$. ∎

3.2 A Semantics for Sorted-Pareto Dominance

We now look at a semantics for the Sorted-Pareto dominance class of relations. Firstly, as already given in the preliminaries, this semantics assumes some totally ordered qualitative or ordinal scale T, which is ordered by $\geq$. Also we consider each decision $\alpha \in \mathcal{A}$ to be characterised by its vector of m evaluations, (over the m decision aspects), where each evaluation is on the scale T, and so for simplicity we interchangeably refer to some $\alpha \in \mathcal{A}$ as either a decision or as a vector of evaluations corresponding to a decision.

One way of comparing decisions using these evaluations is to map the qualitative scale values onto quantitative values, possibly representing some sort of utility, e.g., one can define a weights function f on the scale values, i.e., $f : T \to \mathbb{R}^+$, where the function is monotonic with respect to the ordering of the scale, i.e., $\forall v, v' \in T$, $v \geq v' \Leftrightarrow f(v) \geq f(v')$. Therefore, for our set of decisions $\mathcal{A}$, and using a particular weights function f, the decisions can be compared and ordered using a order relation $\geq_f$, which is given by the sum of the m weights, i.e., for some $\alpha, \beta \in \mathcal{A}$, $\alpha \geq_f \beta$, if and only if: $\sum_{i=1}^{m} f(\alpha_i) \geq \sum_{i=1}^{m} f(\beta_i)$. This order relation $\geq_f$ is a total preorder on decisions, but for different mappings (i.e., different f), the resulting orders may be different.

Table 1. Example 2

	$f_1(v)$	$f_2(v)$
$v = good$	6	5
$v = ok$	3	4
$v = bad$	2	1

$\mu = (\mu_1, \mu_2)$	$\sum_{i=1}^{m} f_1(\mu_i)$	$\sum_{i=1}^{m} f_2(\mu_i)$
$\alpha = (good, good)$	12	10
$\beta = (ok, good)$	9	9
$\gamma = (ok, ok)$	6	8
$\delta = (bad, good)$	8	6
$\epsilon = (bad, ok)$	5	5
$\sigma = (bad, bad)$	4	2

Example 2. Consider some ordinal scale $T = \{good, ok, bad\}$, where of course $good \geq ok \geq bad$. Table 1 defines the functions f_1 and f_2, which are weights functions that map an evaluation on the scale T to an associated utility in $\mathbb{R}^+$, and also shows the resulting sum of weights for a particular set of decisions $\mathcal{A} = \{\alpha, \beta, \gamma, \delta, \epsilon, \sigma\}$. For function f_1, the resulting order $\geq_{f_1}$ is $[\alpha \geq_{f_1} \beta \geq_{f_1} \gamma \geq_{f_1} \delta \geq_{f_1} \epsilon \geq_{f_1} \sigma]$, and for function f_2, the resulting order $\geq_{f_2}$ is $[\alpha \geq_{f_2} \beta \geq_{f_2} \delta \geq_{f_2} \gamma \geq_{f_2} \epsilon \geq_{f_2} \sigma]$. We can see that both f_1 and f_2 are monotonic with respect to the scale T but the resulting orders $\geq_{f_1}$ and $\geq_{f_2}$ are different.

When it is possible to provide a weights function (like f_1 or f_2) to map the scale values to some quantitative measure, then it is easy to compare decisions by using the sum of these weights. However, sometimes it is not possible to create this quantitative mapping, e.g., when this information is not available, so in these such cases, we can consider a different order that does not rely on this quantitative information. If we consider all possible weights functions $f : T \to \mathbb{R}^+$ (such that f is monotonic with respect to the ordering of T), then we can define an order relation $\geq_F$ on $\mathcal{A}$ as:

$$\forall \alpha, \beta \in \mathcal{A},\ \alpha \geq_F \beta \Leftrightarrow \alpha \geq_f \beta,\ \forall f : T \to \mathbb{R}^+$$

This relation is the intersection of all possible order relations $\geq_f$, for all monotonic functions f defined on T, i.e.,

$$\geq_F = \bigcap \geq_f, \forall f : T \to \mathbb{R}^+$$

This order $\geq_F$ is a preorder on $\mathcal{A}$, i.e., it is reflexive $(\forall \alpha, \beta \in \mathcal{A}, \alpha \geq_F \beta)$, transitive $(\forall \alpha, \beta, \gamma \in \mathcal{A}, \alpha \geq_F \beta \wedge \beta \geq_F \gamma \Rightarrow \alpha \geq_F \gamma)$, and by Theorem 1 below, is equal to the Weak Sorted-Pareto order $\succcurlyeq_S$.

Theorem 1. *$\geq_F$ is equal to the Weak Sorted-Pareto order $\succcurlyeq_S$.*

Proof. First we show that $\forall \alpha, \beta \in \mathcal{A}, \alpha \succcurlyeq_S \beta \Rightarrow \alpha \geq_F \beta$. Assume $\alpha \succcurlyeq_S \beta$. Therefore, by definition of $\succcurlyeq_S$, $\alpha^\uparrow \succcurlyeq_P \beta^\uparrow$, which means that $\alpha_{(i)} \geq \beta_{(i)}$, $\forall i \in \{1, \ldots, m\}$. Therefore, for any f, $\sum_{i=1}^m f(\alpha_{(i)}) \geq_f \sum_{i=1}^m f(\beta_{(i)})$. Since $\alpha^\uparrow$ and $\beta^\uparrow$ are permutations of α and β respectively, then $\sum_{i=1}^m f(\alpha_{(i)}) \geq_f \sum_{i=1}^m f(\beta_{(i)}) \Leftrightarrow \sum_{i=1}^m f(\alpha_i) \geq \sum_{i=1}^m f(\beta_i)$, which implies, for any monotonic function f, $\alpha \geq_f \beta$. Since this is true for any f, then $\alpha \geq_F \beta$.

Now we show that $\forall \alpha, \beta \in \mathcal{A},\ \alpha \not\succcurlyeq_S \beta \Rightarrow \alpha \not\geq_F \beta$. Assume $\alpha \not\succcurlyeq_S \beta$. Therefore, by definition of $\succcurlyeq_S$, $\alpha^\uparrow \not\geq_P \beta^\uparrow$. Therefore, $\exists i \in \{1, \ldots, m\}$ such that $\alpha_{(i)} \not\geq \beta_{(i)}$. We can construct a monotonic function f such that $\alpha \not\geq_f \beta$, and therefore $\alpha \not\geq_F \beta$. For example, for $f : T \to \{0, 1\}$, assign f such that $f(\alpha_{(i)}) = 0$ and $f(\beta_{(i)}) = 1$, e.g., $f(v) = 0$ if $v \leq \alpha_{(i)}$, and $f(v) = 1$ otherwise. Therefore, $\sum_{j=1}^m f(\alpha_{(j)}) \leq m - i$ and $\sum_{j=1}^m f(\beta_{(j)}) \geq m - i + 1$, so $\sum_{j=1}^m f(\alpha_{(j)}) < \sum_{j=1}^m f(\beta_{(j)})$, therefore $\exists f$ such that $\alpha \not\geq_f \beta$, and therefore $\alpha \not\geq_F \beta$. ∎

This gives a semantics to the Sorted-Pareto order, as a relation that can be used in decision making situations where there may only be ordinal or qualitative information available, and it provides an ordering that is consistent with any possible weights function selected to map an ordinal scale to a numerical one. It can be viewed as a more cautious representation than a weighted constraints one, and it can be applied in many of the application areas of weighted constraints. The weighted constraints formalism assumes that the costs are on an additive scale, where the cost of A and B is the sum of the costs of A and B; however, in many situations this can be questionable. For example, suppose one is using a weighted constraints solver to find a most probable explanation (MPE problem). Elicitation of probabilities can be problematic and unreliable. If instead of taking the elicited values at face value, one considers them as just representing the ordering between the probabilities, then Theorem 1 shows that the Sorted-Pareto ordering represents the ordered pairs that all compatible probability assignments agree with.

4 Sorted-Pareto and Soft Constraints

In this section, we briefly introduce Soft Constraints, and look at how the Sorted-Pareto dominance relation can be used within a Soft Constraints setting. We look at a depth first branch and bound algorithm for solving a Soft Constraints problem, using the Sorted-Pareto dominance relation to prune parts of the search tree, to provide a set of Sorted-Pareto optimal solutions.

4.1 A General Framework for Soft Constraints Problems

Soft constraints [12] can be used to model many real-world problems where there is a need to specify preferences on particular aspects of the problem solutions. A soft constraint associates a preference degree to a particular aspect of a solution, e.g., an assignment to a set of problem variables, and the preference levels of a complete solution provided by the soft constraints can be combined to give the overall preference degree of that solution, with a view to ordering the solution set and obtaining optimal solutions.

A Preference Degree Structure (PDS), (similar to as defined in [4]), is a tuple $\mathcal{P} = \langle I, \otimes, \succeq \rangle$, where I is an ordered set of preference degrees, $\succeq$ is a partial order on I, and $\otimes$ is a commutative and associative operator, monotonic with respect to $\succeq$, which is used to combine the preference degrees. We define a general Soft Constraints problem [16] to be a tuple $\mathcal{F} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{P} \rangle$, where $\mathcal{X}$ is a set of variables, with finite domain $\mathcal{D}$, and $\mathcal{C}$ is a set of soft constraints, where a soft constraint $c \in \mathcal{C}$ is a mapping from an assignment of a subset V of the variables in $\mathcal{X}$ to a preference degree, i.e., $c : \mathcal{D}^{|V|} \to I$. The subset V of the variables in $\mathcal{X}$ to which the constraint is applied is called the scope of the constraint, we also denote this by *scope*(c), i.e., *scope*$(c) = V$.

For some assignment u, each soft constraint is applied to the subtuple of the assignment which corresponds to the scope of the constraint, i.e., $c(u^{\downarrow V})$, (where $u^{\downarrow V}$ is the projection of u to the variables in V), and we abbreviate this to $c(u)$. The overall preference degree $\rho(u)$ of that assignment is the combination of all the preference levels of all constraints, i.e. $\rho(u) = \bigotimes_{c \in \mathcal{C}} c(u)$. An optimal assignment u is one such that there is no other assignment u' such that $\rho(u')$ is preferred to $\rho(u)$, i.e., there is no other assignment u' such that $\rho(u') \succeq \rho(u)$.

Sorted-Pareto Instance. First, we recall the general decision making setup in Section 2, where we have a scale T, and $\geq$ is a total order on T. For a Sorted-Pareto instance of a general Soft Constraints problem, we use a PDS $\mathcal{P} = \langle I, \otimes, \succeq \rangle$, where I is the set of multisets of T, (denoted by M^T), the combination operator $\otimes$ is *multiset sum*, i.e., $\otimes = \uplus$, so the combination of the preference degrees for an assignment is just the multiset containing all the preference degrees, e.g., for some $c, c' \in \mathcal{C}$, and for some assignment u, if $c(u) = \{v\}$ and $c'(u) = \{v'\}$, (where $v, v' \in T$), then $c(u) \otimes c'(u) = \{v, v'\}$. For $\succeq$, we use the Weak Sorted-Pareto ordering $\succcurlyeq_S$ as defined in Section 3, extended in the obvious way to multisets in T of the same cardinality.

Using the Sorted-Pareto instance of the PDS, $\mathcal{P} = \langle M^T, \uplus, \succcurlyeq_S \rangle$, the overall preference degree of an assignment u is $\rho(u) = \bigotimes_{c \in \mathcal{C}} c(u) = \biguplus_{c \in \mathcal{C}} c(u) = \{c(u\} : c \in \mathcal{C}\}$. Therefore, an optimal assignment u is one that is not Sorted-Pareto dominated by any other assignment, i.e., there is no other assignment u' such that $\rho(u') \succ_S \rho(u)$.

4.2 DFBB Search

One way of solving a Soft Constraints problem is to use a Depth First Branch and Bound search (DFBB), we now briefly describe a general setup for such a search and we look at one such DFBB algorithm for solving Soft Constraints problems. We assume a standard Constraint Satisfaction Problem, where there is a set of hard constraints, as well as some strict partial order $\succ$ on the set of complete assignments (or solutions), and the domains of the variables are updated when a new assignment is made (e.g., using arc consistency). For ease of presentation, we use a fixed variable ordering $(X_1, X_2, \ldots, X_m)$ of the variables in $\mathcal{X}$. Let level i in the search tree correspond to the variable X_i in the variable ordering. A path from root to a particular node at level i corresponds to some partial assignment to the variables $X_1, \ldots, X_{i-1}$, and a full path from root to a leaf node corresponds to a complete assignment. Therefore, for some partial assignment u at a node at level i, each successor node represents an extension to assignment u with variable X_i being assigned a value from $\mathcal{D}(X_i)$. For some partial assignment u, let $S(u)$ represent the set of possible full extensions (solutions) to partial assignment u.

We consider two conditions that may hold for some partial assignment u and how they can be used to either prune the search tree at a particular node or reduce the amount of dominance checks that occur at subsequent nodes during the search.

Partial Assignment Dominated (PAD) condition: Given partial assignment u at node N, and set S of undominated solutions, if all extensions of u are strictly dominated by some $x \in S$, i.e., there is no $x \in S$, such that for all $y \in S(u)$, $\rho(x) \succ \rho(y)$, then we can prune the tree at node N.

Partial Assignment Non-Dominated (PAND) condition: Given partial assignment u at node N, and set S of undominated solutions, for some complete assignment $x \in S$, if all extensions of u are not strictly dominated by x, i.e, for all $y \in S(u)$, $\rho(x) \not\succ \rho(y)$, then we do not need to include x in subsequent dominance checks for extensions of u below node N.

4.3 DFBB Algorithm

We now present a DFBB algorithm for computing a set of undominated solutions for our Soft Constraints problem (see Algorithm 1). First we describe the auxiliary functions used within the main DFBB function.

The function HAS-NEXT-VAR$(u, \mathcal{X})$ returns $true$ if there is another variable to assign, $false$ otherwise. The function NEXT-VAR$(u, \mathcal{X})$ is a variable selection function, that selects the next uninstantiated variable from $\mathcal{X}$ for which

Algorithm 1 Recursive algorithm for DFBB with $\langle \text{RUS}, \text{NEW} \rangle$

```
 1: function DFBB(Tuple: $u$, $\langle$Set : RUS, Set : NEW$\rangle$) $\rightarrow$ $\langle$Set : RUS$'$, Set : NEW$'\rangle$
 2:     if HAS-NEXT-VAR($u, \mathcal{X}$) = $false$ then                ▷ $u$ is complete assignment...
 3:         if $\neg$ IS-DOMINATED($u$, RUS) then               ▷ ... and not strictly dominated
 4:             NEW := $\{u\}$                                          ▷ update NEW set
 5:         end if
 6:         return $\langle$RUS, NEW$\rangle$
 7:     else                                                    ▷ $u$ is partial assignment
 8:         for all $x \in$ RUS do
 9:             if PAD($u, x$) then                                ▷ PAD condition holds
10:                 return $\langle$RUS, NEW$\rangle$        ▷ backtrack with $\langle$RUS, NEW$\rangle$ unchanged
11:             end if
12:         end for
13:         NEW$'$ := NEW
14:         $X$ := NEXT-VAR($u, \mathcal{X}$)                       ▷ next variable in ordering
15:         for all $a \in D(X)$ do
16:             $u' := u \cup (X, a)$                        ▷ assign domain value to variable
17:             RUS$'$ := RUS$'$ $\cup$ NEW$'$
18:             OTHERS := $\emptyset$
19:             NEW$'$ := $\emptyset$
20:             for all $x \in$ RUS$'$ do                                        ▷ Reduce RUS$'$
21:                 if $\langle$PAND($u', x$)$\rangle$ then                ▷ PAND condition holds...
22:                     RUS$'$ := RUS$'$ $\setminus \{x\}$            ▷ ... remove $x$ from future checks
23:                     OTHERS := OTHERS $\cup \{x\}$
24:                 end if
25:             end for
26:             $\langle$RUS$'$, NEW$'\rangle$ := DFBB($u'$, $\langle$RUS$'$, NEW$'\rangle$)
27:             OTHERS := REMOVE-DOMINATED(OTHERS, NEW$'$)
28:             RUS$'$ := RUS$'$ $\cup$ OTHERS
29:         end for
30:     end if
31:     return $\langle$RUS$'$, NEW$'\rangle$
32: end function
```

to assign a domain value. The function IS-DOMINATED(u, S) returns $true$ if some complete assignment u is strictly dominated by any $x \in S$, $false$ otherwise. The function REMOVE-DOMINATED(S, S') returns a new set, containing the elements of S that are not dominated by any element of S'.

The main function DFBB takes as input a partial assignment u and a pair of sets $\langle \text{RUS}, \text{NEW} \rangle$, where RUS is the set of relevant undominated solutions inherited from the parent node, and NEW is the set of undominated solutions found so far in leaf nodes. At any return point during the algorithm, when the solution pair $\langle \text{RUS}', \text{NEW}' \rangle$ is being returned, the following property holds: $Max(\text{RUS}' \cup \text{NEW}') = (\text{RUS}' \cup \text{NEW}')$.

The algorithm will backtrack if it encounters a partial assignment that will not extend to a non-dominated solution, (PAD condition holds), i.e., if there is some solution in RUS that dominates u. Also, if it can be shown that some complete assignment $x \in$ RUS fails to dominate any complete assignment extending partial assignment u, (PAND condition holds), then there is no need to consider x in nodes below the current node in the search tree, so the algorithm removes x from RUS (and adds x to a set variable OTHERS to allow such x to be restored on backtracking).

4.4 Sorted-Pareto Problem Instances

We consider a couple of different general types of situations where we can use DFBB search for Sorted-Pareto soft constraints instances:

Situation 1. For each variable, there is a unary soft constraint representing the preference degree of the assignment to that variable, i.e., for a Soft Constraint Problem $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{P} \rangle$, where $\mathcal{P}$ is as previously defined in Section 4.1, and $\mathcal{C}$ is a set of soft constraints, such that for all $c \in \mathcal{C}$, $\mathit{scope}(c) = \{X\}$ for some $X \in \mathcal{X}$, i.e., $|\mathit{scope}(c)| = 1$.

Situation 2. There are non-unary soft constraints on two or more variables, representing a preference degree of an assignment to those variables, i.e., for a Soft Constraint Problem $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{P} \rangle$, where $\mathcal{P}$ is as previously defined in Section 4.1, and $\mathcal{C}$ is a set of soft constraints, such that for all $c \in \mathcal{C}$, $\mathit{scope}(c) = V \subseteq \mathcal{X}$, i.e., $|\mathit{scope}(c)| \geq 1$.

4.5 Sufficient Checks for PAD/PAND Conditions.

We now look at some sufficient checks for both the PAD and PAND conditions, for use in the two described situations.

PAD Check for Situation 1: For some partial assignment u at node N in the search tree, let $\rho^*(u)$ be an upper bound multiset of preference degrees for complete extensions of u, defined as:

$$\rho^*(u) = \rho(u) \cup \{\max c(\mathcal{D}(\mathit{scope}(c))) \,:\, c \in \mathcal{C}, \mathit{scope}(c) \in \mathcal{X} \setminus \mathcal{U}\}$$

If, for some $x \in$ RUS, $\rho(x) \succcurlyeq_S \rho^*(u)$, then the PAD condition holds, and the search tree can be pruned at node N.

PAD Check for Situation 2: For some partial assignment u at node N in the search tree, let $\rho^*(u)$ be an upper bound multiset of preference degrees for complete extensions of u, defined as:

$$\rho^*(u) = \rho(u) \cup \{\max c(\mathcal{D}(\mathit{scope}(c))) \,:\, c \in \mathcal{C}, \mathit{scope}(c) \nsubseteq \mathcal{U}\}$$

If, for some $x \in$ RUS, $\rho(x) \succcurlyeq_S \rho^*(u)$, then the PAD condition holds, and the search tree can be pruned at node N.

PAND Check for Situation 1: For some partial assignment u at node N in the search tree, let $\rho_*(u)$ be a lower bound multiset of preference degrees for complete extensions of u, defined as:

$$\rho_*(u) = \rho(u) \cup \{\min c(\mathcal{D}(scope(c))) \,:\, c \in \mathcal{C}, scope(c) \in \mathcal{X} \setminus \mathcal{U}\}$$

If, for some $x \in$ RUS, $\rho(x) \not\succcurlyeq_S \rho_*(u)$, then the PAND condition holds, and we do not need to include x in subsequence dominance checks for extensions of u below node N.

PAND Check for Situation 2: For some partial assignment u at node N in the search tree, let $\rho_*(u)$ be a lower bound multiset of preference degrees for complete extensions of u, defined as:

$$\rho_*(u) = \rho(u) \cup \{\min c(\mathcal{D}(scope(c))) \,:\, c \in \mathcal{C}, scope(c) \nsubseteq \mathcal{U}\}$$

If, for some $x \in$ RUS, $\rho(x) \not\succcurlyeq_S \rho_*(u)$, then the PAND condition holds, and we do not need to include x in subsequence dominance checks for extensions of u below node N.

5 A Sorted-Pareto extension

In this section, we look at one possible extension to the Sorted-Pareto relation, in particular, we look at a lexicographic extension, where some decision aspects are given higher priority than others, for example, the aspects might be states that are more likely to occur, criteria that are more important, or voters with more weight. This is similar to the approach taken by [8] for handling preferences between criteria in multi-criteria problems, and to Lexicographic Constraint Satisifaction Problems in [6].

5.1 Lexicographic Sorted-Pareto

In addition to the general decision making setup in Section 2, (where we have a set of decisions $\mathcal{A}$, a set of decisions aspects $\mathcal{S}$, and an ordered scale T, ordered by $\geq$), we consider the following.

Let $\mathcal{L} = \{L_1, \ldots, L_k\}$ represent an ordered partition of the set of aspects $\mathcal{S}$, ordered by $>_z$ in terms of importance, i.e., for some $L, L' \in \mathcal{L}$, $L >_z L'$, if and only if L has a higher importance than L'. Each $i \in \mathcal{S}$ appears in only one $L \in \mathcal{L}$, and for $i, j \in \mathcal{S}$, $i \equiv_z j$ if they appear in the same $L \in \mathcal{L}$, i.e., i and j are equally important. The Lex-Sorted permutation, for some $\alpha \in \mathcal{A}$, is defined as $\alpha^{\uparrow}_{\mathcal{L}} = (\alpha^{\uparrow}[L_1], \alpha^{\uparrow}[L_2] \ldots, \alpha^{\uparrow}[L_k])$, where $\alpha^{\uparrow}[L_i]$ is the sorted permutation of $\alpha[L_i]$, i.e., the sorted permutation of the evaluations of α for the aspects of L_i. We can now define the Lex-Sorted-Pareto dominance relation $\succcurlyeq^{\mathcal{L}}_S$.

Definition 5 (Lexicographic Sorted-Pareto). *For a partition $\mathcal{L}$ of $\mathcal{S}$, $\mathcal{L} = \{L_1, L_2, \ldots, L_k\}$, where $\mathcal{L}$ is ordered by $>_z$, and $\forall \alpha, \beta \in \mathcal{A}$, α Lex-Sorted-Pareto dominates β, written as $\alpha \succcurlyeq^{\mathcal{L}}_S \beta$, if and only if, $\exists j$, such that, $\forall i < j$, $\alpha^{\uparrow}[L_i] \equiv_P \beta^{\uparrow}[L_i] \wedge \alpha^{\uparrow}[L_j] \succ_P \beta^{\uparrow}[L_j]$*

Consider the following example.

Example 3. Consider the evaluations of two decisions $\alpha, \beta \in \mathcal{A}$, over five decision aspects $\mathcal{S} = \{1, 2, 3, 4, 5\}$, on an ordered scale $T = \{good, ok, bad\}$. Let $\mathcal{L} = (\{5\}, \{2, 4\}, \{1, 3\})$ be an ordered partition on $\mathcal{S}$, where $L_1 = \{5\}$, $L_2 = \{2, 4\}$ and $L_3 = \{1, 3\}$, and $L_1 >_z L_2 >_z L_3$. Suppose decision α is evaluated as $(good, bad, ok, ok, good)$, and decision β is evaluated as $(bad, ok, bad, good, good)$. The Lex-Sorted permutation for α is $\alpha^{\uparrow}_{\mathcal{L}} = ((good), (bad, ok), (good, ok))$, and the Lex-Sorted permutation for β is $\beta^{\uparrow}_{\mathcal{L}} = ((good), (ok, good), (bad, bad))$. We can see that β Lex-Sorted-Pareto dominates α, since, for L_1, $\alpha^{\uparrow}[L_1] \equiv_P \beta^{\uparrow}[L_1]$, i.e., the decisions are equivalent with respect to L_1, and for L_2, $\beta^{\uparrow}[L_2] \succ_P \alpha^{\uparrow}[L_2]$, i.e., $(ok, good) \succ_P (bad, ok)$, therefore $\beta \succcurlyeq^{\mathcal{L}}_{S} \alpha$.

When $|\mathcal{L}| = |\mathcal{S}|$, then this ordering is equal to a lexicographic ordering, and when $|\mathcal{L}| = 1$, then this ordering is equal to the Sorted-Pareto order. When $|\mathcal{L}| > 1$ and $|\mathcal{L}| < |\mathcal{S}|$, this represents some situation where there is some information available on the relative importance of some decision aspects.

6 Related Work

The notion of Strict Sorted-Pareto appears in [11], where it is used in a method for comparing and ranking alternatives in a multicriteria decision making problem. Another version appears in [9], where it is called "Ordered Pareto", and is used for handling preferences and comparing alternatives using possibilistic logic. [15] looks at Pareto dominance between non-decreasingly ordered income distributions, which is called "Pareto-rank dominance". This amounts to Sorted-Pareto dominance, however it is the nature of problem itself that these income distributions are already ordered before the Pareto dominance relation is applied.

Some work that looks at different Branch and Bound algorithms for various soft constraints and multi-criteria problems includes [16], which looks at algorithms for Depth First Branch and Bound search for Soft Constraints, and [7] looks at an algorithm for partially ordered Constraint Optimisation Problems (PCOP). [14] looks at algorithms for multi-criteria optimisation in Soft CSPS for approximating Pareto Optimal solution sets, and the work in [1] looks at methods for the computation of leximin optimal solutions in Constraint Networks, which include various Branch and Bound algorithms.

7 Future Work and Discussion

In this section we briefly discuss some future work. As mentioned in Section 4.3, the current version of the algorithm is one such approach, so the future work will involve other versions and improvements of this algorithm, e.g., to improve efficiency. Currently, the implementation of Sorted-Pareto DFBB search is at an early stage of development, and we present no empirical results at this point, but the future work will involve further implementation, possibly on top of the

Choco CSP solver [2], along with some empirical analysis. Also the future work will involve implementing extensions of Sorted-Pareto as discussed in Section 5, along with developing further extensions and implementations thereof.

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

References

1. Bouveret, S., Lemaître, M.: Computing leximin-optimal solutions in constraint networks. Artificial Intelligence vol. 173(2), pp. 343–364 (2009)
2. Choco: an open source Java constraint programming library (2011), http://www.emn.fr/z-info/choco-solver/index.html
3. Congar, R., Maniquet, F.: A trichotomy of attitudes for decision-making under complete ignorance. Mathematical Social Sciences vol. 59(1), pp. 15–25 (2010)
4. Fargier, H., Rollon, E., Wilson, N.: Enabling local computation for partially ordered preferences. Constraints vol. 15(4), pp. 516–539 (2010)
5. Figueira, J., Greco, S., Ehrgott, M.: Multiple Criteria Decision Analysis: State of the Art Surveys. Springer Verlag, Boston, Dordrecht, London (2005)
6. Freuder, E.C., Heffernan, R., Wallace, R.J., Wilson, N.: Lexicographically-ordered constraint satisfaction problems. Constraints vol. 15(1), pp. 1–28 (2010)
7. Gavanelli, M.: Partially ordered constraint optimization problems. In: CP 2001: 7th International Conference on Principles and Practice of Constraint Programming, November 26–December 1, 2001, Paphos, Cyprus, Proceedings. Springer Verlag (2001)
8. Junker, U.: Preference-based search and multi-criteria optimization. Annals OR vol. 130(1-4), pp. 75–115 (2004)
9. Kaci, S., Prade, H.: Mastering the processing of preferences by using symbolic priorities in possibilistic logic. In: ECAI 2008, 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings. pp. 376–380 (2008)
10. Keeney, R., Raiffa, H.: Decisions with multiple objectives: Preferences and value tradeoffs. J. Wiley, New York (1976)
11. Larichev, O.I., Moshkovich, H.M.: ZAPROS-LM – a method and system for ordering multiattribute alternatives. European Journal of Operational Research vol. 82(3), pp. 503–521 (1995)
12. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, NY, USA (2006)
13. Sen, A.K.: Collective choice and social welfare. North-Holland Publishing Co., Amsterdam (1970)
14. Torrens, M., Faltings, B.: Using soft CSPs for approximating Pareto-optimal solution sets. In: AAAI 2002, Workshop Proceedings, Preferences in AI and CP: Symbolic Approaches. AAAI Press (2002)
15. Traub, S., Seidl, C., Schmidt, U.: An experimental study on individual choice, social welfare, and social preferences. European Economic Review vol. 53(4), pp. 385–400 (2009)
16. Wilson, N., Fargier, H.: Branch-and-bound for soft constraints based on partially ordered degrees of preference. In: ECAI 08, Proceedings, Workshop on Inference methods based on graphical structures of knowledge (WIGSK08) (2008)

Two-sided Function Filtering

M. Pujol-Gonzalez, J. Cerquides, P. Meseguer and J. A. Rodriguez-Aguilar

IIIA - CSIC, Universitat Autònoma de Barcelona
08193 Bellaterra, Catalonia, Spain.
{mpujol|cerquide|pedro|jar}@iiia.csic.es

Abstract. Function filtering enhances dynamic programming methods working on a tree decomposition of the constraint graph. It is based on bounds for tuples: if the lower bound of tuple t is equal to or higher than a suitable upper bound, t can be discarded, decrementing the size of the message to travel in the tree decomposition. We present a new form of lower bound that tightens the lower bound of the original function filtering, so this new version –called *two-sided function filtering*– is more powerful. We provide experimental evidence of its benefits.

1 Introduction

In constraint satisfaction, inference is widely used but in a very limited form. A simple example is arc consistency: by the inspection of constraints and domains, it is able to deduce that some values will never be in a solution so they can be removed. Arc consistency is incomplete inference since it cannot always produce a solution. Inference can also be complete. Some algorithms are adaptive consistency [5], cluster tree methods [7] and bucket elimination [3]. Their temporal and spatial complexities are exponential in some parameters of the constraint graph (see [4] for details). When compared with search methods (exponential complexity in time but linear complexity in space), they look unattractive, especially when search is enhanced with the powerful machinery of local consistency coupled with global constraints [14].

In the soft constraints realm, satisfaction is replaced by optimization. This causes that problems with soft constraints become more difficult to solve than their hard counterparts. The same solving ideas are recreated here. Search methods, based on a branch-and-bound schema, are combined with soft local consistencies to filter domains [10]. Complete inference methods are easily adapted to compute the optimum, at the cost of dragging large arity constraints. Their high spatial complexity is the main drawback to be used in practice. Nevertheless, this issue is not always unavoidable: when there are ways to control the spatial complexity, complete inference can provide excellent performance [9].

While search algorithms consider assignments of individual variables, dynamic programming methods handle whole cost functions which are combined and exchanged among nodes of a suitable decomposition of the problem instance. Function filtering [13] reduces the size of cost functions by filtering out those tuples that are found unfeasible to be extended into an optimal solution. Provided a lower bound on the cost of the best extension of the tuple, and an upper bound on the cost of the optimal solution, a tuple is filtered out when its lower bound reaches the upper bound. Authors of [13] proposed a form of computing a lower bound of the cost of the best extension of a tuple t, that we call *one-sided lower bound*. It works as follows. When computing a cost function at node i for node j, the cost of the best extension of t is computed combining the cost of t in one particular cost function at node i with the cost of functions on t variables coming from node j. We extend this form of lower bound, producing the *two-sided lower bound* where all functions at node i (and not only one) are taken into account when computing the cost of the best extension of t. Since the new lower bound tightens the old one, it is direct to see that this new approach is more powerful than the original one. Combining this new lower bound with the function filtering idea, we obtain the *two-sided function filtering* approach, that is the paper contribution.

The structure of the paper is as follows. Section 2 contains some concepts used throughout the paper. Section 3 details the original one-sided filtering method. Section 4 focuses on improving lower bounds, presenting the two-sided function filtering approach. This approach is empirically evaluated in Section 5. Finally, Section 6 draws some conclusions from this work.

2 Preliminaries

In this paper we consider soft constraints that are represented as cost functions using the weighted model [12]. A weighted CSP (WCSP) is defined as $\langle X, D, C, S(k)\rangle$ where X and D are variables and domains as in CSP. C is a finite set of constraints as cost functions; $f_T \in C$ (T is the scope of f_T) assigns costs to value tuples $t \in \prod_{x_i \in T} D_i$, such that,

$$f(t) = \begin{cases} 0 & \text{if } t \text{ is allowed} \\ 1 \dots k-1 & \text{if } t \text{ is partially allowed} \\ k & \text{if } t \text{ is totally forbidden} \end{cases}$$

$S(k) = \langle [0, 1, ..., k], \oplus, \geq \rangle$ is a valuation structure such that $a \oplus b = min\{k, a + b\}$, $\top = k$, $\bot = 0$ [8]. We assume that the reader is familiar with assignments or value tuples t_S with scope S, complete tuples ($S = X$), projections over $S' \subset S$, $t_S[S']$, and concatenation of two tuples $t_S \cdot t'_T$, defined only if common variables coincide in their corresponding values. We assume that $f_T(t_S)$ (with $T \subset S$) always means $f_T(t_S[T])$. A complete tuple t_X is *consistent* if $\bigoplus_{f_T \in C} f_T(t_X) < k$, else t_X is *inconsistent*. A *solution* is a complete consistent assignment with minimum cost. Finding a solution is NP-hard. With $k = 1$ WCSP reduces to CSP.

We define the *combination* of two functions f_T and g_S as a new function $f_T \bowtie g_S$ with scope $T \cup S$ and $\forall t \in \prod_{x_i \in T} D_i$, $\forall t' \in \prod_{x_j \in S} D_j$ such that $t \cdot t'$ is defined, $f_T \bowtie g_S(t \cdot t') = f_T(t) \oplus g_S(t')$. Let $F = \{f_{T_1}, \dots, f_{T_m}\}$ be a set of functions, *the combination of* F, $\bowtie F$, is the function resulting from the joint combination of every function in F, namely,

$$\bowtie F = f_{T_1} \bowtie \dots \bowtie f_{T_m}$$

Let $V \subseteq X$ be a subset of the variables of the problem and t_V a tuple that assigns values to each of the variables in V. An extension of t_V to X is a tuple that keeps the assignments of t_V and assigns new values to the variables in $X \setminus V$. If the cost of each possible extension of t_V is larger than or equal to LB, we say that LB is a lower bound of the cost of the best extension of tuple t_V. Likewise, a function f_T is a *lower bound* of function f_S, noted $f_T \leq f_S$, iff $T \subseteq S$, and $\forall t_S \; f_T(t_S[T]) \leq f_S(t_S)$. A function g_V is a lower bound of a set of functions F if it is a lower bound of its combination $\bowtie F$.

The *min-marginal* $f_S[T]$ of a cost function f_S over $T \subset S$ is a new cost function on T variables which assigns to each tuple t_T the minimum cost among all the extensions of t_T to S. Formally,

$$\forall t_T \qquad f_S[T](t_T) = \min_{t_S \text{ extension of } t_T} f_S(t_S[T]).$$

The tightest lower bound is provided by the min-marginal. Similarly, the *min-marginal of F over V* is the min-marginal of the combination of all the functions in F, that is $(\bowtie F)[V]$.

Given a set of functions F, the time to compute the min-marginal of F over V is bounded by $\mathcal{O}(d^{|T|})$, where $T = \bigcup_{i=1}^{m} T_i$, and d is the size of the common domain of the variables in T. In some scenarios, this can be overdemanding. For that reason we introduce a less costly way of computing a lower bound of a set of functions. Specifically, we define $\overset{\text{V}}{\bowtie} F$, *the combination of F under V* as the result of combining the min-marginals of each of its functions over V. That is,

$$\overset{\text{V}}{\bowtie} F = f_{T_1}[T_1 \cap V] \bowtie \ldots \bowtie f_{T_m}[T_m \cap V].$$

$\overset{\text{V}}{\bowtie} F$ is a lower bound of F and can be assessed in $\mathcal{O}(d^k)$ time, where parameter $k = \max\left(\max_{i=1}^{m} |T_i|, |V|\right)$ [1], which can be way smaller than $\mathcal{O}(d^{|T|})$. However, this lower bound can be inferior to $(\bowtie F)[V]$.

To apply dynamic programming methods, we need a suitable decomposition of the problem instance. A *tree decomposition* (also called *joint tree* or *junction tree*) of a WCSP $\langle X, D, C, S(k)\rangle$ is a triplet $\langle T, \chi, \psi\rangle$, where $T = \langle N, E\rangle$ is a tree (N is a set of nodes and E is a set of edges), χ and ψ are labeling functions which associate with each node $i \in N$ two sets, $\chi(i) \subseteq X$ and $\psi(i) \subseteq C$ such that: (1) for each function $f_S \in C$, there is exactly one node $i \in N$ such that $f_S \in \psi(i)$ and $S \subseteq \chi(i)$; (2) for each variable $x \in X$, the set $\{i \in N | x \in \chi(i)\}$ induces a connected subtree of T. The *tree-width* of a tree decomposition is $tw = max_{i \in N}|\chi(i)|$. If $(i, j) \in E$, the *separator* is $sep(i, j) = \chi(i) \cap \chi(j)$. In the following, nodes of the tree decomposition are called *clusters*. The neighbors of cluster i, $neigh(i)$, is the set of clusters linked to i in the tree decomposition. Figure 1 shows cluster i and j linked by an edge in a tree decomposition. Observe that removing the edge connecting i and j splits the tree decomposition into two different connected components, which we call subproblems (see Figure 1). Formally, we say that the i-subproblem involves every cost function in the component containing i after the edge is removed. Subproblems i and j are coupled by a set of variables they share and must agree upon, namely their separator $sep(i, j)$.

[1] Computing each $f_{T_i}[T_i \cap V]$ takes $\mathcal{O}(d^{|T_i|})$ time, while computing the whole expression takes $\mathcal{O}(d^{|V|})$ time.

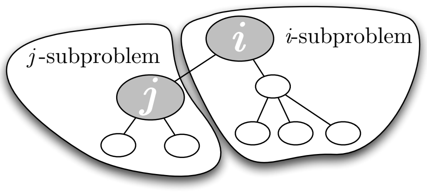


Fig. 1: Subproblems in a tree decomposition.

Cluster-Tree Elimination (CTE) is an algorithm that optimally solves WCSP by sending messages along tree decomposition edges [11, 6, 7]. CTE can be seen as the fully serial version of the Generalized Distributive Law (GDL) algorithm for the all-nodes (or all-clusters) problem [1]. Edge $(i, j) \in E$ has associated two CTE messages: $\hat{g}^{(i \to j)}$, from i to j, and $\hat{g}^{(j \to i)}$, from j to i. $\hat{g}^{(i \to j)}$ is the min-marginal computed combining all functions in $B(i, j)$ (set of functions formed by $\psi(i)$ with all incoming CTE messages except $\hat{g}^{(j \to i)}$) over $sep(i, j)$. CTE complexity is time $O(d^{tw})$ and space $O(d^s)$, where d is the largest domain size and s is the maximum separator size.

Mini-Cluster-Tree Elimination (MCTE(r) [7]) approximates CTE (and therefore it computes solutions that are not necessarily optimal). If the number of variables in a cluster is high, it may be impossible to compute $\hat{g}^{(i \to j)}$ due to memory limitations. MCTE(r) computes a lower bound by limiting by r the arity of the functions sent in the messages. A MCTE(r) message, $G^{i \to j}$, is a set of functions that approximate the corresponding CTE message $\hat{g}^{(i \to j)}$. It is computed as $\hat{g}^{(i \to j)}$ but instead of combining all functions of set $B(i, j)$, it computes a partition $P = \{B_1, B_2, \ldots, B_p\}$ of $B(i, j)$ such that the combination of the functions in every B_k does not exceed arity r. The MCTE(r) algorithm is time and space complexity $O(d^r)$.

3 One-sided Function Filtering

Function filtering [13] is a technique that reduces the size of cost functions by filtering out those tuples that are found unfeasible to be extended into an optimal solution, because their cost reach or surpass a suitable upper bound. To perform function filtering, each cluster i intending to send a cost function f_U to cluster j needs: (1) a lower bound $lb_U(t_U)$ on the cost of the best extension of each tuple t_U; and (2) an upper bound UB on the value of the optimal solution. Provided that, we say that the cluster i filters f_U with lb_U and UB when it filters out those tuples t_U such that $lb_U(t_U) \geq UB$ (*i.e.* the ones that cannot be extended into an optimal solution), and sends the remaining ones.

While CTE exchanges exact cost functions among clusters, MCTE(r) exchanges approximate cost functions (one or several of arity up to r), first bottom-up and then top-down the tree decomposition. After this, each cluster i has: (1) a set of functions $\psi(i)$, containing its stake at the problem; and (2) for each neighbor j a set of functions $G^{j \to i}$ defined on the variables $sep(i, j)$. $G^{j \to i}$ stands for a summary of the j-subproblem, namely a lower bound on the cost of each tuple in that subproblem. Observe that cluster

i can assess the cost of an assignment by adding its own costs and the costs of its neighbors' subproblems. Likewise, the cluster can assess a lower bound for the costs of a tuple in the complete problem by combining its own cost functions with those received from its neighbors. Formally,

$$lb_{\chi(i)}(t_{\chi(i)}) = (\bowtie F)(t_{\chi(i)}) \tag{1}$$

where $F = \psi(i) \cup \bigcup_{j \in neigh(i)} G^{j \to i}$ (that is, the combination of all received functions in cluster i with the functions initially present in cluster i). However, the lower bound assessed in Equation (1) requires $\mathcal{O}(d^{|\chi(i)|})$ time, where $\chi(i)$ are the variables of cluster i and d the common domain size. This can be very costly to compute, both in terms of time and memory (in fact, this is the temporal complexity of the exact CTE algorithm).

As an alternative, [13] proposes the following. Let us assume that cluster i wants to send a set of functions to cluster j. Let f_U be one of these functions, $U \subseteq sep(i,j) \subsetneq \chi(i)$. A lower bound of the cost of the best extension of tuple t_U can be computed by adding a lower bound on the cost of the best extension of tuple t_U in the j-subproblem to the cost that f_U assigns to t_U. Formally,

$$lb_U(t_U) = (g_U^{j \to i} \bowtie f_U)(t_U), \tag{2}$$

where $g_U^{j \to i} = \overset{U}{\bowtie} G^{j \to i}$. Henceforth we shall refer to this lower bound as *one-sided lower bound*. In this case, the complexity of combining under U is $\mathcal{O}(d^k)$ time, where now parameter k is the maximum between $|U|$ and the arity of any function in $G^{j \to i}$, that could be cheaper than using Equation (1).

When MCTE(r) runs with increasing r, the cost of the best solution found so far is an upper bound UB. At each iteration, cluster i willing to send a set of cost functions to cluster j can filter each of them separately. For each function, cluster i (a) uses Equation (2) to assess a lower bound from the last message received from j; and (b) filters the function with this lower bound and UB. The resulting algorithm is known as IMCTEf [13].

4 Two-sided Function Filtering

Next, we aim at tightening the one-sided lower bound described above. Consider that cluster i has already received $G^{j \to i}$ from cluster j. After that, it intends to send a set of functions $G^{i \to j}$ (set that contains the function f_U mentioned in Equation (2)), summarizing the cost information in the i-subproblem, to cluster j. Since no cost function appears in both the i-subproblem and the j-subproblem, we can assess a lower bound for the complete problem by adding a lower bound of each of them. Notice that the one-sided lower bound in Equation (2) already assesses the summary of the costs of the j-subproblem from $G^{j \to i}$. Likewise, we can assess the summary of the costs of the i-subproblem from $G^{i \to j}$. Therefore, we can employ the cost summaries of both subproblems to obtain a tighter bound.

Formally, when sending cost function $f_U \in G^{i \to j}$, we compute the lower bound of tuple t_U as:

$$lb_U(t_U) = (g_U^{j \to i} \bowtie g_U^{i \to j})(t_U) \tag{3}$$

where

Given

$x\,y$	$g_{xy}^{j\to i}$
$a\,a$	3
$a\,b$	4
$b\,a$	3
$b\,b$	3

,

$x\,y$	f_{xy}
$a\,a$	5
$a\,b$	2
$b\,a$	8
$b\,b$	6

,

$x\,z$	f_{xz}
$a\,a$	4
$a\,b$	3
$b\,a$	5
$b\,b$	2

, and UB=10

				One-sided	**Two-sided**
$x\,y$	f_{xy}	$\bowtie f_{xz}[x]$	$= g_{xy}^{i\to j}$	$f_{xy} \bowtie g_{xy}^{j\to i}$	$g_{xy}^{i\to j} \bowtie g_{xy}^{j\to i}$
$a\,a$	5	3	8	5 + 3	8 + 3 ✗
$a\,b$	2	3	5	2 + 4	5 + 4
$b\,a$	8	2	10	8 + 3 ✗	10 + 3 ✗
$b\,b$	6	2	8	6 + 3	8 + 3 ✗

Fig. 2: One-sided vs. two-sided filtering. Ticked tuples (✗) are the ones being filtered out.

- $g_U^{i\to j} = \overset{U}{\bowtie} G^{i\to j}$ is a lower bound on the contribution of the i-subproblem.
- $g_U^{j\to i} = \overset{U}{\bowtie} G^{j\to i}$ is a lower bound on the contribution of the j-subproblem.

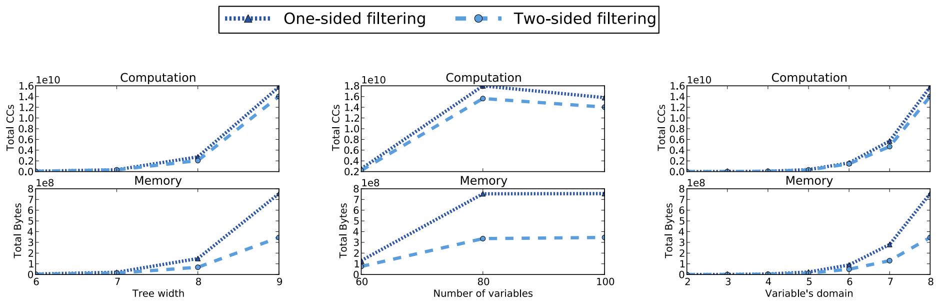
Observe that there is no double counting of costs because no cost function appears in both the i-subproblem and the j-subproblem. Henceforth, we will refer to the lower bound in Equation (3) as *two-sided lower bound*. The name stems from the symmetrical use of both subproblems. Hereafter, two-sided filtering refers to filtering employing the two-sided lower bound.

Comparing both lower bounds, we observe that one-sided lower bound computes a different lower bound for each function f_U that cluster i wants to send to cluster j (Equation (2)), while two-sided lower bound computes the same lower bound for all functions to be sent from cluster i to cluster j, namely the lower bound given by Equation (3).

As example, consider that cluster i has received a set of functions $G^{j\to i}$, which combined under $\{x, y\}$ produces the function $g_{xy}^{j\to i}$ shown in Figure 2. Furthermore, cluster i knows that the cost of the optimal solution is smaller than or equal to 10 ($UB = 10$). Now, it wants to send functions $G^{i\to j} = \{f_{xy}, f_{xz}\}$ (in Figure 2) to cluster j. Consider that it starts by sending function f_{xy}. Cluster i can calculate the one-sided lower bound using Equation (2), filtering out tuple $(x{=}b, y{=}a)$ as shown in Figure 2. Alternatively, the cluster can compute the two-sided lower bound using Equation (3), by assessing the lower bound on the contribution of its own subproblem, namely $g_{xy}^{i\to j} = \overset{xy}{\bowtie} G^{i\to j} = f_{xy} \bowtie f_{xz}[x]$. Figure 2 shows that two-sided filtering performs better, keeping only the tuple $(x{=}a, y{=}b)$ as feasible.

5 Empirical evaluation

In this section we empirically compare the performance of IMCTEf when using one-sided filtering and two-sided filtering. For each experiment, we track the amount of memory used by the algorithm (the maximum amount of memory required by the algorithm in its whole execution, from $r = 2$ until the problem is solved) along with the total amount of computation (as the number of constraint checks performed, which are directly related with the CPU time used). Moreover, we conducted signed rank tests [15]



(a) Increasing treewidth, constant domain 8 and 100 variables.

(b) Increasing number of variables, constant treewidth 8.

(c) Increasing domain size, constant treewidth 9 and 100 variables.

Fig. 3: Experimental results of one-sided filtering against two-sided filtering. Median of constraint checks and bytes exchanged are reported.

on all results to ensure that differences between methods are statistically significant ($\alpha = 0.01$).

One may be curious about the number of tuples that are filtered by our method. We report the amount of memory used and not the number of filtered tuples because not all filtered tuples cause the same savings in memory (savings depends on tuple's arity). Since we provide results aggregating increasing arity limits, we believe that the number of saved bytes is a more precise measure than the number of filtered tuples. It is worth noting that filtering a tuple in an early iteration implies that none of its extensions have to be considered in the future ones; therefore, filtering a tuple at a given iteration has a multiplicative effect in future iterations.

It is well-known that the treewidth is the most important indicator of problem hardness for CTE-based algorithms (the temporal complexity is exponential in such parameter). Hence, we segmented our experiments according to this parameter, and ensured that all algorithms use the very same tree decomposition when solving the same problem instance. We generated hard instances characterizing each scenario by three parameters: number of variables, variables' domain size, and treewidth. For each scenario, we generated 100 problems by: (1) randomly drawing problem structures following an Erdös-Rényi $G(n, p)$ model [2]; (2) selecting those structures having the treewidth requested for the scenario; and (3) randomly drawing costs from a $\mathcal{N}(0, 1)$ normal distribution (function costs are made positive by adding its minimum value to each function).

First, we ran an experiment to evaluate the savings as the treewidth increases. We generated scenarios with 100 variables of domain 8, and treewidths ranging from 6 to 9. Figure 3a shows that two-sided filtering reduces, with respect to one-sided filtering, the amount of memory required by a median of more than 25% for the easier problems (treewidth 6). It achieves even better results for the harder problems (more than 50% for the set with treewidth 9).

Next, we arranged an experiment to assess the impact of increasing numbers of variables. Hence, we generated 5 sets of 100 problems, with an increasing number of variables for each set. Since we only wanted to measure the impact of the varying number of variables, we generated many random problem structures and selected only those that yielded a fixed treewidth of 8. Figure 3b shows the median results achieved by both approaches on each set of problems. Notice that using two-sided instead of one-sided filtering reduces the total amount of bytes by more than 40% in most cases, while achieving a 54% reduction in the 100-variables problems set. Nevertheless, there is an interesting trend change in both computation and memory requirements past the 80-variables set. The cause of this change is that all problems have the very same treewidth of 8. Therefore, as the number of variables increases, the resulting problems are sparser.

Finally, we designed an experiment to measure the trend of both filtering styles as the variables' domain sizes increase. Thus, we generated scenarios with 100 variables, treewidth 9 and domain sizes ranging from 2 to 8. Once again, two-sided filtering achieves significant memory savings for all the experiment's problems. Further, as the domain increases, so do the savings with respect to one-sided filtering: starting with a narrow 7% reduction for domains of size 2, and reaching more than 50% reduction for the toughest scenario (domain size 8).

6 Conclusions

We have presented the two-sided function filtering approach in the context of WCSP, when soft constraints are represented as cost functions. This approach comes from the combination of the two-sided lower bound computation with the function filtering idea. Given a tree decomposition, two-sided lower bound considers the aggregation of costs coming from the two disjoint subproblems, such that its union constitutes the whole problem instance. Specifically, two-sided lower bound for tuple t at cluster i considers the costs of t in the subproblem i coming not only from function f_U to be send to cluster j, but also from other functions of cluster i. This cost is combined with the cost coming from subproblem j, to compute the cost of t's best extension. This two-sided lower bound directly extends one-sided lower bound, and it is straightforward to see that it is more powerful. Experimentally, we have shown the benefits of this approach with respect to one-sided function filtering in both time and memory, using a number of experiments.

References

1. S. M. Aji and R. J. Eliece. The generalized distributive law. *IEEE Trans. Inf. Theory*, 46(2):325–343, 2000.
2. B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.
3. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artifical Intelligence*, 113:41–85, 1999.
4. R. Dechter. *Constraint Processing*. Elsevier Science, 2003.
5. R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.
6. R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38, 1989.
7. K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artifical Intelligence*, 166:165–193, 2005.
8. J. Larrosa. Node and arc consistency for weighted csp. In *Proc. AAAI*, 2002.
9. J. Larrosa, E. Morancho, and D. Niso. On the practical applicability of bucket elimination: Still-life as a case study. *Journal of Artificial Intelligence Research*, to appear, 2005.
10. J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159, 2004.
11. S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal statistical Society, Series B*, 50(2):157–224, 1988.
12. P. Meseguer, F. Rossi, and T. Schiex. *Soft Constraints*, chapter 9 of Handbook of Constraint Programming, pages 281–328. Elsevier, 2006.
13. M. Sánchez, J. Larrosa, and P. Meseguer. Improving tree decomposition methods with function filtering. In *IJCAI*, pages 1537–1538, 2005.
14. W. J. van Hoeve and I. Katriel. *Global Constraints*, chapter 6 of Handbook of Constraint Programming, pages 169–208. Elsevier, 2006.
15. F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

Soft Nonlinearity Constraints and their Lower-Arity Decomposition

Venkatesh Ramamoorthy[1], Marius C.Silaghi[1], Toshihiro Matsui[2], Katsutoshi Hirayama[3], and Makoto Yokoo[4]

[1] Florida Institute of Technology, Melbourne, FL 32901, United States of America
vramamoo@my.fit.edu, msilaghi@cs.fit.edu
[2] Nagoya Institute of Technology, Nagoya, Aichi, 466-8555, Japan
matsui.t@nitech.ac.jp
[3] Kobe University, Kobe, 657-8501, Japan hirayama@maritime.kobe-u.ac.jp
[4] Kyushu University, Hakozaki 6-10-1, Higashi-ku, Fukuoka, 812-8581, Japan
yokoo@is.kyushu-u.ac.jp

Abstract. In this paper we express nonlinearity constraints in terms of soft global n-ary constraints. We describe a method to decompose nonlinearity constraints to obtain redundant hard constraints as projections of global lower-arity constraints. The nonlinearity constraints apply to the inputs and outputs of discrete functions $f : \mathbb{Z}_{2^n} \rightarrow \mathbb{Z}_{2^m}$ mapping n-bit inputs to m-bit outputs, $n > m$. No output bit of the function f should be too close to a linear function of (a subset of) its input bits. That is, if we select any output bit position and any subset of the six input bit positions, the fraction of inputs for which this output bit equals the exclusive-OR of these input bits should not be close to 0 or 1, but rather should be near $\frac{1}{2}$. We analyze this constraint and find that the obtained redundant constraints increase the efficiency of arc consistency maintenance solver by several orders of magnitude.

Keywords: CSP Model, Soft Constraint, S-boxes, DES, 3DES, Nonlinearity, Cryptanalysis, Global Constraint, Projection, Decomposition, n-ary Constraint

1 Introduction

The nonlinearity constraint is proposed in [9] to model nonlinearity requirements that are essential for the security of cryptographic algorithms (ciphers). If substitution operations in ciphers could be represented as linear relations, their parameters could be easily obtained by solving a system of such equations connecting pairs of inputs and outputs. Even when the functions are not perfectly linear, any success in approximating them with linear functions can increase the chances of success in guessing their parameters. As such, [4] and [7] define one of the main nonlinearity requirements as: *No output bit of the function f should be too close to a linear function of the input bits. That is, if we select any output bit position and any subset of the six input bit positions, the fraction of inputs for which this output bit equals the exclusive-OR of these input bits should not be close to 0 or 1, but rather should be near $\frac{1}{2}$.* In this paper, we discuss our formulation of the nonlinearity constraint as a soft global n-ary constraint, and prove a method to obtain a set of equivalent hard, redundant constraints by

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Fig. 1. The 3DES 6×4 S-box S_8

S-1 Each S-box has six bits of input and four bits of output.
S-2 No output bit of an S-box should be too close to a linear function of the input bits. (That is, if we select any output bit position and any subset of the six input bit positions, the fraction of inputs for which this output bit equals the exclusive-OR of these input bits should not be close to 0 or 1, but rather should be near $\frac{1}{2}$).
S-3 If we fix the leftmost and rightmost input bits of the S-box and vary the four middle bits, each possible 4-bit output is attained exactly once as the middle four input bits range over their 16 possibilities.
S-4 If two inputs to an S-box differ in exactly one bit, the corresponding outputs must differ in at least two bits.
S-5 If two inputs differ in the two middle bits exactly, the outputs must differ in at least two bits.
S-6 If two inputs differ in the first two bits and are identical in the last two bits, the two outputs must be different.
S-7 For any nonzero 6-bit difference between inputs $\Delta I_{i,j}$, no more than eight of the 32 pairs of inputs exhibiting $\Delta I_{i,j}$ may result in the same output difference $\Delta O_{i,j}$.

Table 1. The nonlinearity criteria used by IBM for designing 3DES S-boxes [4]

employing projections. The nonlinearity constraints apply on the inputs and outputs of discrete functions $f : \mathbb{Z}_{2^n} \rightarrow \mathbb{Z}_{2^m}$ mapping n-bit inputs to m-bit outputs, $n > m$. Such functions are commonly referred to as Substitution boxes (S-boxes). We analyze these constraints and find that the obtained redundant constraints increase the efficiency of arc consistency maintenance solver by orders of magnitude.

2 Background

We now introduce the nonlinearity requirement, originally defined in [4].

S-box Criteria and Nonlinearity An $n \times m$ substitution box (S-box) that scrambles (substitutes) an n-bit input data to yield an m-bit output, is a function $S : \mathbb{Z}_{2^n} \rightarrow \mathbb{Z}_{2^m}$ where $\mathbb{Z}_k$ stands for the set $\{0, ...k-1\}$. S is not necessarily invertible. Substitution-permutation networks [10], a common cipher architecture, use S-boxes in the generation of a parametrized substitution of x. These S-boxes have to be nonlinear and invertible. One of the SP-network versions, the Feistel cipher architecture [5], relaxes this constraint by removing the invertibility requirement. Namely, it proceeds through iterations of the function:

$$F : \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^m} \rightarrow \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^m}, F(x,y) = split(y||f(x,y),n,m),$$

where $a||b$ stands for the concatenation of the bits of a and b. The function f is:

$$f : \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^m} \rightarrow \mathbb{Z}_{2^n}, f(x, y) = x \oplus S(y),$$

and the function *split* is:

$$split : \mathbb{Z}_{2^{n+m}} \times \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}_{2^n} \times \mathbb{Z}_{2^m}, split(z, n, m) = (z >> n, z\&(2^m - 1))$$

which splits the $(n + m)$-bit number z in two parts of n bits and m bits respectively, $\mathbb{Z}$ being the set of (non-negative) integers. The definition uses the operator $a >> b$ which right-shifts a number a by b bits, and $\&$ for bit-wise AND. The obtained substitution function $f(x, \cdot)$ is a parametrized bijection, and it is therefore often referred to as a permutation function. This function of parameter y should necessarily be nonlinear [4]. Since it is difficult to design and verify a nonlinear S-box function that works on a large sequence of bits n, cipher designers replace them with a set of smaller S-boxes, where each handles a fraction of the sequence of bits.

One of the most commonly-used ciphers in Netscape's Secure Sockets Layer (SSL) protocol and the newer Transport Layer Security (TLS) protocol is Triple-DES (3DES), which is an example of a Feistel architecture. Other known Feistel architectures are Blowfish, Twofish, RC5, Camellia, etc. 3DES works by a triple-application of the old Data Encryption Standard (DES) developed by IBM [1]. It employs eight 6×4 S-boxes numbered $S_1, S_2, \ldots, S_8$, with S_8 shown in Fig. 1. An S-box substitution of 4 bits for a 6-bit input i is obtained by indexing into the row number formed by the first and last bits of i, and the column number formed by the middle bits of i. For example, input of 45 ($= 101101_2$) to S-Box S_8 yields 8 $= (1000_2)$, obtained by reading the entry in row 3 ($= 11_2$), column 6 ($= 0110_2$) of Fig. 1. The S-boxes are so designed to satisfy criteria numbered **S-1**, **S-2**, and so on [4], which are listed in Table 1. The S-box nonlinearity constraint **S-2** states that *the output of an S-box should be highly nonlinear*. A proposal by Matsui [7] compiles this criteria into a complex metric but which allows for a quantitative comparison of S-boxes. This is the metric that we employ here under the form of a soft global nonlinearity n-ary constraint.

3 Concepts and Problem Formulation

Notations A `0x` prefixed to the left of a number, such as `0x2ab3`, specifies it is in hexadecimal notation. $|x|$ denotes the absolute value of a number x. For a set S, $|S|$ represents its cardinality while for a set expressed using braces, its cardinality is denoted by preceding the braces with a $\#$. The symbols $\cdot$ and $\oplus$ represent the bit-wise AND and exclusive-OR (XOR) operation respectively, on two identical-sized bit patterns. A linear Boolean function $L_\omega(x)$ on an n-bit pattern $x = x_0 \ldots x_{n-1}$ selected by an n-bit pattern $\omega = \omega_0 \ldots \omega_{n-1}$ is defined [3] as:

$$L_\omega(x) = \omega_0 \cdot x_0 \oplus \ldots \oplus \omega_{n-1} \cdot x_{n-1} = \bigoplus_{i=0}^{n-1} \omega_i \cdot x_i \tag{1}$$

The *Hamming weight* of a bit pattern x, denoted by $wt(x)$, is equal to the number of **1**'s in x. The amount by which x and y differ, as mentioned in Table 1, equals $wt(x \oplus y)$.

3.1 Variables and Domains

	$i_1i_2i_3i_4$							
i_0i_5	0	1	2	3	...	13	14	15
0	x_0	x_2	x_4	x_6	$\dots$	x_{26}	x_{28}	x_{30}
1	x_1	x_3	x_5	x_7	$\dots$	x_{27}	x_{29}	x_{31}
2	x_{32}	x_{34}	x_{36}	x_{38}	$\dots$	x_{58}	x_{60}	x_{62}
3	x_{33}	x_{35}	x_{37}	x_{39}	$\dots$	x_{59}	x_{61}	x_{63}

Fig. 2. Diagrammatic relationship between the defined CSP variables and 6×4 S-box entries

We now define the elements of the (X, D, C)-based CSP model for the general case of designing a nonlinear and non-invertible $n \times m$ S-box. Our concrete examples are for 6×4 S-boxes such as those used in 3DES. Note that invertible S-boxes can be obtained when $n = m$ by simply adding an `alldiff` constraint, which makes the function one to one.

To model our nonlinearity criteria, we define the set X of 2^n variables $X = \{x_0, x_1, \dots, x_{2^n-1}\} = \{x_i | i \in \mathbb{Z}_{2^n}\}$, each representing an entry in the S-box. The domain in D of each variable is $\mathbb{Z}_{2^m} = \{0, 1, \dots, 2^m - 1\}$.

To adapt the CSP for our case study of $n \times m$ S-boxes, the i^{th} variable x_i specifies the m-bit S-box output for an n-bit input i. Using the variables in X, a 6×4 S-box such as the ones used in 3DES, is organized as shown in Fig. 2, addressed by incrementing the input. In Fig. 2, a 6-bit input $i, 0 \leq i \leq 63$ is represented by the bit pattern $i_0i_1i_2i_3i_4i_5$ for clarity. Criterion **S-1** in Table 1 is already satisfied based on our choice of variables.

3.2 Nonlinearity Metrics for Variable Assignments

Since for each input i the S-box returns the value of x_i, therefore the nonlinearity of the S-box can be stated as a nonlinearity between each index i and the value of x_i. The ability of expressing each bit of an m-bit value $e \in \mathbb{Z}_{2^m}$ in the assignment $x_i = e$, as a linear combination of the bits in the n-bit subscript $i \in \mathbb{Z}_{2^n}$ [7, 6], is now examined. Here, we use this measure as the score of a solution (to be optimized) and extend the definition to a partial assignment.

Consider an n-bit subscript $i = i_0 \dots i_{n-1}$ of a variable x_i, and a corresponding assignment to x_i of a value from $\mathbb{Z}_{2^m}$. The linear combinations to be checked for equality are obtained by selecting bits in i and the value assigned to x_i using selectors a and b respectively, $\forall a, b, 0 \leq a < 2^n$ and $0 \leq b < 2^m$. We denote, by $L_\omega(x_i)$, the application of the function L_ω of Equation 1 on the value assigned to the CSP variable x_i. For a complete assignment Φ with all variables in X assigned, let $N_X^\Phi(a, b)$, quantifying the *success of linearization* of the relation between i to x_i using coefficients a and b, be:

$$N_X^\Phi(a,b) = \#\{i | x_i \in X; L_a(i) = L_b(x_i)\} \tag{2}$$

Observe that $0 \leq N_X^\Phi(a, b) \leq 2^n$.

Given a partial-assignment Φ' resulting from a partial instantiation of variables $X' \subseteq X$, we further define the *partial success of linearization* $N_{X'}^{\Phi'}(a,b)$ as follows:

$$N_{X'}^{\Phi'}(a,b)=\#\{i|x_i \in X'; L_a(i)=L_b(x_i)\} \tag{3}$$

Besides the properties for $N_X^{\Phi}(a,b)$ [7], the following properties are also inferred directly from the definition of $N_{X'}^{\Phi'}(a,b)$.

Property 1. $\forall a$, b, X', Φ', $0 \leq N_{X'}^{\Phi'}(a,b) \leq |X'|$.

Property 2. $\forall a, b, u, X', \Phi',$ and $u \in X \setminus X'$, $N_{X' \cup \{u\}}^{\Phi'}(a,b) - N_{X'}^{\Phi'}(a,b) \in \{0,1\}$.

Proof. Property 2 states the immediate observation that the consideration of each additional input can raise the number of correctly linearized inputs by at most 1. This reasoning applied consecutively to each variable in X' is used to explain Property 1.
Q.E.D.

These two properties are used to design heuristics that improve the efficiency of search for solutions to satisfy the nonlinearity constraint **S-2**.

Nonlinearity as a Probability Measure For *each* variable x_i corresponding to input i in a complete assignment Φ, given selectors a and b defined as above, let $p(a,b)$ denote the fraction of cases when $L_a(i) = L_b(x_i)$, computed as:

$$p(a,b) = \frac{N_X^{\Phi}(a,b)}{2^n} \tag{4}$$

$p(a,b) = 1$ is the condition where the linear combination of the bits in the value assigned to x_i selected by b equals a linear combination of the bits in i selected by a, i.e., $\forall i, L_a(i) = L_b(x_i)$. If $p(a,b)$ is equal to zero, the linear combination of the output bits selected by b is always equal to the negation of the linear combination of input bits selected by a. According to the nonlinearity requirement **S-2**, $p(a,b)$ should be near $\frac{1}{2}$.

Linear Approximation Table (LAT) The Linear Approximation Table [7] for a complete assignment is a $2^n \times 2^m$ matrix. Its rows are headed by selector a, $0 \leq a < 2^n$, and columns by selector b, $0 \leq b < 2^m$ (see Table 2). Each entry specifies the quantity $N_X^{\Phi}(a,b) - \frac{|X|}{2}$, with one entry in row a and column b representing an offsetted measure of the correlation between the bits of x_i selected by b and the bits of i selected by a. As an example, for the 3DES S-box S_8, the first and last two rows of its LAT are in Table 2. The LAT of a solution is formed by an arithmetic accumulation of individual contributions due to *each* variable assignment $x_i = e, i \in \mathbb{Z}_{2^n}, e \in \mathbb{Z}_{2^m}$. A contribution arising from an assignment $x_i = e$ is equal to $L_a(i) \oplus L_b(e) \oplus 1, a \in \mathbb{Z}_{2^n}, b \in \mathbb{Z}_{2^m}$, that is, 0 or 1. The offset quantity $\frac{|X|}{2}$ is subtracted from each entry in the LAT of a solution.

b / a	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
62	0	-8	4	0	-2	2	-2	6	10	6	2	2	0	0	-4	0
63	0	-8	0	4	2	-2	-10	-2	-6	6	-2	6	-4	4	-4	0

Table 2. The Linear Approximation Table for the S-box S_8 of Fig. 1

The Score of an Assignment The most effective linear approximation of a complete assignment Φ containing $|X|$ variables is obtained if, for some a and b, $|N_X^{\Phi}(a,b) - \frac{|X|}{2}|$ is maximal. To reduce the weakest point of the assignment Φ, we use the so-called *effectiveness of linearization* [8] as the optimization score:

$$\sigma_X(\Phi) = \max_{a,b}\{|N_X^{\Phi}(a,b) - \frac{|X|}{2}| : 1 \leq a < |X|; 1 \leq b < |D|\} \tag{5}$$

A complete assignment with a smaller score is considered better. We look for $\underset{\Phi}{\operatorname{argmin}}(\sigma_X(\Phi))$. The score $\sigma_{X'}$, $X' \subseteq X$, of a partial assignment Φ' is defined as:

$$\sigma_{X'}(\Phi') = \max_{a,b}\{|N_{X'}^{\Phi'}(a,b) - \frac{|X|}{2}| : 1 \leq a < |X|; 1 \leq b < |D|\} \tag{6}$$

4 The Nonlinearity Global Constraint

The straightforward modeling of the nonlinearity requirement leads to a soft constraint that minimizes $\sigma_X(\Phi)$. When used as a hard constraint for a threshold τ, it becomes:

$$\sigma_X(\Phi) \leq \tau \tag{7}$$

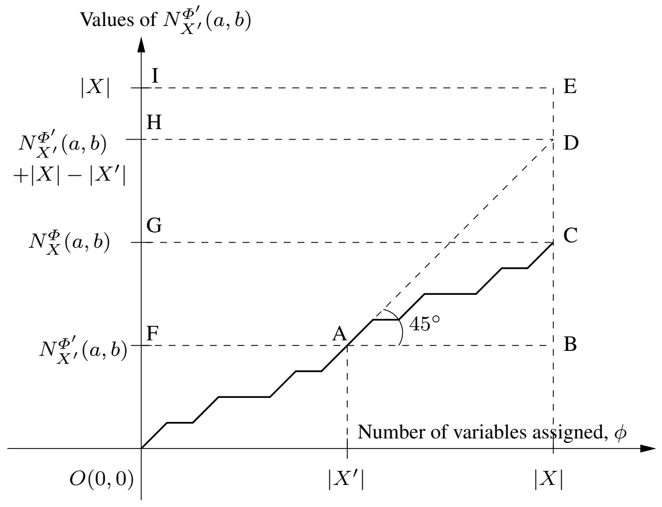
The following property of a partial assignment allows for projection of Equation 7 to lower-arity constraints.

Property 3 (Projections). A partial assignment Φ' with values for variables in X', $X' \subseteq X$, cannot be extended to a solution with score better than a threshold τ if the following inequality is not satisfied:

$$|X'| - \tau - \frac{|X|}{2} \leq \max_{a,b} N_{X'}^{\Phi'}(a,b) \leq \frac{|X|}{2} + \tau \tag{8}$$

Proof. During projection, the goal is for the score of S-box Φ' to never exceed the maximum threshold τ:

$$\max_{a,b} |N_{X'}^{\Phi'}(a,b) - \frac{|X|}{2}| \leq \tau \tag{9}$$



Values of $N^{\Phi'}_{X'}(a,b)$
I
$|X|$
E
H
$N^{\Phi'}_{X'}(a,b)$ $+|X|-|X'|$
D
G
$N^{\Phi}_{X}(a,b)$
C
F
A
45°
$N^{\Phi'}_{X'}(a,b)$
B
Number of variables assigned, ϕ
$O(0,0)$
$|X'|$
$|X|$

Fig. 3. Evaluating partially instantiated S-boxes.

Figure 3 depicts the distribution of $N^{\Phi'}_{X'}(a,b)$ (Equation 3) for a partially instantiated S-box Φ'. The horizontal axis is the number of variables instantiated, ϕ. After $|X'|$ variables are instantiated at point A along the solid line, the dashed line at a 45-degree angle with the horizontal represents the pathological case where the count $N^{\Phi'}_{X'}(a,b)$ increases by one for every subsequent extension of Φ' up to point D. The solid zig-zag lines connecting points A and C represents the corresponding, actual distribution of $N^{\Phi'}_{X'}(a,b)$ for the complete S-box Φ to attain the count equal to $N^{\Phi}_{X}(a,b)$ at point C. From this construction, we have $OF = N^{\Phi'}_{X'}(a,b)$, $OG = N^{\Phi}_{X}(a,b)$, $FH = BD = AB = |X| - |X'|$, and $OH = OF + FH = N^{\Phi'}_{X'}(a,b) + |X| - |X'|$.

By construction, $(|X| - |X'|)$ remaining variables are to be instantiated in order to extend Φ' to Φ. To guarantee extensibility: $OG \leq OH$, i.e.,

$$N^{\Phi}_{X}(a,b) \quad \leq \quad N^{\Phi'}_{X'}(a,b) + |X| - |X'|$$

This is true for all selectors a and b, and in particular, holds for the maximum value of $N^{\Phi}_{X}(a,b)$ (resp. $N^{\Phi'}_{X'}(a,b)$) over all a, b:

$$\max_{a,b} N^{\Phi}_{X}(a,b) \leq |X| - |X'| + \max_{a,b} N^{\Phi'}_{X'}(a,b) \tag{10}$$

From Equation 9, $\frac{|X|}{2} - \max_{a,b} N^{\Phi}_{X}(a,b) \leq \tau$

$$\text{i.e. } \frac{|X|}{2} - \tau \quad \leq \quad \max_{a,b} N^{\Phi}_{X}(a,b) \tag{11}$$

Combining Equation 10 and Equation 11,

$$\frac{|X|}{2} - \tau \leq \max_{a,b} N^{\Phi}_{X}(a,b) \leq |X| - |X'| + \max_{a,b} N^{\Phi'}_{X'}(a,b) \tag{12}$$

By transitivity and regrouping, $\max_{a,b} N_{X'}^{\Phi'}(a,b) \geq \frac{|X|}{2} - \tau - |X| + |X'|$

$$\text{i.e. } \max_{a,b} N_{X'}^{\Phi'}(a,b) \quad \geq \quad |X'| - \tau - \frac{|X|}{2} \tag{13}$$

Given a partial S-box assignment Φ' with variables in X', by the end of the construction of any solution Φ obtained by extending Φ', the following inequality holds: $OF \leq OG$.

$$\text{i.e. } N_{X'}^{\Phi'}(a,b) \quad \leq \quad N_{X}^{\Phi}(a,b) \tag{14}$$

This is true for all selectors a and b, and in particular, holds for the maximum value of $N_{X'}^{\Phi'}(a,b)$ (resp. $N_{X}^{\Phi}(a,b)$) over all a, b:

$$\max_{a,b} N_{X'}^{\Phi'}(a,b) \leq \max_{a,b} N_{X}^{\Phi}(a,b) \tag{15}$$

From Equation 9, $\max_{a,b} N_{X}^{\Phi}(a,b) - \frac{|X|}{2} \leq \tau$

$$\text{i.e. } \max_{a,b} N_{X}^{\Phi}(a,b) \quad \leq \quad \frac{|X|}{2} + \tau \tag{16}$$

Combining Equations 15 and 16,

$$max_{a,b} N_{X'}^{\Phi'}(a,b) \leq \max_{a,b} N_{X}^{\Phi}(a,b) \leq \frac{|X|}{2} + \tau \tag{17}$$

The result follows by combing Equation 13 and Equation 17.

Q.E.D.

5 Results

The experimentation setup consists of an Intel Pentium Core-2 Duo 3-GHz CPU, 3.3 GB RAM and GNU/Linux with kernel version 2.6.28-11. The constraints are precompiled for DES criteria **S-3**, **S-4**, **S-5**, **S-6** and **S-7**. The precompiled constraints are fed to our implementation of a solver that supports Maintenance of Arc Consistency (MAC) with AC2001 [2]. The soft constraint of Equation 7 modeling **S-2** is transformed into a hard constraint by setting the threshold value for τ. We experiment with $\tau = 16$ and $\tau = 10$.

Better-quality S-boxes based on the score The score for the standard 3DES S-box S_4 is found to be 10 (minimum), while the score for S_7 is 18 (maximum). *Our approach yielded S-boxes with score 8, superior in quality to any of the standard 3DES S-boxes*. Fig. 4 reports one such S-box.

Performance Statistics The MAC solver is initially started only with the binary constraints. We test three heuristics for integrating the n-ary constraints in this solver.

0	3	5	6	9	10	15	12	7	4	14	13	2	1	8	11
3	0	6	5	10	9	12	15	4	7	13	14	1	2	11	8
3	15	0	12	5	6	9	10	4	8	7	11	14	13	2	1
0	12	3	15	9	10	5	6	7	11	4	8	2	1	14	13

Fig. 4. A 6×4 S-box with score 8, generated by our CSP solver

Time (hrs)	$r^{(6\times4)} \times 10^{49}$	S**-box Count**	
		$\sigma_X(\Phi) = 10$	$\sigma_X(\Phi) = 8$
1	$355,940$	$8,562$	$3,583$
2	$572,810,000$	$17,827$	$4,999$
3	$646,070,000$	$27,875$	$7,836$
4	$688,140,000$	$37,875$	$10,883$
5	$1,030,000,000$	$47,671$	$13,602$

Table 3. Solver Performance Using Incomplete, Incremental Heuristic $H_I^{64,10}$

- *Complete, Non-incremental heuristic,* $H_S^{\phi,\tau}$. This is the basic case where the n-ary constraint for **S-2** is checked only after all assignments, without using them in any domain-filtering.
- *Incomplete, Incremental heuristic,* $H_I^{\phi,\tau}$. At each node in the search tree, incrementally assign and check if the constraint in Equation 7 is partially satisfied. On violation, abandon the assignment and proceed with the next one.
- *Complete, Incremental heuristic* $H_C^{\phi,\tau}$. At each node in the search tree, project the constraint in Equation 7 by enforcing Property 3 on the current partial assignment.

Within the first hour, with a threshold $\tau = 10$ specified, the incomplete, incremental heuristic $H_I^{64,10}$ found around $3,600$ 6×4 S-boxes with the "best" score equal to 8. This count went up to more than $13,500$ in the 5-hour run that Table 3 reports.

Although this heuristic yields S-boxes with the "best" score, it is not complete. In order to know whether we have found the optimal quality S-boxes we would have to exhaust the whole search space. If the search space is too large to be exhausted, we would like to at least know what fraction of this search space we have managed to explore, as a measure of the probability that the optimal solution could have been found.

We therefore quantify the size of the search space, as the total number of potential S-boxes. As shown later, the search space of our problem instances is very large, and additional research is needed in order to be able to exhaust it. Assuming that the solver is systematic and chronological (visiting alternatives in lexicographic order), each partial or full assignment of values to all variables (whether it satisfies the constraints or not), and visited or skipped by the search tree, is defining a traversed distance (explored search space):

$$S_p^{(n\times m)} = \sum_{i=0}^{|X'|-1} x_i \cdot (2^m)^{|X'|-i-1} \tag{18}$$

Time	**Non-incremental** ($H_S^{64,16}$)		**Incremental** ($H_C^{64,16}$)	
(hrs)	$r^{(6\times4)} \times 10^{49}$	S**-box Count**	$r^{(6\times4)} \times 10^{49}$	S**-box Count**
1	1.198	4	102,160	20,786
2	21.725	14	265,040	35,957
3	42.091	15	915,420	49,110
4	42.091	26	993,950	80,933
5	61.340	40	1,061,500	94,069

Table 4. Solver Performance Using Complete Heuristics, with S-box threshold $\tau = 16$.

Here, $X' \subseteq X$ is a set of already-instantiated variables in the current partial assignment, $x_i \in X'$ is assigned a specific value from its domain D, and in Equation 18, x_i stands for the specific value assigned to the variable x_i.

With dynamic reordering of values and variables, Equation 18 still applies and one only has to use the current order.

For 6×4 S-boxes, $S_p^{(6\times4)}$ evaluates to 78-digit base-10 numbers. Given the large size of this search space, distances typically covered by the MAC solver in reasonable time differed only in their last few assignments (78-digit numbers differed in approximately the last 15 digits). Sometimes, certain constraints rule out much larger areas of the search space. To conveniently report this, we define a *search offset* metric S-box $S_{p_1}^{(n\times m)}$:

$$r^{(n\times m)} = \frac{S_p^{(n\times m)} - S_{p_1}^{(n\times m)}}{2^{m\times 2^n}} \tag{19}$$

Here, $S_{p_1}^{(n\times m)}$ denotes the value for $S_p^{(n\times m)}$ (determined from Equation 18) for the first S-box obtained by the solver. The solver has yielded $S_{p_1}^{(6\times4)} \approx \texttt{0x033} \times \texttt{16}^{\texttt{60}}$. (The hexadecimal form is for convenience and S_{p_1} could be alternatively written in decimal.) The difference between $S_{p_1}^{(6\times4)}$ for the incomplete and complete heuristics is $\approx 3 \times 16^{52}$ even when they use the same value for τ (graphs not shown due to lack of space). Table 3 reports the (scaled) search offsets of the solver using incomplete heuristics.

Performance Analysis of the three Heuristics Table 3 reports performance of the incomplete, incremental heuristic, with threshold $\tau = 10$. Table 4 compares the non-incremental and complete, incremental heuristics. The quantities reported at each hour represent the (scaled) fraction r of Equation 19, and the number of S-boxes generated up to that point in each case. Based on a 5-hour run of the experiment, the complete, incremental heuristic is observed to relatively vary between a factor of 17 and 85 times faster than the non-incremental heuristic (in terms of size of explored search space). The number of S-boxes generated is observed to correspondingly increase by an average factor of over 3,300. We tried all values of τ and report in Table 4 only for $\tau = 16$.

6 Conclusion

A soft global nonlinearity n-ary constraint, is projected onto fewer variables and thereby applied for dynamic domain filtering during search. This heuristic yielded a 17–85-fold relative increase in 6×4 S-box generation efficiency.

References

1. Data encryption standard (DES). Federal Information Processing Standard 46-2 (January 1988)
2. Bessière, C., Régin, J.C.: Refining the basic constraint propagation algorithm. In: Nebel, B. (ed.) IJCAI. pp. 309–315. Morgan Kaufmann (2001)
3. Clark, J., Jacob, J., Maitra, S., Stanica, P.: Almost boolean functions: the design of boolean functions by spectral inversion. Evolutionary Computation 3, 2173–2180 Vol.3 (Dec 2003)
4. Coppersmith, D.: The data encryption standard (des) and its strength against attacks. IBM J. Res. Dev. 38(3), 243–250 (1994)
5. Feistel, H.: Cryptography and computer privacy 228, 15–23 (1973)
6. Heys, H.M.: A tutorial on linear and differential cryptanalysis. Cryptologia XXVI(3), 189–221 (2002)
7. Matsui, M.: Linear cryptanalysis method for des cipher. In: EUROCRYPT '93: Workshop on the theory and application of cryptographic techniques on Advances in cryptology. pp. 386–397. Springer-Verlag (1994)
8. O'Connor, L.: Properties of linear approximation tables 1008 (1995)
9. Ramamoorthy, V., Silaghi, M., Matsui, T., Hirayama, K., Yokoo, M.: The design of cryptographic s-boxes using csps. In: CP (to appear) (2011)
10. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal 28, 656–715 (1949)

Inference Schemes for M Best Solutions for Soft CSPs

Emma Rollon[1], Natalia Flerova[2] and Rina Dechter[2]

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

erollon@lsi.upc.edu,{nflerova,dechter}@ics.uci.edu

Abstract. The paper present a formalization of the m-best task within the unifying framework of semirings. As a consequence, known inference algorithms are defined and their correctness and completeness for the m-best task are immediately implied. We also describe and analyze a Bucket Elimination algorithm for solving the m-best task, *elim-m-opt*, presented in an earlier workshop[1] and introduce an extension to the mini-bucket framework, yielding a collection of bounds for each of the m-best solutions. Some empirical demonstration of the algorithms and their potential for approximations are provided.

1 Introduction

Given an optimization problem, the objective is typically 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. Such a set can be useful, for example, in assessing the sensitivity of the optimal solution to variation of the parameters of the problem, or when a set of diverse assignments with approximately the same cost is wanted.

Lawler [11] provided a general scheme for using any optimization algorithm to solve the m-best task. Its main idea is to compute the m-best solutions by successively computing the best solution, each time using a slightly different reformulation of the original problem. This approach has been extended and improved over the years and is still one of the primary strategies to date for finding the m-best solutions. The approach used in this paper is to develop direct algorithms that avoid the repeated computation inherent in Lawler's scheme. The main idea is to integrate the m-best task into existing optimization schemes. In particular we focus on graphical models. This work is the continuation of our previous efforts [7], [8], where we derived and analyzed algorithm *elim-m-opt* that extends the widely-used Bucket Elimination (BE) to compute the m-best solutions by a relatively simple modification of its underlying combination and marginalization operators [4] and proposed extensions to Mini-Bucket Elimination to compute bounds on each of the m-best solutions, yielding algorithm *mbe-m-opt*.

[1] [7]

The main contribution of this paper is the formalization of the m-best task within the framework of semirings [14, 1, 10, 2]. This unifying formulation ensures the soundness and correctness of inference algorithms applied to any problem that fits into the framework. In particular, we show that *elim-m-opt* solves the m-best optimization task and we provide new empirical analysis for *mbe-m-opt* demonstrating its effectiveness both as an exact and approximation scheme.

2 Background

We consider problems expressed as graphical models. Let $\mathbf{X} = (X_1, \ldots, X_n)$ be an ordered set of variables and $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ an ordered set of domains. Domain $\mathbf{D}_i$ is a finite set of potential values for X_i. The assignment of variable X_i with $a \in \mathbf{D}_i$ is noted $(X_i = a)$. A tuple is an ordered set of assignments to different variables $(X_{i_1} = a_{i_1}, \ldots, X_{i_k} = a_{i_k})$. A *complete assignment* to all the variables in $\mathbf{X}$ is called a *solution*. Let t and s be two tuples having the same instantiations to the common variables. Their join, noted $t \cdot s$, is a new tuple which contains the assignments of both t and s (this notation is also used for multiplication, so we assume the meaning will be clear from the context). If t is a tuple over a set $T \subseteq X$ and $\mathbf{S}$ is a set of variables, then $t_{[S]}$ is a relational projection of t on $\mathbf{S}$.

We denote by $\mathbf{D}_{\mathbf{Y}}$ the set of tuples over a subset of variables $\mathbf{Y}$, also called the *domain of Y*. Let $f : \mathbf{D}_{\mathbf{Y}} \rightarrow \mathbf{A}$ be a function defined over $\mathbf{Y}$. $\mathbf{A}$ is a set of elements called *valuations* . Typical sets of valuations $\mathbf{A}$ are natural, real and booleans. If $f : \mathbf{D}_{\mathbf{Y}} \rightarrow \mathbf{A}$ is a function the scope of f, denoted $var(f)$, is $\mathbf{Y}$. In the following, we will use $\mathbf{D}_f$ as a shorthand for $\mathbf{D}_{var(f)}$.

We assume two binary operations over valuations: $\otimes : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{A}$ called combination and $\oplus : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{A}$ called addition. Both operators are associative and commutative. Typical combination operators are sum and product over numbers, and logical and (i.e., $\wedge$) over booleans. Typical addition operators are min, max and sum over numbers and logical or (i.e., $\vee$) over booleans. We extend these operators to operate over functions.

Definition 1 (combination operator, marginalization operator). *Let $f : \boldsymbol{D}_f \rightarrow \boldsymbol{A}$ and $g : \boldsymbol{D}_g \rightarrow \boldsymbol{A}$ be two functions. Their* combination*, noted $f \bigotimes g$ is a new function with scope $var(f) \cup var(g)$, s.t. $\forall t \in \boldsymbol{D}_{var(f) \cup var(g)}$, $(f \bigotimes g)(t) = f(t) \otimes g(t)$. Let $f : \boldsymbol{D}_f \rightarrow \boldsymbol{A}$ be a function and $\boldsymbol{W} \subseteq \boldsymbol{X}$ be a set of variables. The* marginalization *of f over $\boldsymbol{W}$, noted $\Downarrow_{\boldsymbol{W}} f$, is a function whose scope is $var(f) - \boldsymbol{W}$, s.t. $\forall t \in D_{var(f) - \boldsymbol{W}}, (\Downarrow_{\boldsymbol{W}} f)(t) = \oplus_{t' \in D_W} f(t \cdot t')$.*

Definition 2 (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; $\boldsymbol{A}$ is a set of valuations; $\boldsymbol{F} = \{f_1, \ldots, f_r\}$ is a set of discrete functions where $var(f_j) \subseteq \boldsymbol{X}$ and $f_j : \boldsymbol{D}_{f_j} \rightarrow \boldsymbol{A}$; and $\bigotimes$ is a combination operator over functions as defined in Definition 1. The graphical model $\mathcal{M}$ represents the function $F(\boldsymbol{X}) = \bigotimes_{f \in \boldsymbol{F}} f$.*

Definition 3 (reasoning task).
A reasoning task *is a tuple* $P = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$ *where* $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes)$ *is a graphical model and* $\Downarrow$ *is a marginalization operator over functions as defined in Definition 1. The reasoning task is to compute* $F(\boldsymbol{X}) \Downarrow_{\boldsymbol{X}}$.

For a reasoning task $\mathcal{P} = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$ the choice of $(\mathbf{A}, \otimes, \oplus)$ determines the combination $\bigotimes$ and marginalization $\Downarrow$ operators over functions, and thus the nature of the graphical model and its reasoning task. For example, if $\mathbf{A}$ is the set of non-negative reals and $\bigotimes$ is product, the graphical model is a Markov network or a Bayesian network. If $\Downarrow$ is max, the task is to compute the Most Probable Explanation (MPE), while if $\Downarrow$ is sum, the task is to compute the Probability of the Evidence.

The correctness of the algorithmic techniques for computing a given reasoning task relies on the properties of its set of valuations and operators. These properties are axiomatically described by means of an algebraic structure over $(\mathbf{A}, \otimes, \oplus)$. In this paper we consider reasoning tasks $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$ such that their valuation structure $(\mathbf{A}, \otimes, \oplus)$ is a semiring. Several works [14, 1, 10] showed that the correctness of inference algorithms over a reasoning task P is ensured whenever P is defined over a semiring.

Definition 4 (semiring). *A commutative semiring is a triplet* $(\boldsymbol{A}, \otimes, \oplus)$ *which satisfies the following three axioms:*

> **A1**. *The operation* $\oplus$ *is associative, commutative and idempotent, and there is an additive identity element called* 0 *such that* $a \oplus 0 = a$ *for all* $a \in \boldsymbol{A}$. *In other words,* $(\boldsymbol{A}, \oplus)$ *is a commutative monoid.*
> **A2**. *The operation* $\otimes$ *is also associative and commutative, and there is a multiplicative identity element called* 1 *such that* $a \otimes 1 = a$ *for all* $a \in \boldsymbol{A}$. *In other words,* $(\boldsymbol{A}, \otimes)$ *is also a commutative monoid.*
> **A3**. $\otimes$ *distributes over* $\oplus$, *i.e.,* $(a \otimes b) \oplus (a \otimes c) = a \otimes (b \oplus c)$

Example 1. MPE task is defined over semiring $\mathcal{K} = (\mathtt{R}, \times, \max)$, a CSP is defined over semiring $\mathcal{K} = (\{0, 1\}, \wedge, \vee)$, and a Weighted CSP is defined over semiring $\mathcal{K} = (\mathtt{N} \cup \{\infty\}, +, \min)$. The task of computing the Probability of the Evidence is defined over semiring $\mathcal{K} = (\mathtt{R}, \times, +)$.

Bucket elimination (BE) [4] is a well-known inference algorithm that generalizes dynamic programming for many reasoning tasks.

Definition 5 (bucket elimination). *The input of BE is a reasoning task* $P = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$ *and an ordering* $o = (X_1, X_2, \ldots, X_n)$, *dictating an elimination order for* BE, *from last to first. Each function from* $\boldsymbol{F}$ *is placed in the bucket of its latest variable in* o. *The algorithm 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 functions in the* $\boldsymbol{B}_i$, *some of which are original* f_i'*s and some are earlier computed messages. The result of the computation is a new function, also called* message, *that is placed in the bucket of its latest variable in the ordering* o.

The message passing between buckets follows a bucket-tree structure.

Definition 6 (bucket tree). *Bucket elimination defines a* bucket tree*, where the bucket of each X_i is linked to the destination bucket of its message (called the parent bucket). A node of the bucket is associated with its bucket variable.*

Theorem 1 *[4] Given a reasoning task $\mathcal{P} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$, BE is sound and complete. The time and space complexity of $BE(\mathcal{P})$ is exponential in a structural parameter called* induced width*, which is the largest scope of all the functions computed.*

3 M-best Optimization Task

In this section we formally define the problem of finding a set of best solutions over an optimization task. We consider optimization tasks defined over a set of totally ordered valuations. In other words, we consider reasoning tasks where the marginalization operator $\Downarrow$ is *min* or *max*. Without loss of generality, in the following we assume minimization tasks (i.e., $\Downarrow$ is *min*).

Definition 7 (optimization task). *Given a graphical model $\mathcal{M}$, its optimization task is $P = (\mathcal{M}, min)$. The goal is to find a complete assignment t such that $\forall t' \in D_{\boldsymbol{X}}, F(t) \leq F(t')$. $F(t)$ is called the optimal solution.*

Definition 8 (m-best optimization task). *Given a graphical model $\mathcal{M}$, its m-best optimization task is to find m complete assignments $\boldsymbol{T} = \{t_1, \ldots, t_m\}$ such that $F(t_1) \leq, \cdots, \leq F(t_m)$ and $\forall t' \in \boldsymbol{D_X} \backslash T$ and $\forall t \in \boldsymbol{T}, F(t') \geq F(t)$. The solution is the set of valuations $\{F(t_1), \ldots, F(t_m)\}$, called m-best solutions.*

The main goal of this paper is to phrase the m-best optimization task as a reasoning task over a semiring, so that well known algorithms can be immediately applied to solve this task. Namely, given an optimization task P over a graphical model $\mathcal{M}$, we need to define a reasoning task P^m that corresponds to the set of m-best solutions of $\mathcal{M}$.

We introduce the set of ordered m-best elements of a subset $S \subseteq \mathbf{A}$.

Definition 9 (set of ordered m-best elements, m-space). *Let S be a subset of a set of valuation $\boldsymbol{A}$. The* set of ordered m-best elements *of S is $Sorted^m\{S\} = \{s_1, \ldots, s_j\}$ such that $s_1 \leq s_2 \leq \ldots \leq s_j$ where $j = m$ if $|S| \geq m$ and $j = |S|$ otherwise, and $\forall s' \notin Sorted^m\{S\}, s_j \leq s'$. The* m-space *of $\boldsymbol{A}$, denoted $\boldsymbol{A}^m$, is the set of subsets of ordered m-best elements of $\boldsymbol{A}$. Formally, $\boldsymbol{A}^m = \{S \subseteq A \mid Sorted^m\{S\} = S\}$.*

The combination and addition operators over the m-space $\mathbf{A}^m$, noted $\otimes^m$ and *sort*^{m} respectively, are defined as follows.

Definition 10 (combination and addition over the m-space). *Let $\boldsymbol{A}$ be a set of valuations, and $\otimes$ and min be its combination and marginalization operators, respectively. Let $S, T \in A^m$. Their* combination*, noted $S \otimes^m T$, is the set $Sorted^m\{a \otimes b \mid a \in S, b \in T\}$, while their* addition*, noted $sort^m\{S, T\}$, is the set $Sorted^m\{S \cup T\}$.*

Theorem 1. *The valuation structure $(\boldsymbol{A}^m, \otimes^m, sort^m)$ is a semiring.*

We will refer to functions over the m-space $\mathbf{A}^m$ $f : \mathbf{D}_f \rightarrow \mathbf{A}^m$ as *vector functions*. Abusing notation, we extend the $\otimes^m$ and $sort^m$ operators to operate over vector functions similar to how operators $\otimes$ and $\oplus$ were extended to operate over scalar functions in Definition 1.

Definition 11 (combination and marginalization over vector functions). *Let $f : \boldsymbol{D}_f \rightarrow \boldsymbol{A}^m$ and $g : \boldsymbol{D}_g \rightarrow \boldsymbol{A}^m$ be two vector functions. Their combination, noted $f \bigotimes g$, is a new function with scope $var(f) \cup var(g)$, s.t. $\forall t \in \boldsymbol{D}_{var(f) \cup var(g)}$, $(f \bigotimes g)(t) = f(t) \otimes^m g(t)$.*

Let $\boldsymbol{W} \subseteq \boldsymbol{X}$ be a set of variables. The marginalization of f over $\boldsymbol{W}$, noted $\underset{\boldsymbol{W}}{sort}^m\{f\}$, is a new function whose scope is $var(f) - \boldsymbol{W}$, s.t. $\forall t \in D_{var(f) - \boldsymbol{W}}$, $\underset{\boldsymbol{W}}{sort}^m\{f\}(t) = sort^m_{t' \in D_W} f(t \cdot t')$.

h_1:

X_1	X_2	
a	a	{2,4}
a	b	{1,3}
b	a	{4}
b	b	{3}

h_2:

X_2	
a	{1,3}
b	{1}

$h_1 \otimes^m h_2$:

X_1	X_2	
a	a	{3,5}
a	b	{2,4}
b	a	{5,7}
b	b	{4}

$sort^m_{X_2}\{h_1\}$:

X_1	
a	$\{1, 2\}$
b	$\{3, 4\}$

Fig. 1: Combination and marginalization over vector functions. For each pair of values of (X_1, X_2) the result of $h_1 \otimes^m h_2$ is an ordered set of size 2 obtained by pair-wise summation of the corresponding elements of h_1 and h_2. The result of $sort^m_{X_2}\{h_1\}$ is an ordered set containing the two lower values of function h_1 for each value of X_1.

Example 2. Figure 1 shows the combination and marginalization over two vector functions h_1 and h_2 for $m = 2$.

The m-best extension of an optimization problem $\mathcal{P}$ is a new reasoning task $\mathcal{P}^m$ that expresses the m-best task over $\mathcal{P}$.

Definition 12 (m-best extension). *Let $\mathcal{P} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$ be an optimization problem defined over a semiring $(\boldsymbol{A}, \otimes, min)$. Its* m-best extension *is a new reasoning task $\mathcal{P}^m = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}^m, \boldsymbol{F}^m, \bigotimes, sort^m)$ over semiring $(\boldsymbol{A}^m, \otimes^m, sort^m)$. Each function $f : \boldsymbol{D}_f \rightarrow \boldsymbol{A}$ in $\boldsymbol{F}$ is trivially transformed into a new vector function $f' : \boldsymbol{D}_f \rightarrow \boldsymbol{A}^m$ defined as $f'(t) = \{f(t)\}$. In words, function outcomes of f are transformed to singleton sets in f'. Then, the set $\boldsymbol{F}^m$ contains the new f' vector functions.*

The following theorem shows that the optimum of P^m corresponds to the set of m-best valuations of P.

Theorem 2 *Consider an optimization problem $\mathcal{P} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$ defined over a semiring $(A, \otimes, min)$. Let $\{F(t_1), \ldots, F(t_m)\}$ be its m-best solutions. Let P^m be the m-best extension of P. The optimization task P^m computes the set of m-best solutions of P. Formally,*

$$sort_X{}^m\{\overline{\bigotimes}_{f \in \boldsymbol{F}^m} f\} = \{F(t_1), \ldots, F(t_m)\}$$

It is easy to see how the same extension applies to maximization tasks. The only difference is the set of valuations selected by operator $sort^m$.

4 Algorithm *elim-m-opt*

In this section we extend the bucket elimination algorithm to solving the m-best reasoning task. We subsequently show the derivation of the algorithm through an example.

4.1 The Algorithm Definition

Consider an optimization task $\mathcal{P}$. The bucket-elimination algorithm *elim-m-opt* solving $\mathcal{P}^m$ (i.e., the m-best extension of P) is described in Algorithm 1. First, the algorithm transforms scalar functions in F to their equivalent vector functions as described in Definition 12. Then, the algorithm processes the buckets from last to first as usual, using the two new combination and marginalization operators $\overline{\bigotimes}$ and $sort^m$, respectively. Roughly, the elimination of variable X_i from a vector function will produce a new vector function λ_i such that $\lambda_i(t)$ will contain the m-best extensions of t to the eliminated variables $X_{i+1}, \ldots, X_n$ with respect to the subproblem below the bucket variable in the bucket tree.

Since we are interested in recovering at least one complete assignment for each m-best solution, the algorithm propagates the variable assignments along with the vector messages when processing each bucket. These variable assignments are generated using the *argsort*^{m} operator defined as follows.

Definition 13. *Operator $argsort^m_{X_i} f$ returns a vector function $\overline{x_i}(t)$ such that $\forall t \in D_{var(f) \setminus X_i}$, where $\langle f(t \cdot x_i{}^1), \ldots, f(t \cdot x_i{}^m)\rangle$, are the m-best valuations extending t to X_i.*

In words, $\overline{x_i}(t)$ is the vector of assignments to X_i that yields the m-best extensions to t.

The correctness of the algorithm follows from the formulation of the m-best optimization task as a reasoning task over a semiring.

Theorem 3 *Algorithm* elim-m-opt *is sound and complete for finding the m-best solutions over a graphical model.*

The details of how to efficiently compute combination and marginalization are one of the main contributions of our previous work [7]. We recap the main algorithmic issues and demonstrate the intuition behind the method in the following section by deriving *elim-m-opt* through an example. For clarity reasons, we omit the generation of actual m-best solution assignments.

Algorithm 1 elim-m-opt algorithm

```
Input: An optimization task $\mathcal{P} = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, min)$; An ordering of variables $o = \{X_1, \ldots, X_n\}$;
Output: A zero-arity function $\lambda_1 : \emptyset \to \mathbf{A}^m$ containing the solution of the $m$-best optimization task.
1: Initialize: Transform each function $f \in \mathbf{F}$ into a singleton vector function $h(t) = \{f(t)\}$; 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: Backward:
3: for $i \leftarrow n$ down to 1 do
4:     Generate $\lambda_i = sort^m_{X_i}(\overline{\bigotimes}_{f \in B_i} f)$
5:     Generate assignment $\overline{x_i} = argsort^m_{X_i}(\overline{\bigotimes}_{f \in B_i})$, concatenate with relevant elements of the previously generated assignment messages.
6:     Place $\lambda_i$ and corresponding assignments in the bucket of the largest-index variable in $var(\lambda_i)$
7: end for
8: Return: $\lambda_1$
```

4.2 Deriving the Algorithm Using an Example

Consider a graphical model with three functions $F = \{f_1(z,x), f_2(z,y), f_3(t,z)\}$, and its optimization task over semiring $(\mathtt{N} \cup \{\infty\}, +, min)$ (i.e., the task is to find the minimum cost assignment). Finding the m-best valuations of the function $F(t,z,x,y) = f_3(t,z) + f_1(z,x) + f_2(z,y)$ can be expressed as finding Sol, defined by $Sol = \underset{t,x,z,y}{sort}^{\,m}\Big(f_3(t,z) + f_1(z,x) + f_2(z,y)$.

Since operator $sort^m$ is an extention of operator min, it inherits its distributive properties over summation. Due to this distributivity, we can apply symbolic manipulation and migrate each of the functions to the left of the $sort^m$ operator over variables that are not in its scope. In our example we rewrite as:

$$Sol = \underset{t}{sort}^m \underset{z}{sort}^m \left(f_3(t,z) + (\underset{x}{sort}^m f_1(z,x)) + \left(\underset{y}{sort}^m f_2(z,y) \right) \right) \tag{1}$$

The output of $sort^m$ is a set, so in order to make equation 1 well defined, we replace the summation operator by the combination over vector functions as in Definition 11.

$$Sol = \underset{t}{sort}^m \underset{z}{sort}^m (f_3(t,z) \overline{\bigotimes} (\underset{x}{sort}^m f_1(z,x)) \overline{\bigotimes} (\underset{y}{sort}^m f_2(z,y))) \tag{2}$$

BE computes expression 2 from right to left, corresponding to elimination ordering $o = \{T, Z, X, Y\}$. Figure 2 shows the messages passed between buckets and its bucket tree under o. Bucket $\mathbf{B}_Y$ containing function $f_2(z,y)$ is processed first. The algorithm applies operator $\underset{y}{sort}^m$ to $f_2(z,y)$, generating a vector function called a *message* and denoted by $\overline{\lambda_Y}(z)$ which is placed in $\mathbf{B}_Z$. Note that this message associates each z with the vector of m best valuations of $f_2(z, y)$.Namely,

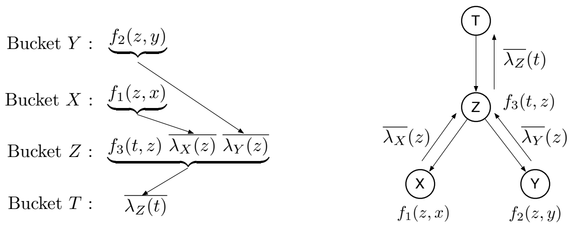
$$\underset{y}{sort^m} f_2(z, y) = (\lambda_Y^1(z), \ldots, \lambda_Y^j(z), \ldots, \lambda_Y^m(z)) = \overline{\lambda_Y}(z) \tag{3}$$

where for z each $\lambda_Y^j(z)$ is the j^{th} best value of $f_2(z, y)$. Similar computation is carried in $\mathbf{B}_X$ yielding $\overline{\lambda_X}(z)$ which is also placed in $\mathbf{B}_Z$.

When processing $\mathbf{B}_Z$, we need to compute, (see expression 2)

$$\overline{\lambda_Z}(t) = \underset{z}{sort^m} f_3(t, z) \overline{\bigotimes} \overline{\lambda_X}(z) \overline{\bigotimes} \overline{\lambda_Y}(z)$$

The result of the combination of the scalar function $f_3(t, z)$ with the two messages $\overline{\lambda_X}(z)$ and $\overline{\lambda_Y}(z)$ is a new vector function that has m^2 elements for each tuple (t, z). Applying $\underset{z}{sort^m}$ to the resulting combination generates the m best elements out of those m^2 yielding message $\overline{\lambda_Z}(t)$. As we show in [7], it is possible apply a more efficient procedure that would calculate at most $2m$ elements per tuple (t, z) instead. Finally, processing the last bucket yields the vector of m best solution costs for the entire problem: $Sol = \overline{\lambda_T} = \underset{t}{sort^m} \overline{\lambda_Z}(t)$ (see Figure 2a).


Bucket Y : $f_2(z,y)$
Bucket X : $f_1(z,x)$
Bucket Z : $f_3(t,z)$ $\overline{\lambda_X(z)}$ $\overline{\lambda_Y(z)}$
Bucket T : $\overline{\lambda_Z(t)}$
T
$\overline{\lambda_Z}(t)$
Z
$f_3(t,z)$
$\overline{\lambda_X}(z)$
$\overline{\lambda_Y}(z)$
X
Y
$f_1(z,x)$
$f_2(z,y)$

(a) Bucket messages (b) Bucket-tree

Fig. 2: Example of applying *elim-m-opt*

4.3 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 with largest domain size k, it is possible to efficiently compute a messages between two buckets in $O(k^{w^*} m \cdot deg_i \log m)$, yielding the total time complexity of *elim-m-opt* of $O(\sum_{i=1}^n k^{w^*} m \cdot deg_i \log m)$ as we showed in [7]. Assuming $deg_i \leq deg$ and since $\sum_{i=1}^n deg_i \leq 2n$, we get the total time complexity of $O(nmk^{w^*} \log m)$. The space complexity is dominated by the size of the messages between buckets, each containing m costs-to-go for each of $O(k^{w^*})$ tuples. Having at most n such message yields the total space complexity of $O(mnk^{w^*})$.

4.4 The Mini-Bucket for the m-best

Mini-bucket Elimination *(MBE)* [5] 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.

Theorem 4 *[5] Given a reasoning task $\mathcal{P}$, MBE computes a bound on $\mathcal{P}$. Given an integer control parameter z, the time and space complexity of MBE is exponential in z.*

Recall that throughout this paper, we are assuming minimization tasks. In this case, MBE computes a lower bound.

Algorithm *mbe-m-opt* (Algorithm 2) is a straightforward extension of MBE to solve the m-best reasoning task, where the combination and marginalization operators are the ones defined over vector functions. The input of the algorithm is an optimization task P, and its output is a *m-best bound* on the m-best solutions of P.

Definition 14 (m-best lower bound). *Let $S = \{a_1, \ldots, a_j\}$ and $T = \{b_1, \ldots, b_k\}$ be two sets of ordered m-best elements (i.e., $S, T \in \boldsymbol{A}^m$). S is a m-best lower bound of T iff: (i) $|S| \geq |T|$, (ii) $b_1, b_2, \ldots, b_{l-1} \in S$ and $b_l, b_{l+1}, \ldots, b_k \notin S$, and (iii) $a_j < b_l$ (where by definition $b_l = 0$ if $l - 1 = |T|$).*

The idea behind this definition is that S contains all elements in T from b_1 up to b_{l-1} plus some other elements, and the maximum element in S (i.e., a_j) is smaller than the first element in T not included in S (i.e., b_l). For example, $S = \{4, 6, 10\}$ is not a 3-best lower bound of $T = \{4, 7, 10\}$, but it is a 3-best lower bound of $R = \{4, 11\}$.

Theorem 5 (*mbe-m-opt* bound and complexity) *Given a minimization task P,* mbe-m-opt *computes an m-best lower bound on the m-best optimization task $\mathcal{P}^m$. 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.*

Sketch of proof. mbe-m-opt solves a relaxed version of the original problem. The relaxation is based on adding duplicates of the variables eliminated in different mini-buckets. In the limit (i.e., when m is infinity), the relaxed problem's solution set contains all solutions to the original problem (corresponding to assignments where duplicated variables take on the same domain value), plus a set of other solutions (corresponding to assignments where duplicated variables take on different domain values). When m is different to infinity, and depending on its value, the output of mbe-m-opt will contain all solutions to the original problem, some of them, or none. In all cases, the output satisfies the conditions to be an m-best lower bound of the set of m-best solutions to the original problem.

Algorithm 2 mbe-m-opt algorithm

```
Input: An optimization task $\mathcal{P} = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, min)$; 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 assign-
    ments 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 $\mathbf{B}_i$ along $o$.
2: Backward:
3: for $i \leftarrow n$ down to 1 (Processing bucket $B_i$) do
4:     Partition functions in bucket $B_i$ into $\{Q_{i_1}, ..., Q_{i_l}\}$, where each $Q_{i_j}$ has no more
       than $z$ variables.
5:     Generate cost messages $\lambda_{i_j} = sort^m_{X_i}(\bigotimes_{f \in Q_{i_j}} f)$ and place each in the largest
       index variable in $var(Q_{i_j})$
6: end for
7: Return: The set of all buckets, and the vector of m-best costs bounds in the first
   bucket.
```

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

Here is a simple, but quite fundamental observation. Recall that whenever 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. We next discuss the ramification of this observation.

Proposition 1. *Given the m-best solutions generated by* mbe-m-opt *(for clarity we consider minimization problem, the results can be extended for maximization)* $\tilde{C} = \{\tilde{p}_1 \leq \tilde{p}_2 \leq, ..., \leq \tilde{p}_m\}$*, let* p^{opt} *be the optimal value (the minimum cost) and let* j_0 *be the first index such that* $\tilde{p}_{j_0} = p^{opt}$*, or else we assign* $j_0 = m + 1$*. Then, if* $j_0 > m$*,* $\tilde{p}_m$ *is a lower 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* $\tilde{p}_1$*.*

Proof. Let $\tilde{C} = \{\tilde{p}_1 \leq \tilde{p}_2 \leq, ..., \leq \tilde{p}_{N_1}$ be an ordered set of costs of all tuples over the relaxed problem (with duplicate variables). By the nature of any relaxation, $\tilde{C}$ must also contain all the cost values associated with solutions of the original problem denoted by $C = \{p_1 \leq \cdots \leq p_{N_2}\}$. Therefore, if j_0 is the first index such that $\tilde{p}_{j_0}$ coincides with p^{opt}, then clearly for all $i < j_0$, $p^{opt} \geq \tilde{p}_i$ with $\tilde{p}_{j-1}$ being the tightest lower-bound. Also, when $j_0 > m$ we have $\tilde{p}_m \leq c^{opt}$

In other words if $j \leq m$, we already have the optimal value, otherwise we can use $\tilde{p}_m$ as our better lower bound. Such tighter bounds would be useful during search algorithm such as A*. It is essential therefore to decide efficiently if a bound coincides with the exact optimal cost. Luckily, the nature of the MBE relaxation supplies us with an efficient decision scheme.

Proposition 2. *Given a m-best lower bound produced by* mbe-m-opt $\tilde{p}_1 \leq \tilde{p}_2 \leq, ... \leq \tilde{p}_m$*, deciding if* $\tilde{p}_j = p^{opt}$ *can be done efficiently.*

Proof. *mbe-m-opt* provides both the bounds on the m-best costs and for each bound a corresponding tuple, where assignments to duplicated variables is 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. And, if no such tuple is observed, the optimal value is smaller than $\tilde{p}_m$. Since the above tests require just $O(nm)$ steps applied to m-best assignments already obtained in polytime, the claim follows.

5 Related work

Comparing with exact schemes. Lawler's approach, whose complexity is $O(nmT(n))$, where $T(n)$ is the complexity of finding a single best solution, was applied by Nilsson [12] to a join-tree. Nilsson utilizes the results from previous computations, achieving worst case complexity of $O(mT(n))$. If applied to a bucket-tree his algorithm dominates schemes mentioned here, with run time of $O(nk^{w*} + mn\log(mn) + mnk)$. Yanover and Weiss [15] developed a belief propagation approximation scheme for loopy graphs, called BMMF. When applied to juction tree it can function as an exact algorithm with complexity $O(mnk^{w*})$.

Two algorithms that are similar to elim-m-opt, both based on dynamic programming, are [13] and [6]. Seroussi and Golmard algorithm extracts the m solutions directly, by propagating the m best partial solutions along a junction tree that is pre-compiled. Given a junction tree with p cliques, each having at most *deg* children, the complexity of the algorithm is $O(m^2 p \cdot k^{w^*} deg)$. Elliot [6], explores the representation of Valued And-Or Acyclic Graph, i.e., smooth deterministic decomposable negation normal form (sd-DNNF) [3]. He propagates the m best solutions partial assignments to the problem variables along the DNNF structure which is pre-compiled as well. The complexity of Elliot's algorithm is $O(nk^{w^*} m \log m \cdot deg)$. Clearly our *elim-m-opt* algorithm does not boast the best complexity compared to the related methods. However, it demonstrates the direct applicability of established inference schemes to the generalized formulation of the m best solution problem as the *m-best reasoning problem.* Moreover, the main significance *elim-m-opt* is in the natural extension to an approximation scheme with guarantees on the solution quality that provides flexible trade off between accuracy and complexity.

Comparing with approximation schemes. In addition to BMMF, another extension of Nilsson's and Lawler's idea that yields an approximation scheme is an algorithm called STRIPES by [9]. They focus on m-MAP problem over binary Markov networks, solving each new subproblem by an LP relaxation. The algorithm solves the task exactly if the solutions to all LP relaxations are integral, and provides an upper bound of each m MAP assignments otherwise. In contrast, our algorithm *mbe-m-opt* can compute bounds over any graphical model (not only binary) and over a variety of m-best optimization tasks.

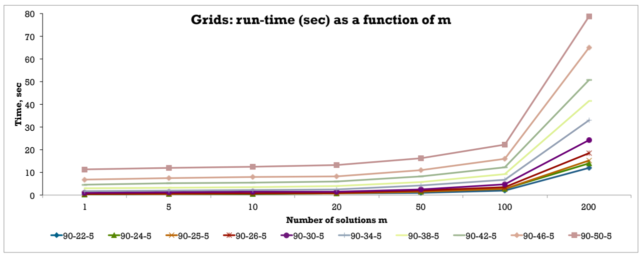


Fig. 3: *mbe-m-opt* run time (sec) as a function of number of solutions m for the grid instances. The z-bound=10, $m = [1, 5, 10, 20, 50, 100, 200]$, $n \in [500, 2500]$, $w^* \in [30, 74]$, $k = 2$.

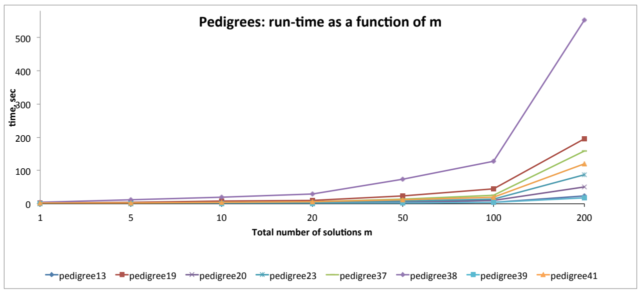


Fig. 4: *mbe-m-opt* run time (sec) as a function of number of solutions m for the pedigree instances. The z-bound=10, $m = \in [1, 5, 10, 20, 50, 100, 200]$, $n \in [400, 1272]$, $w^* \in [20, 30]$, $k = 4$.

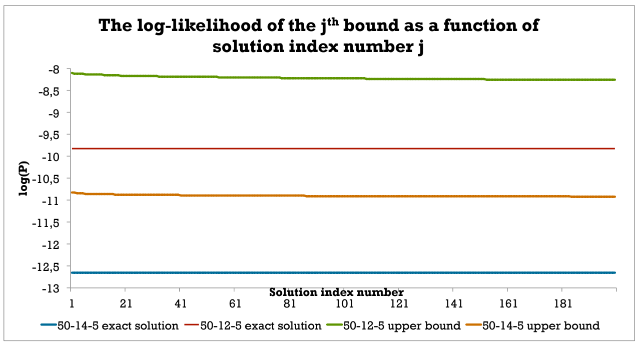


Fig. 5: The change in the cost of the j^{th} solution as j increases from 1 to 200 for two binary grid instances. Instance 50-12-5 has $n = 144$ and $w^* = 15$, 50-14-5 has n=196 and $w^* = 18$. The upper bounds outputted by *mbe-m-opt* with z-bound=10, the exact best solutions found by a search algorithm.

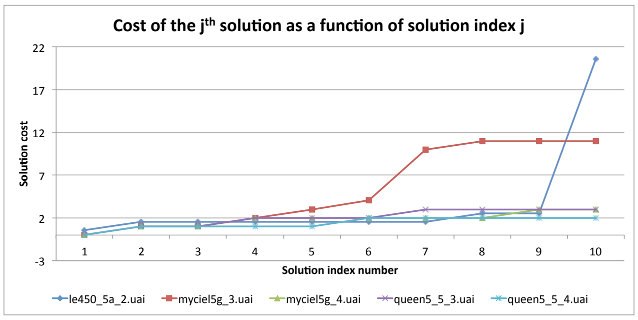


Fig. 6: The change in the cost of the j^{th} solution as j increases for chosen WCSP instances, $n \in [25, 475]$, $w^* \in [18, 293]$, $k \in [2, 4]$. Results obtained by *mbe-m-opt* with z-bound=10.

6 Empirical demonstrations

The first part of our experiments assumes 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. The first set contained grid instances with a hundred to 2.5 thousand variables and tree-width from 12 to 50, the second - pedigree instances with several hundred variables and tree width from 15 to 30. Those instances were taken from the UAI 2008 evaluation. For clarity and space reasons we present only a subset of instances illustrating typical behaviour. Figures 3 and 4 present the dependence of the run-time on m, for a few selected instances.

Figure 5 shows the change in the upper bound as a function of index of the solution j. For these grid instances as j increases, the bound on the cost of the j^{th} solution approaches the exact best solution, but extremely slowly. However, as can be seen in Figure 6, it is not the case for all type of instances. This figure depicts some of the results of the experiments on the set of weighted CSPs from UAI 2008 competition. The instances in question have from 25 to 450 variables, domain size 2-4 and induced width 18-293. We can see considerable differences between the costs of the 1^{st} and 10^{th} for some instances. This demonstrates that there is a potential of improving the bound on the optimal assignment using the m-best bounds as discussed in Section 4.5.

We carried some comparison with BMMF by [15] on randomly generated 10 by 10 grids for MPE task. The run times of the algorithms are not comparable since our algorithm is implemented in C and BMMF in Matlab, which is inherently slower. For most instances that *mbe-m-opt* can solve exactly in under a second, BMMF takes more than 5 minutes. 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 from viewing the two algorithms side by side as is demonstrated by a typical result in Figure 7. We know that in this case the solutions obtained with z-bound equal to 1000 are exact, while z-bound equal to 10 yields an upper bound. 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.

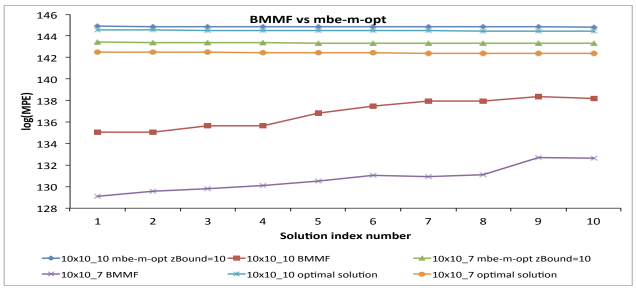


Fig. 7: Comparison of *mbe-m-opt* with z-bounds 10 and BMMF on random 10 by 10 grids. The exact solutions were obtained by *mbe-m-opt* with z-bound$> w^*$. While *mbe-m-opt* provides upper bounds on the solutions, BMMF gives no guarantees whether it outputs an upper or a lower bound. Also, its accuracy on these instances are clearly worse.

7 Conclusions

We presented a formulation of the m-best reasoning task within a framework of c-semiring. Such problem definition make existing inference and search algorithms immediately applicable for the task, as we demonstrated on the example of a new bucket-elimination algorithm for solving the m-best task over a graphical model, analyzed its performance and related it to other approaches in the literature.

The significance of the proposed algorithm is primarily in providing an inference framework for the m-best task that can both suggest approximation schemes and yield heuristic advice. Indeed, optimization tasks that seek a single optimal solution are solved far more effectively by search (e.g., branch and bound and best-first search), than by variable elimination, because they can benefit from the bounding power of the guiding cost function. It is also likely that search will be more effective for m-best task. The promise of the elim-m-opt inference algorithm is in its potential to yield viable lower- and upper-bounds for the m-best solutions via the mini-bucket algorithm, as we discussed.

Furthermore, it could also lead to loopy propagation message-passing schemes that are now the most common way for approximations in graphical models, since those schemes are relaxation of exact message-passing schemes such as bucket-elimination. In particular, our algorithm can be extended into a loopy max-prod for the m-best task. This approach will yield a direct loopy-propagation for the m-best reasoning problem, while the approach by Yanover and Weiss uses loopy max-prod for solving a sequence of optimization problems in the style of Lawler's approach. Moreover, all such approximation extensions would be applicable to the broad range of graphical models captured by the unifying framework of c-semiring. 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. Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.
2. S. Bistarelli, H. Faxgier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparison. *Over-Constrained Systems*, pages 111–150, 1996.
3. A. Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.
4. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85, 1999.
5. R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.
6. P.H. Elliott. Extracting the K Best Solutions from a Valued And-Or Acyclic Graph. Master's thesis, Massachusetts Institute of Technology, 2007.
7. N. Flerova and R. Dechter. M best solutions over Graphical Models. In *CRAGS10 Workshop*, 2010.
8. N. Flerova, R. Dechter, and E. Rollon. Bucket and mini-bucket schemes for m best solutions over graphical models. In *GKR'11 Workshop*, 2011.
9. M. Fromer and A. Globerson. An LP View of the M-best MAP problem. *Advances in Neural Information Processing Systems*, 22:567–575, 2009.
10. J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.
11. E.L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.
12. D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159–173, 1998.
13. B. Seroussi and JL Golmard. An algorithm directly finding the K most probable configurations in Bayesian networks. *International Journal of Approximate Reasoning*, 11(3):205–233, 1994.
14. G. R. Shafer and P.P. Shenoy. Probability propagation. *Anals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.
15. 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.