Dynamic Programming on Nominal Graphs[*]

Nicklas Hoch
Volkswagen AG, Corporate Research Group
nicklas.hoch@volkswagen.de

Ugo Montanari
University of Pisa, Computer Science Department
ugo@di.unipi.it

Matteo Sammartino
Radboud University, Institute for Computing and Information Sciences
m.sammartino@cs.ru.nl

Many optimization problems can be naturally represented as (hyper) graphs, where vertices correspond to variables and edges to tasks, whose cost depends on the values of the adjacent variables. Capitalizing on the structure of the graph, suitable dynamic programming strategies can select certain orders of evaluation of the variables which guarantee to reach both an optimal solution and a minimal size of the tables computed in the optimization process. In this paper we introduce a simple algebraic specification with parallel composition and restriction whose terms up to structural axioms are the graphs mentioned above. In addition, free (unrestricted) vertices are labelled with variables, and the specification includes operations of name permutation with finite support. We show a correspondence between the well-known tree decompositions of graphs and our terms. If an axiom of scope extension is dropped, several (*hierarchical*) terms actually correspond to the same graph. A suitable graphical structure can be found, corresponding to every hierarchical term. Evaluating such a graphical structure in some target algebra yields a dynamic programming strategy. If the target algebra satisfies the scope extension axiom, then the result does not depend on the particular structure, but only on the original graph. We apply our approach to the parking optimization problem developed in the ASCENS e-mobility case study, in collaboration with Volkswagen. Dynamic programming evaluations are particularly interesting for autonomic systems, where actual behavior often consists of propagating local knowledge to obtain global knowledge and getting it back for local decisions.

1 Introduction

Many optimization problems can naturally be represented as hypergraphs, where each hyperedge is an atomic subproblem and is attached to vertices corresponding to the problem's variables. However, these hypergraphs often lack an algebraic structure, which would allow for the recursive resolution of problems, and are not able to represent the *secondary optimization problem*, that is: finding an optimal *variable elimination strategy*. The order in which variables are eliminated may dramatically affect the computation cost of solutions.

In this paper we introduce a simple algebraic specification for representing optimization problems. It is similar to a process calculus, based on nominal structures (namely *permutation algebras*, see e.g. [21, 14]). Optimization problems are represented as terms over the variables of the problem, consisting of the parallel composition of subproblems. A key feature is the representation of variable elimination via the restriction operator. In fact, since a restricted variable can only occur in its scope, its value can be determined when solving the subproblem it encloses. For instance

$$(x_1)((x_2)A(x_1,x_2) \parallel (x_3)B(x_1,x_3))$$

[*] Research supported by the EU Integrated Project 257414 ASCENS and by the Italian MIUR Project CINA (PRIN 2010), grant number 2010LHT4KM.

A. Rensink & E. Zambon (Eds.): Graphs as Models 2015 (GaM'15)
EPTCS 181, 2015, pp. 80–96, doi:10.4204/EPTCS.181.6

represents a problem with three variables x_1,x_2,x_3 and two atomic problems A,B, sharing x_1. One can solve the subproblem $(x_2)A(x_1,x_2)$ w.r.t. x_2 right away, thus eliminating x_2 and obtaining a solution $A'(x_1)$ parametric w.r.t. x_1; similarly for $(x_3)B(x_1,x_3)$. Then x_1 can be eliminated in $(x_1)(A'(x_1) \parallel B'(x_1))$, which yields the global solution.

The specification provides axioms that are common in process algebras, and have a natural interpretation in terms of optimization problems. For instance: subproblems can be solved in any order, the formal names of their variables are irrelevant and so on. In particular, *scope extension* (actually, scope reduction) becomes a crucial operation: it allows variables to be eliminated *earlier*, when solving a smaller subproblem. This produces a more efficient solution of the secondary optimization problem of dynamic programming.

The computation of an optimal solution is formalized as an evaluation of terms in a suitable domain of *cost functions*, giving a cost to each assignment of free variables. This domain is indeed an algebra for the specification: constructors are interpreted as operations over cost functions, performing optimization steps, and axioms become useful properties. In particular, they tell that cost functions are preserved under rearranging the structure of the problem to get a more efficient computation. Moreover, the underlying nominal structure provides a notion of *support*, that is the set of variables that are really relevant for the computation. This is the key for a finite, efficient representation of cost functions.

Then we introduce a graphical notation for optimization problems, called *nominal hypergraphs*. They are hypergraphs with an *interface* where names are assigned to some vertices. These vertices are variables for the overall problem. Other vertices are variables of subproblems that must be eliminated in the optimization process.

We show that nominal hypergraphs can be given an algebraic structure, and that (isomorphic) nominal hypergraphs can be described by (congruent) terms of our specification. This allows us to recursively compute the cost function of a nominal hypergraph by performing the computation on any of the corresponding terms. Moreover, we show that well-known structures to represent parsing of graphs, namely *tree decompositions* [22, 16], can be represented in our framework as terms.

However, nominal graphs still lack a description of the variable elimination strategy. We describe this information as *hierarchical* nominal hypergraphs, that are trees describing the decomposition of a nominal hypergraph in terms of nested components, each corresponding to a subproblem. Such trees correspond to terms without the scope extension axioms, i.e., where the scope of restrictions is fixed. A bottom-up visit of a hierarchical nominal graph yields a dynamic programming algorithm with the given variable elimination strategy.

We apply our approach to the *e-mobility case study* from the *Autonomic Service-Component Ensembles* (ASCENS) project. In ASCENS, systems are modeled as self-aware, self-adaptive and autonomic components running in ensembles (dynamic aggregations of components), which through interactions among them and with the environment accomplish both individual (local) and collective (global) goals by optimizing the use of resources.

In the e-mobility case study, carried on in collaboration with Volkswagen, the traffic system is modeled as ensembles of electrical vehicles with the goal to optimize the usage of resources (electricity, parking places, etc.), while ensuring the fulfillment of individual goals (such as reaching in time the destination) and collective goals (avoiding traffic jams or guaranteeing that all vehicles find a spot where to park). So in general, besides optimizing local resources, for example by finding the best trips and journeys for each vehicle, the e-mobility case study aims at solving global problems, involving large ensembles of different vehicles. In [15] several optimization problems are presented for the e-mobility case study. In [8] a parking problem is considered. In the formulation we present in this paper, parking systems are regarded as nominal hypergraphs, where each hyperedge is a parking zone and vertices are cars that may be parked inside the zone. Their term representation is interpreted as functions telling the cost of parking cars inside or outside certain parking zones. Thus cost functions only have binary arguments. While more efficient, this domain choice yields operators remarkably neater than those of the classical point-wise interpretation and our framework is flexible enough to accommodate them.

2 An algebraic specification for optimization problems

We introduce an algebraic specification for describing the structure of optimization problems. Variables are represented as *names*, belonging to an enumerable set $\mathcal{N}$. We write $Perm(\mathcal{N})$ for the set of permutations over $\mathcal{N}$, i.e., bijective functions $\pi{:}\mathcal{N} \to \mathcal{N}$.

Definition 1 (Optimization signature)**.** *Let C be a set of constants denoting atomic problems, equipped with an arity function $ar : C \to \mathbb{N}$ telling how many variables each problem involves. We assume the* empty problem *nil, with $ar(nil) = 0$. The* optimization signature *is given by the following grammar*

$$p,q := p \parallel q \mid (x)p \mid p\pi \mid A(\tilde{x}) \mid nil$$

where $A \in C$, $\pi \in Perm(\mathcal{N})$, $\{x\} \cup \tilde{x} \subseteq \mathcal{N}$ and $|\tilde{x}| = ar(A)$ (we overload the notation $\tilde{x}$ to indicate both a vector and a set of names).

Here:

- the *parallel composition* $p \parallel q$ represents the problem consisting of two subproblems p and q, possibly sharing some variables;
- the *restriction* $(x)p$ is p where the assignment for x has already been determined;
- the *permutation* $p\pi$ is p where variable names have been exchanged according to π;
- the *atomic problem* $A(\tilde{x})$ represents a problem that only involves the problem A over variables $\tilde{x}$;
- *nil* represents the *empty problem*.

We assume restriction has precedence over parallel composition.

Free names of p are recursively defined as follows

$$fn(p \parallel q) = fn(p) \cup fn(q) \qquad fn((x)p) = fn(p) \setminus \{x\} \qquad fn(p\pi) = \pi(fn(p))$$
$$fn(A(\tilde{x})) = \tilde{x} \qquad fn(nil) = \varnothing$$

We consider syntax up to structural congruence axioms shown in fig. 1. The operator $\parallel$ forms a commutative monoid, meaning that problems in parallel can be solved in any order $(\mathbf{AX}_{\parallel})$. Restrictions can be α-converted $(\mathbf{AX}_{\alpha})$, i.e. names of assigned variables are irrelevant. Restrictions can also be swapped, i.e., assignments can happen in any order, and removed, whenever their scope is *nil* $(\mathbf{AX}_{(x)})$. The scope of restricted variables can be narrowed to terms where they occur free $(\mathbf{AX}_{SE})$. Axioms regarding permutations say that identity and composition behave as expected $(\mathbf{AX}_{\pi})$ and that permutations distribute over syntactic operators $(\mathbf{AX}^{p}_{\pi})$. Permutations are assumed to behave in a capture avoiding way when applied to $(x)p$. We call *optimization algebraic specification* the specification made of the optimization signature and the congruence axioms, and *optimization terms* the terms for the specification.

We include permutations in the specification because they provide a general mechanism to compute the set of "free" names in any algebra, called *(minimal) support*.

Definition 2 (Support)**.** *Let A be an algebra for the optimization specification, and let π^A be the interpretation of π in A. We say that $X \subset \mathcal{N}$* supports *$a \in A$ whenever, for all permutations π acting as the identity on X, we have $a\pi^A = a$. The minimal support $supp(a)$ is the intersection of all sets supporting a.*

For instance, let π^t be the interpretation of π on optimization terms: given a term p, $p\pi^t$ applies π to all free names of p in a capture avoiding way. It is easy to verify that $supp(p) = fn(p)$.

$$
\begin{array}{rl}
(\mathbf{AX}_{\|}) & p \parallel q \equiv q \parallel p \qquad (p \parallel q) \parallel r \equiv p \parallel (q \parallel r) \qquad p \parallel nil \equiv p \\
(\mathbf{AX}_{(x)}) & (x)(y)p \equiv (y)(x)p \qquad (x)nil \equiv nil \\
(\mathbf{AX}_{\alpha}) & (x)p \equiv (y)p[x \mapsto y] \qquad (y \notin fn(p)) \\
(\mathbf{AX}_{SE}) & (x)(p \parallel q) \equiv (x)p \parallel q \qquad (x \notin fn(q)) \\
(\mathbf{AX}_{\pi}) & p\,\mathsf{id} \equiv p \qquad (p\pi')\pi \equiv p(\pi \circ \pi') \\
(\mathbf{AX}^{p}_{\pi}) & \begin{array}{c} A(x_1,\dots,x_n)\pi \equiv A(\pi(x_1),\dots,\pi(x_n)) \qquad nil\,\pi \equiv nil \qquad (p \parallel q)\pi \equiv p\pi \parallel q\pi \\ ((x)p)\pi \equiv (x)p\pi' \qquad (\pi'(x) = x, \pi'(y) = \pi(y) \text{ for } x \neq y) \end{array}
\end{array}
$$

Figure 1: Structural congruence axioms of the optimization specification.

2.1 Hierarchical optimization specification

The scope of restrictions determines a solution for the *secondary optimization problem*, because it specifies when restricted variables should be eliminated. However, the presence of $(\mathbf{AX}_{SE})$ identifies terms corresponding to different solutions. We call *hierarchical optimization specification* the optimization specification without $(\mathbf{AX}_{SE})$, and hierarchical terms its freely generated terms.

We are interested in two forms of hierarchical terms.

Definition 3 (Normal and canonical forms)**.** *A term is said to be in* normal form *whenever it is of the form*

$$(\tilde{x})(A_1(\tilde{x}_1) \parallel A_2(\tilde{x}_2) \parallel \cdots \parallel A_n(\tilde{x}_n))$$

with $A_i \in C$ $(i = 1,\dots,n)$ and $\tilde{x} \subseteq \tilde{x}_1 \cup \cdots \cup \tilde{x}_n$. It is in canonical form *whenever it is obtained by the repeated application to a non-hierarchical term of* $(\mathbf{AX}_{SE})$*, from left to right, until termination. For both forms, we assume that subterms of the form $(\tilde{x})nil$ (where $\tilde{x}$ may be empty) are removed using* $(\mathbf{AX}_{(x)})$ *and* $(\mathbf{AX}_{\|})$*.*

Normal and canonical forms are somewhat dual: normal forms have all restrictions at the top level, whereas in canonical forms every restriction (x) is as close as possible to the atomic terms where x occurs (if any). A term in normal form is intuitively closer to a typical optimization problem: $\tilde{x}$ specifies which variables should be assigned, and the term in its scope represents subproblems and their connections. In a term in canonical form, variables are eliminated as soon as possible. Notice that a term may have more than one canonical form, whereas normal forms are unique (up to the hierarchical optimization specification congruence).

Remark 1. *Hierarchical terms in normal and canonical form can be regarded as canonical representatives of $\equiv$-classes (recall that $\equiv$ is the structural congruence of fig. 1), because $\equiv$ is coarser than the hierarchical optimization specification congruence.*

3 Optimization problems as nominal hypergraphs

Recall that a hypergraph G is a triple $(V_G, E_G, a_G\colon E_G \to V_G^\star)$, where V_G is the set of vertices, E_G is the set of hyperedges and, for each $e \in E_G$, $a_G(e)$ is the tuple of vertices attached to e ($V_G^\star$ is the set of tuples over V_G). Let $\mathcal{E}$ be a set of edge labels, equipped with a function $ar\colon \mathcal{E} \to \mathbb{N}$ telling the number of vertices $ar(l)$

of an edge with label l. A *labeled* hypergraph G is a hypergraph G plus a function $lab: E_G \to \mathcal{E}$ mapping each hyperedge $e \in E_G$ to its label l such that $|a_G(e)| = arl(l)$. Given two (labeled) hypergraphs G_1 and G_2, we write $G_1 \uplus G_2$ for their component-wise disjoint union.

Optimization problems can naturally be seen as hypergraphs labeled over atomic subproblems, where vertices correspond to variables. We introduce a notion of labeled hypergraph where some vertices are associated variable names.

Definition 4 (Labeled nominal hypergraph and their morphisms)**.** *A* labeled nominal hypergraph *(NH-graph in short) is a pair $\eta \triangleright G$, where G is a labeled hypergraph without isolated vertices and η is a partial injection from V_G to $\mathcal{N}$, assigning names to some vertices of G. The set $img(\eta)$ is called the* interface *of G and $def(\eta)$ (the domain of definition of η) are called* interface vertices*. Given two NH-graphs $\eta_1 \triangleright G_1$ and $\eta_2 \triangleright G_2$, a NH-graph morphism $h: \eta_1 \triangleright G_1 \to \eta_2 \triangleright G_2$ is a homomorphism $G_1 \to G_2$ of labeled hypergraphs that preserves names, namely $\eta_1 \circ h_V = \eta_2$, where h_V is the action of h on vertices.*

Interface vertices can be understood as "external" vertices, with a public, global identity. They may be interaction points, i.e., they may be shared, with other graphs. This will allow for a simple definition of parallel composition of NH-graphs. Notice that NH-graph homomorphisms must be injective on vertices, because they must commute with functions that are injective on vertices.

We say that $\eta_1 \triangleright G_1$ and $\eta_2 \triangleright G_2$ are isomorphic, written $\eta_1 \triangleright G_1 \cong \eta_2 \triangleright G_2$, whenever there is an NH-graph isomorphism (i.e., a NH-graph morphism whose underlying hypergraph homomorphism is an isomorphism) between them.

Remark 2. *A NH-graph $\eta \triangleright G$ can be seen as the following span of (total) injective graph homomorphisms*

$$\begin{tikzcd}{[\mathcal{N}]} & {[img(\eta)]} \arrow[l, hook', "\eta_l"'] \arrow[r, hook, "\eta_r"] & G\end{tikzcd}$$

where $[img(\eta)]$ is the discrete graph with vertices $img(\eta)$, $\mathcal{N}$ is the infinite discrete graph with vertices $\mathcal{N}$, $\eta_l(v) = \eta(v)$ and η_r is an embedding.

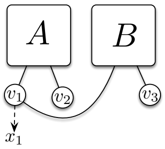
3.1 Example

Consider the optimization term shown in the introduction, without the outer restriction

$$(x_2)A(x_1,x_2) \parallel (x_3)B(x_1,x_3)$$

Recall that such a term may represent the following optimization problem: given two subproblems A and B, with cost functions parametric in x_1, x_2, and x_1, x_3, respectively, find the optimal total cost. Actually, here x_1 is free, meaning that the total cost is parametric in x_1.

The problem can be represented as a NH-graph $\eta \triangleright G$ with two hyperedges, labeled A and B, and three vertices v_1, v_2, v_3, corresponding to x_1, x_2 and x_3. Actually, only x_1 becomes an interface name, because it is the only variable the problem "exposes". Other variables are not part of the interface, meaning that they are taken up to α-conversion. The NH-graph is depicted on the right: the dashed line describes the domain of definition of η, namely $\eta(v_1) = x_1$.



3.2 An algebra for NH-graphs

Now we show that we can regard NH-graphs as elements of an algebra for the optimization specification. This will allow us to recursively evaluate (parsing of) NH-graphs as cost functions, as done in section 4.

Operations are interpreted as follows

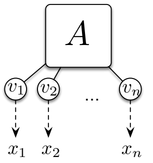
$$\eta_1 \triangleright G_1 \parallel^g \eta_2 \triangleright G_2 = \eta \triangleright (G_1 \uplus G_2)_{/\sim_V} \quad \text{where} \begin{cases} \sim_V = \left\{ (v_1, v_2) \in V_{G_1} \times V_{G_2} \,\middle|\, \begin{array}{c} \eta_1(v_1) = \eta_2(v_2) \\ \neq \text{undefined} \end{array} \right\} \\ \eta([v]_{\sim_v}) = \begin{cases} \eta_1(v) & v \in V_{G_1} \\ \eta_2(v) & v \in V_{G_2} \end{cases} \end{cases}$$

$$(x)^g(\eta \triangleright G) = \eta_{\setminus x} \triangleright G \quad \text{where} \quad \eta_{\setminus x}(v) = \begin{cases} \text{undefined} & \eta(v) = x \\ \eta(v) & \text{otherwise} \end{cases}$$

$$(\eta \triangleright G)\pi^g = (\pi \circ \eta) \triangleright G$$

The parallel composition $\eta_1 \triangleright G_1 \parallel_g \eta_2 \triangleright G_2$ is computed by taking the disjoint union of the two NH-graphs and then identifying vertices with the same interface names (formally, $/\sim_V$ takes equivalence classes of vertices). The function η is defined on equivalence classes of vertices as expected. The restriction $(x)_g\eta \triangleright G$ of $\eta \triangleright G$ w.r.t. x simply removes x from the interface of $\eta \triangleright G$.

The interpretation of constants can be defined via a mapping

$[\![A(x_1, x_2, \dots, x_n)]\!]^g =$ 

$\qquad [\![nil]\!]^g = !:\varnothing \to \mathcal{N} \triangleright 0_G$

where $!:\varnothing \to \mathcal{N}$ is the (unique) mapping from $\varnothing$ to $\mathcal{N}$ and 0_G is the empty hypergraph

Now we have to show that we have indeed defined an algebra. We have to check that congruence axioms are satisfied. We first need the following characterization of the minimal support of a NH-graph.

Lemma 1. *$supp(\eta \triangleright G) = img(\eta)$.*

Proposition 1. *Operations $\parallel^g$, $(x)^g$, π^g satisfy axioms of fig. 1, where $\equiv$ becomes $\cong$ and $fn(-)$ becomes $supp(-)$.*

Now we can define a unique evaluation of optimization terms: given p, the corresponding NH-graph $[\![p]\!]^g$ can be computed by structural recursion using the evaluation of constants given above and the interpretation of operations on NH-graphs. This induces a sound and complete axiomatization for NH-graphs. In fact, structurally equivalent optimization terms are evaluated to isomorphic NH-graphs (soundness).

For completeness, given a NH-graph $\eta \triangleright G$, we can construct an equivalent term in normal form (regarded as canonical representative of its $\equiv$-class, by remark 1) as follows. We encode each edge e of G as an atomic term with label $lab_G(e)$. The arguments of this term are the names of the interface vertices of e, and arbitrary ones for non-interface vertices. Finally, we form the parallel composition of all these terms and we restrict the names that are not in the interface. Every choice of restricted names is valid: α-conversion guarantees that all possible encodings of $\eta \triangleright G$ are in the same structural congruence class.

3.3 Tree decompositions

In graph theory, we have the well-known notion of *tree decomposition* of a graph [22], which can be understood as a way of parsing a graph. We report the definition by [16].

Definition 5 (Tree decomposition)**.** *A* tree decomposition *(TD) of a hypergraph G is a pair (T,X), where $T = (N,A)$ is a tree (i.e., an undirected, acyclic graph) and $X = \{X_n\}_{n\in N}$ is a family of subsets $X_n \subseteq V_G$, one for each node of N, such that: (a) $\bigcup_{n\in N} X_n = V_G$; (b) for all $e \in E_G$ there is $n \in N$ such that $a_G(E) \subseteq X_n$; (c) for all $v \in V_G$, the set of nodes $\{n \in N \mid v \in X_n\}$ induces a subtree of T.*

In our framework, ordinary hypergraphs (without isolated vertices) can naturally be seen as NH-graphs with empty interface, and each of their TDs can be represented as an optimization term.

Theorem 1. *Every TD for G induces an optimization term t such that $[\![t]\!]^g \cong \uparrow \triangleright G$, where $\uparrow : V_G \to \mathcal{N}$ is the nowhere defined function.*

The idea is constructing t via a visit of T from a chosen root r. We first associate the induced subgraph G_n of G to X_n, for each node $n \in N$. Each time a new node n is expanded in the visit, we generate the following subterms of t, all in parallel: one term representing edges and nodes of G_n not already in t; the subterms corresponding to n's children. Correctness of the translation is guaranteed by (a)-(c) of definition 5. Notice that, since all variables are restricted, choosing a different root amounts to rearranging restrictions. By soundness, this operation results in terms with isomorphic images via $[\![-]\!]^g$, all isomorphic to G.

4 Representing and solving optimization problems

We now show how typical optimization problems can be represented and solved in our algebraic framework.[1] Suppose we have n atomic problems $A_1,\dots,A_n$ whose variables can be assigned values in $\mathbb{D}$, and we want to minimize a function of the form

$$\sum_{1\le i\le n} c_{A_i}(\tilde{x}_i)$$

where each $c_{A_i}(\tilde{x}_i) : \mathbb{D}^{|\tilde{x}_i|} \to \mathbb{R}_\infty$, for $i = 1,\dots,n$, gives a cost to each variable assignment for the problem A_i; an infinite cost represents a forbidden assignment.

The problem can be represented as the following term in normal form

$$p = (\tilde{x})(A_1(\tilde{x}_1) \parallel \cdots \parallel A_n(\tilde{x}_n)) \qquad \text{where} \quad \tilde{x} = \tilde{x}_1 \cup \cdots \cup \tilde{x}_n$$

and the computation of the optimal cost as a function

$$[\![p]\!]^c : (\mathcal{N} \to \mathbb{D}) \to \mathbb{R}_\infty$$

giving a cost to each assignment of variables. More precisely, its computation is performed by assigning values to the free variables of p (discarding assignments to other variables), and minimizing w.r.t. bound ones. Typically we have $fn(p) = \varnothing$, so minimization is performed w.r.t. all variables.

Formally, we take an algebra for the optimization specification formed by *cost functions* $\phi : (\mathcal{N} \to \mathbb{D}) \to \mathbb{R}_\infty$, where we interpret optimization terms. For any assignment of variables $\rho : \mathcal{N} \to \mathbb{D}$, the interpretation of constants is

$$[\![A_i(x_1,\dots,x_n)]\!]^c \rho = c_{A_i}(\rho(x_1),\dots,\rho(x_n)) \qquad\qquad [\![nil]\!]^c \rho = 0$$

and complex terms are recursively interpreted as follows

$$[\![p_1 \parallel p_2]\!]^c \rho = [\![p_1]\!]^c \rho + [\![p_2]\!]^c \rho \qquad [\![(x)p]\!]^c \rho = \min_{v\in\mathbb{D}} [\![p]\!]^c (\rho[x \mapsto v]) \qquad [\![p\pi]\!]^c \rho = [\![p]\!]^c (\rho \circ \pi^{-1})$$

We have the following property, which comes from the theory of permutation algebras.

[1] A different, more efficient, setting will be described in section 5.1.

Property 1. $supp(\llbracket p \rrbracket^c) \subseteq fn(p)$.

We introduce a condition on cost functions, called *compactness*. A compact ϕ depends only on a "few" variables. This is essential to compute and store cost functions in an efficient way, as we will see later.

Property 2 (Compactness). *We say that* $\phi\colon(\mathcal{N} \to \mathbb{D}) \to \mathbb{R}_\infty$ *is* compact *if* $\rho_{|supp(\phi)} = \rho'_{|supp(\phi)}$ *implies* $\phi\rho = \phi\rho'$*, for all* $\rho,\rho'\colon\mathcal{N} \to \mathbb{D}$.

When considering a term p, by property 2, $\llbracket p \rrbracket^c$ is compact if it depends only on assignments to free variables of p. This is clearly the case for the interpretation of constants, and can be shown by structural induction for complex terms. Notice that this property is not true for the whole algebra of functions $\phi\colon(\mathcal{N} \to \mathbb{D}) \to \mathbb{R}_\infty$, but only for the subalgebra in the image of $\llbracket - \rrbracket^c$.

Canonical forms and normal forms of a term always have the same cost function. This is thanks to the following proposition, which is a direct consequence of cost functions forming an algebra of the optimization specification.

Proposition 2. *If* $p \equiv q$ *then* $\llbracket p \rrbracket^c = \llbracket q \rrbracket^c$.

4.1 Computational complexity of cost functions

Although structurally congruent terms have the same cost functions, these functions may be computed in different ways, each possibly with a different computational cost. In fact, the position of restrictions inside a term determines a strategy for variable elimination. As already mentioned, finding the best one amounts to giving a solution for the secondary optimization problem.

We introduce a notion of complexity for an optimization problem p, similar to the one of [1], estimating the cost of computing its value $\llbracket p \rrbracket^c$. This is given by the function with greatest "size" encountered while inductively constructing $\llbracket p \rrbracket^c$, the size being given by the variables in the support, that are the only ones determining the value of $\llbracket p \rrbracket^c$ (property 2).

Formally, the complexity of p, written $\langle\!\langle p \rangle\!\rangle$, is recursively defined as follows

$$\langle\!\langle A(\tilde{x}) \rangle\!\rangle = |\tilde{x}| \qquad \langle\!\langle nil \rangle\!\rangle = 0 \qquad \langle\!\langle (x)p \rangle\!\rangle = \langle\!\langle p \rangle\!\rangle \qquad \langle\!\langle p \parallel q \rangle\!\rangle = \max\{\langle\!\langle p \rangle\!\rangle, \langle\!\langle q \rangle\!\rangle, |fn(p \parallel q)|\}$$

The interesting cases are $(x)p$ and $p \parallel q$: the computation of $\llbracket (x)p \rrbracket^c$ relies on that of $\llbracket p \rrbracket^c$, whose support may be bigger, so we set the complexity of $(x)p$ to that of p; computing $\llbracket p \parallel q \rrbracket^c$ requires computing $\llbracket p \rrbracket^c$ and $\llbracket q \rrbracket^c$, but the support of the resulting function is the union of those of p and q, so we have to find the maximum value among $\langle\!\langle p \rangle\!\rangle$, $\langle\!\langle q \rangle\!\rangle$ and the overall number of free variables.

Complexity is well-defined only for hierarchical terms: applying $(\mathbf{AX}_{SE})$ to choose a different variable elimination strategy may change the complexity. Consider, for instance, the following term in normal form

$$p = (x_1)(x_2)(x_3)(A(x_1,x_2) \parallel B(x_2,x_3)) \ \ ;$$

we have $\langle\!\langle p \rangle\!\rangle = 3$, but if we take a canonical form

$$q = (x_2)((x_1)A(x_1,x_2) \parallel (x_3)B(x_2,x_3))$$

we have $\langle\!\langle q \rangle\!\rangle = 2$. Indeed, we have the following results for hierarchical terms.

Lemma 2. *Given* $(x)(p \parallel q)$*, with* $x \notin fn(q)$*, we have* $\langle\!\langle (x)p \parallel q \rangle\!\rangle \leq \langle\!\langle (x)(p \parallel q) \rangle\!\rangle$.

As an immediate consequence, all the canonical forms of a term always have lower or equal complexity than the normal form.

Theorem 2. *Given a term p, let n be its normal form. Then, for all canonical forms c of p we have* $\langle\!\langle c \rangle\!\rangle \leq \langle\!\langle n \rangle\!\rangle$.

(a) $\llbracket A(x_1,x_2)\rrbracket^c$

x_1	x_2	*cost*
d_1	d_1	7
d_1	d_2	5
d_2	d_1	∞
d_2	d_2	2

(b) $\llbracket B(x_2,x_3)\rrbracket^c$

x_2	x_3	*cost*
d_1	d_1	9
d_1	d_2	1
d_2	d_1	6
d_2	d_2	13

(c) $\llbracket (x_1)A(x_1,x_2)\rrbracket^c$

x_2	*cost*
d_1	$\min\{7,\infty\} = 7$
d_2	$\min\{5,2\} = 2$

(d) $\llbracket (x_3)B(x_2,x_3)\rrbracket^c$

x_2	*cost*
d_1	$\min\{9,1\} = 1$
d_2	$\min\{6,13\} = 6$

(e) $\llbracket (x_1)A(x_1,x_2) \parallel (x_3)B(x_2,x_3)\rrbracket^c$

x_2	*cost*
d_1	$7+1 = 8$
d_2	$2+6 = 8$

Table 1: Cost functions for the problems in the example.

4.2 Example

Consider two problems A and B, with two variables each, ranging over $\{d_1,d_2\}$. Their cost functions are shown in Tables 1a and 1b. We consider the optimization problem that consists in finding the minimal value of $A(x_1,x_2)+B(x_2,x_3)$.

As we already saw, the term in canonical form representing the problem is

$$p = (x_2)((x_1)A(x_1,x_2) \parallel (x_3)B(x_2,x_3))$$

We now show how $\llbracket p\rrbracket^c$ can be computed. We proceed in a bottom-up order, from atomic subterms to increasingly complex terms. This is close to a dynamic programming algorithm, as it allows computing and storing a (finite, thanks to the compactness property) representation of cost functions once and for all. Table 1 show such finite representations in a tabular form. We perform the following optimization steps, each corresponding to an operator of the syntax:

1. $\llbracket (x_1)A(x_1,x_2)\rrbracket^c$ and $\llbracket (x_3)B(x_2,x_3)\rrbracket^c$ are computed by minimizing $\llbracket A(x_1,x_2)\rrbracket^c$ and $\llbracket B(x_2,x_3)\rrbracket^c$ w.r.t. x_1 and x_3 respectively (Tables 1c and 1d). Notice that these functions can be computed in parallel.
2. $\llbracket (x_1)A(x_1,x_2) \parallel (x_3)B(x_2,x_3)\rrbracket^c$ is computed by evaluating $\llbracket (x_1)A(x_1,x_2)\rrbracket^c$ and $\llbracket (x_3)B(x_2,x_3)\rrbracket^c$ on the same value for x_2, and then summing up the results (Table 1e).
3. Finally, $\llbracket (x_2)((x_1)A(x_1,x_2) \parallel (x_3)B(x_2,x_3))\rrbracket^c$ is computed by minimizing the function of step 2) w.r.t. x_2.

The last step gives the overall minimal value 8. By looking at Tables in a top-down order, from 1e to 1a, each time picking those variable assignments that contributed to the cost, one can recover the corresponding optimal assignment(s) for x_1,x_2 and x_3, namely d_1,d_1,d_2 and d_2,d_2,d_1.

5 Dynamic programming on hierarchical NH-graphs

The existence of an algebra of NH-graphs allows us to recursively compute cost functions for these graphs: given a NH-graph $\eta \triangleright G$ and an optimization term such that $\llbracket p\rrbracket^g = \eta \triangleright G$, we can compute its cost function in the style of section 4.

$$(x_2)((x_1)A(x_1,x_2) \parallel (x_3)B(x_2,x_3))$$

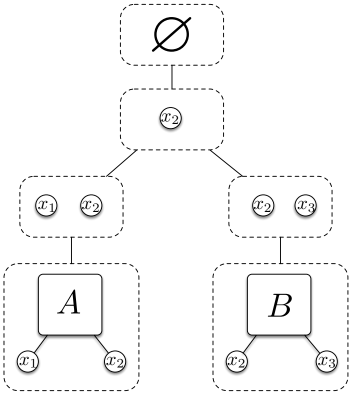


Figure 2: A hierarchical term and the corresponding hierarchical NH-graph.

However, the information about the variable elimination strategy cannot be recovered from the NH-graph itself. In order to do this, we need to introduce the graphical counterpart of hierarchical terms, which we call *hierarchical* NH-graphs. They are trees that describe the structure of a NH-graph $\eta \triangleright G$ in terms of nested components. These trees are such that:

- the root is the discrete hypergraph formed by the interface vertices of G;
- each internal node n is a discrete subgraph of G;
- leaves are hypergraphs with a single hyperedge of G;
- there is an arc from G to G' whenever $G \subset G'$.

The intuition is that each internal node n of the tree is a component of $\eta \triangleright G$ that exposes some additional vertices and includes all the components in the subtree rooted in n. Leaves are basic components, i.e., hyperedges.

The correspondence between hierarchical terms and hierarchical NH-graph graphs is exemplified in fig. 2. The scope of each restriction determines a component in the tree, where a vertex for the restricted name is added. For convenience, we used the same name for restricted variables and corresponding non-interface vertices, but the latter, as in ordinary NH-graphs, are actually up to α-conversion. A top-down visit of the tree amounts to "opening" scopes and revealing their names.

As hierarchical terms, hierarchical NH-graphs describe a solution for the secondary optimization problem. It is possible to show that hierarchical NH-graphs form an algebra for the hierarchical optimization specification. As seen in the example, each hierarchical term corresponds to a hierarchical NH-graph. We also have the opposite correspondence.

Proposition 3. *Each hierarchical NH-graph can be represented as a hierarchical term.*

Exploiting this correspondence, we can interpret hierarchical NH-graphs as evaluations of cost functions with a specific variable elimination strategy, which can be implemented via dynamic programming.

Remark 3. *In remark 2 we have characterized NH-graphs as spans of hypergraph homomorphisms. Interestingly, exploiting this characterization we can recover the NH-graph of which a given hierarchical NH-graph is the decomposition. In fact, a hierarchical NH-graph can be regarded as a span where the right part is a diagram T made of a "tree" of graph embeddings.*

$$\begin{tikzcd} {[\mathcal{N}]} & {[img(\eta)]} \arrow[l, hook', "\eta_l"'] \arrow[r, hook, "\eta_r"] & T \end{tikzcd}$$

Here η_r maps interface vertices to themselves in the root of T. Notice that the morphisms of T tell which are the interface vertices in the nodes of T. In order to "paste" together leaf hypergraphs of T, we can make a colimit of T in the category of graphs and their morphisms. The result is the disjoint union of such hypergraphs, where vertices that are images of the same interface one, along the morphisms of T, are identified.

5.1 The parking optimization problem

We now introduce the parking optimization problem and we apply our approach to it. The parking optimization problem consists in finding the best parking zone for each vehicle of an ensemble, that is a group of vehicles with similar features. This can be formalized as follows. Assume a set of parking zones $P = \{A, B, \dots\}$ and of car variables $C = \{x, y, \dots\}$, and two functions:

- $c{:}P \to \mathbb{N}$, assigning a *capacity* to every zone;
- $F{:}C \to P \to \mathbb{R}_\infty$, specifying the cost $F(x)(A)$ for x to park in A.

Given an assignment $\rho{:}C \to P$ of cars to zones, let $\rho_A = \{x \mid \rho(x) = A\}$. We want to find an assignment ρ such that $|\rho_A| \leq c(A)$, for all $A \in P$, minimizing

$$\sum_{x \in \mathcal{N}} F(x)(\rho(x))$$

The problem can be specified in the style of section 2. Here a term p represents a parking system: $A(x_1, \dots, x_n)$ means that x_i might be parked in A; $(x)p$ means that car x cannot be parked outside of p, so it must have a parking spot in one of the zones of p. In general, a term p represents a part of the system made of one or more parking zones.

In section 4 we presented an algebra of cost functions for typical optimization problems. Here we introduce another, "more efficient" algebra which, nevertheless, fits into our algebraic framework and can be used to evaluate optimization terms. To each parking system p we associate a function

$$[\![p]\!]^c : \mathcal{P}(fn(p)) \to \mathbb{R}_\infty$$

The intended meaning of $[\![p]\!]^c X$ is the cost of parking in p cars $X \subseteq fn(p)$ and all cars corresponding to variables restricted in p. Notice that the evaluation function $[\![-]\!]^c$ is quite different from that of section 4. Here assignments do not fix the values of variables, i.e., the parking zones where cars are allocated, but only their positions with respect to the present p. However, our framework is able to accommodate also this more efficient setting.

To avoid handling polymorphic functions, we automatically extend functions $[\![p]\!]^c$ to the whole $\mathcal{N}$, namely $[\![p]\!]^c{:}\mathcal{P}(\mathcal{N}) \to \mathbb{R}_\infty$, by letting

$$[\![p]\!]^c X = [\![p]\!]^c (X \cap fn(p)). \tag{1}$$

However, this function is still determined by subsets of $fn(p)$, so it admits a finite representation that can be efficiently computed and stored. Formally, properties 1 and 2 hold for extensions to $\mathcal{N}$ (regarded as functions $(\mathcal{N} \to \{0,1\}) \to \mathbb{R}_\infty$).

Cost functions can be defined by recursion on the structure of systems. As mentioned, it is enough to define their action on a subset X of their support. We have

$$[\![A(\tilde{x})]\!]^c X = \begin{cases} \sum_{x \in X} F(x)(A) & |X| \leq c(A) \\ \infty & \text{otherwise} \end{cases}$$

meaning that cars $X \subseteq \tilde{x}$ can be parked in zone A iff their number is at most the capacity of A. For *nil* we simply have $[\![nil]\!]^c X = 0$.

Then we have

$$[\![p \parallel q]\!]^c X = \min_{\{X_1,X_2\} \in \mathcal{P}_2(X)} \left\{ [\![p]\!]^c X_1 + [\![q]\!]^c X_2 \;\middle|\; \begin{array}{l} X_1 \subseteq fn(p), \\ X_2 \subseteq fn(q) \end{array} \right\}$$

where $\mathcal{P}_2(X)$ are the partitions in two sets of $X \subseteq fn(p \parallel q)$. Here, to park cars X in component $p \parallel q$, one has to park each of them in either component p or component q, but not in both. Thus the best option must be chosen. Finally, we have

$$[\![(x)p]\!]^c X = [\![p]\!]^c (X \cup \{x\}).$$

Here it is required that car x is parked in component p.

Typically, the whole system s has no free names. Thus $[\![s]\!]\varnothing$ is a real number, the total minimized cost, or ∞ if the problem has no solution.

In order to have a proper theory of cost functions, we have to show that we have indeed defined a model of the optimization specification. Let $X\pi$ be the element-wise application of a permutation $\pi{:}\mathcal{N} \to \mathcal{N}$ to $X \subseteq \mathcal{N}$. Then we have the following theorem.

Theorem 3. *Cost functions $\phi{:}\mathcal{P}(\mathcal{N}) \to \mathbb{R}_\infty$ satisfying (1) form a model of the optimization specification, together with the given interpretation of operators and the permutation action $(\phi\pi)X = \phi(X\pi^{-1})$.*

5.2 Dynamic programming algorithm

Consider the scenario with three possible parking zones A,B,C and three cars x_1,x_2 and x_3. We assume the following values for F and c.

$F(x_1)(A) = 3$	$F(x_1)(B) = \infty$	$F(x_1)(C) = \infty$	$c(A) = 2$	$c(B) = 2$	$c(C) = 2$
$F(x_2)(A) = 4$	$F(x_2)(B) = 6$	$F(x_2)(C) = \infty$			
$F(x_3)(A) = \infty$	$F(x_3)(B) = 4$	$F(x_3)(C) = 1$			

In fig. 3, on the left side, we show the term in normal form, and the corresponding NH-graph, modeling the system. We want to compute the cost function $[\![p]\!]^c$ using dynamic programming. The crucial property is compactness of cost functions: $[\![p]\!]^c$ can be represented as a table of size $|fn(p)|^2$. Although a problem is typically specified as a term in normal form, we consider its canonical form c, shown on the right side of fig. 3, because its complexity is equal or lower (theorem 2). We assume that all occurrences of *nil* have been eliminated via structural congruence.

We propose a dynamic programming algorithm that is driven by the hierarchical NH-graph that c describes, shown in the right side of fig. 3. This algorithm operates as the one in section 4.2, but tables are computed using the different interpretation of operators on cost functions, introduced in section 5.1.

The algorithm starts from the cost functions for the leaves. These are shown in Tables 2a to 2c, where the leftmost columns indicates whether a car is parked inside (✓) or outside (–) each zone. They are computed as described in section 5.1: e.g., for $[\![A(x_1,x_2)]\!]^c$, the cost for each row is $[\![A(x_1,x_2)]\!]^c X$, where X are the variables marked with ✓ in that row. Then, the algorithm performs a bottom-up visit of the tree and eliminates variables accordingly. More precisely, whenever an edge from G to G' is traversed, with G and G' discrete hypergraphs, variables $G \setminus G'$ are eliminated. In the following we show the elimination steps, also indicated in fig. 3:

[2] The actual size is $\mathcal{O}(2^{|fn(p)|})$, but we show the exponent, as $2^x \le 2^y$ iff $x \le y$.

Normal

$p = (x_1)(x_2)(x_3)(A(x_1,x_2) \parallel B(x_2,x_3) \parallel C(x_3))$

Canonical

$c = (x_2)((x_1)A(x_1,x_2) \parallel (x_3)(B(x_2,x_3) \parallel C(x_3)))$

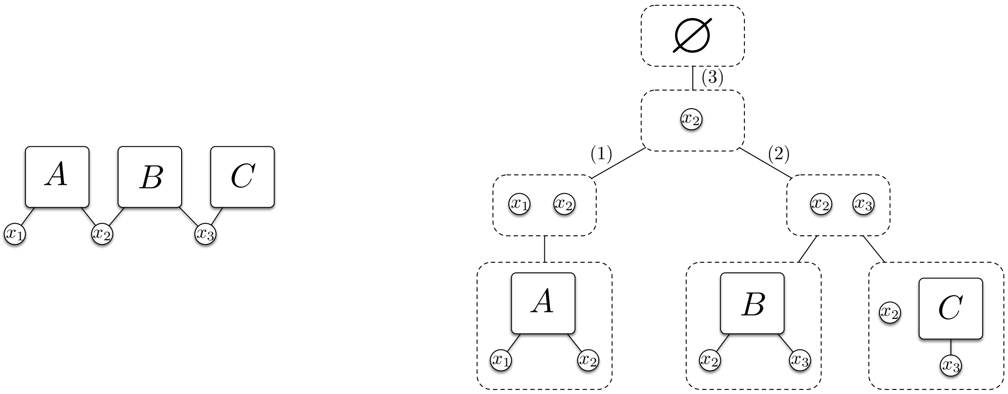


Figure 3: Graphical and corresponding term representation of a parking problem.

(1) *Elimination of* x_1: Table $[\![(x_1)A(x_1,x_2)]\!]^c$ (2d), with only one column x_2, is computed by forcing x_1 to be inside A;

(2) *Elimination of* x_3: the table $[\![(x_3)(B(x_2,x_3) \parallel C(x_3))]\!]^c$ is computed: Table 2e shows values for x_2, the partitions considered when computing the output cost, and the final cost. Notice that this and the previous step could be executed in parallel. This fact comes immediately from terms $(x_1)A(x_1,x_2)$ and $(x_3)(B(x_2,x_3) \parallel C(x_3))$ being composed in parallel in $(x_1)A(x_1,x_2) \parallel (x_3)(B(x_2,x_3) \parallel C(x_3))$.

(3) *Elimination of* x_2: finally, the Table $[\![p]\!]^c$ (2f) is computed, by comparing costs of parking x_2 inside either $(x_1)A(x_1,x_2)$ or $(x_3)(B(x_2,x_3) \parallel C(x_3))$.

(4) *Optimal variable assignment:* tracking back through the Tables we find:
- x_2 inside $(x_1)A(x_1,x_2) \parallel (x_3)(B(x_2,x_3) \parallel C(x_3))$;
- x_2 inside $(x_1)A(x_1,x_2)$;
- x_2 inside $A(x_1,x_2)$ with cost 4;
- x_1 inside $A(x_1,x_2)$ with cost 3;
- x_3 inside $B(x_2,x_3) \parallel C(x_3)$;
- x_3 inside $C(x_3)$ with cost 1.

Notice that, in general, the outcome of the algorithm may be ∞, whenever there is no car assignment to parking zones that respect capacities.

6 Conclusion

In the paper we have introduced two process algebra-like specifications for the description of optimization problems. The more abstract version (which includes the scope extension axiom) defines (hyper) graphs, where vertices are variables, and edges are tasks whose costs depend on the values of the adjacent variables. Dropping the above axiom yields a specification corresponding to different parsing trees

Table 2: Example tables. Parameters of atomic subterms are often omitted.

(a) $\llbracket A(x_1,x_2) \rrbracket^c$

x_1	x_2	*cost*
✓	✓	7
✓	–	3
–	✓	4
–	–	0

(b) $\llbracket B(x_2,x_3) \rrbracket^c$

x_2	x_3	*cost*
✓	✓	10
✓	–	4
–	✓	6
–	–	0

(c) $\llbracket C(x_3) \rrbracket^c$

x_3	*cost*
✓	1
–	0

(d) $\llbracket (x_1)A(x_1,x_2) \rrbracket^c$

x_2	*cost*
✓	7
–	3

(e) $\llbracket (x_3)(\, B(x_2,x_3) \parallel C(x_3) \,) \rrbracket^c$

x_2	$\llbracket B \rrbracket^c$		$\llbracket C \rrbracket^c$	$\llbracket B \rrbracket^c + \llbracket C \rrbracket^c$	*cost*
	x_3	x_2	x_3		
✓	✓	✓	–	10	7
	–	✓	✓	7	
–	✓	–	–	4	1
	–	–	✓	1	

(f) $\llbracket (x_2)(\; (x_1)A(x_1,x_2) \parallel (x_3)(\, B(x_2,x_3) \parallel C(x_3) \,) \;) \rrbracket^c$

$\llbracket (x_1)A \rrbracket^c$	$\llbracket (x_3)(B \parallel C) \rrbracket^c$	$\llbracket (x_1)A \rrbracket^c + \llbracket (x_3)(B \parallel C) \rrbracket^c$	*cost*
x_2	x_2		
✓	–	8	8
–	✓	10	

of the given graph. Choosing a particular tree corresponds to selecting a dynamic programming strategy for the given problem, whose execution can be carried on via a bottom up visit of the tree. We apply our approach to the parking optimization problem developed, in collaboration with Volkswagen, in the ASCENS e-mobility case study.

The idea of exploiting graphs to decompose and solve various kinds of problems is not new. In [11] graphs are represented as elements of an algebra and monadic second-order properties are evaluated on them. In [7] dynamic programming algorithms are derived from (*nice*) tree decompositions of graphs. In [6] tree decompositions are represented as in a category of spans and cospans, and then as terms of an algebraic specification. Our approach has the following advantages w.r.t. the cited ones:

- Our algebraic specification is simpler, but nonetheless expressive. In fact, variable elimination strategies can be represented via restrictions. Moreover, we have a graphical representation of such strategies as hierarchical NH-graphs, which can be regarded as very simple tree decompositions.
- Our algebras are permutation algebras, which provide: (a) a state-of-the-art treatment of α-conversion and of freshness requirements; (b) a uniform and general definition of domain given by the notion of support. Operations are defined on the whole set of names, so they are independent of the actual interface (support), unlike [6]. Moreover, the notion of support automatically defines the sizes of the tables employed in the dynamic programming implementation.

The following lines of research are also related to our work. Bistarelli, Montanari and Rossi deal with SCSP [2] and its combination with logic programming [3, 4] and concurrency [5]. They give an interpretation of constraints over certain semirings, e.g., the tropical semiring, as we do here. However, operations on constraints are defined point-wise using the semiring operations: the approach is too restrictive, e.g., it does not easily accommodate the case study shown in this paper. A direct connection between (logical) CSP and dynamic programming is shown in [20]. Dechter in [13] introduces bucket elimination as a general solution technique for a variety of problems: it consists in a strategy of problem reduction employing a convenient elimination ordering of variables and constraints. The associated technique of conditioning search allows for approximated versions of the bucket elimination approach. Kohlas and Pouly in [17] suggest *valuation algebras* as a foundation for a general view of information processing. They define axioms for valuation algebras, consider a number of instantiations and provide generic inference algorithms for their processing. Our approach is similar, but more direct, being based on a simple process algebra specification and on a bottom up visit of a tree of graphs satisfying the specification. In [10] distributed systems are represented as *CHARMs* (Concurrency and Hiding in an Abstract Rewriting Machine), that are hypergraphs with a global and a local part. They form an algebra including edges and vertices restriction. Our algebras and NH-graphs are similar, but we do not need edges restriction.

Dynamic programming evaluations are particularly interesting for autonomic systems, as studied by the ASCENS project, where the actual behavior often consists, typically for the dynamic programming case, of propagating local knowledge to obtain global knowledge and getting it back for local decisions. When dealing with global problems, however, the complexity of the dynamic programming algorithms can grow exponentially even for graphs of limited complexity. Consider a rectangular grid of size n, with vertices labeled by variables, and edges by cost functions with two arguments. It is shown in [18] that its complexity is exponential in n. There are efficient algorithms for finding the optimal elimination order of vertices in a graph, but they deal with specific cases (e.g., Gaussian elimination [12, 23]). Thus approximation techniques are quite relevant, in particular when a good global solution, possibly not optimal, is still acceptable.

Several heuristic techniques can be experimented. For instance, for the parking problem we could restrict the number of possible zones for each car, taking the best k of them. Then if the optimal solution would include a choice worse than k for some car, the solution found, if any, would not be optimal. However, at least no client would be treated too badly. Another, quite general, approximation technique would be to artificially reduce the dimensions of tables by decomposing high dimensional ones into the sums of a few lower dimensional tables. The latter can be computed minimizing the mean square error [19]. The storage reduction can be propagated in such a way to reduce substantially the overall complexity.

An interesting piece of future work would be to extend our approach to graphs which are incrementally modified, e.g., extended, at run time. The resulting scenario could consist of a (soft) (concurrent) constraint component together with a mobile pi-calculus-like process algebra component. A good example of this combination is cc-pi [9]. Other aspects should be investigated in a precise way: the correspondence between classes of terms (normal,canonical) and NH-graphs; the nominal structure of NH-graphs.

References

[1] Umberto Bertelè & Francesco Brioschi (1973): *On Non-serial Dynamic Programming*. *J. Comb. Theory, Ser. A* 14(2), pp. 137–148, doi:10.1016/0097-3165(73)90016-2.

[2] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (1997): *Semiring-based constraint satisfaction and optimization*. *J. ACM* 44(2), pp. 201–236, doi:10.1145/256303.256306.

[3] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (2001): *Semiring-based contstraint logic programming: syntax and semantics*. ACM Trans. Program. Lang. Syst. 23(1), pp. 1–29, doi:10.1145/383721.383725.

[4] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (2002): *Soft Constraint Logic Programming and Generalized Shortest Path Problems*. J. Heuristics 8(1), pp. 25–41, doi:10.1023/A:1013609600697.

[5] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (2006): *Soft concurrent constraint programming*. ACM Trans. Comput. Log. 7(3), pp. 563–589, doi:10.1145/1149114.1149118.

[6] Christoph Blume, H. J. Sander Bruggink, Martin Friedrich & Barbara König (2013): *Treewidth, pathwidth and cospan decompositions with applications to graph-accepting tree automata*. J. Vis. Lang. Comput. 24(3), pp. 192–206, doi:10.1016/j.jvlc.2012.10.002.

[7] Hans L. Bodlaender & Arie M. C. A. Koster (2008): *Combinatorial Optimization on Graphs of Bounded Treewidth*. Comput. J. 51(3), pp. 255–269, doi:10.1093/comjnl/bxm037.

[8] Tomás Bures, Rocco De Nicola, Ilias Gerostathopoulos, Nicklas Hoch, Michal Kit, Nora Koch, Giacoma Valentina Monreale, Ugo Montanari, Rosario Pugliese, Nikola B. Serbedzija, Martin Wirsing & Franco Zambonelli (2013): *A Life Cycle for the Development of Autonomic Systems: The E-mobility Showcase*. In: SASOW, pp. 71–76, doi:10.1109/SASOW.2013.23.

[9] Maria Grazia Buscemi & Ugo Montanari (2007): *CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements*. In: ESOP, pp. 18–32, doi:10.1007/978-3-540-71316-6_3.

[10] Andrea Corradini, Ugo Montanari & Francesca Rossi (1994): *An Abstract Machine for Concurrent Modular Systems: CHARM*. Theor. Comput. Sci. 122(1&2), pp. 165–200, doi:10.1016/0304-3975(94)90206-2.

[11] Bruno Courcelle & Mohamed Mosbah (1993): *Monadic Second-Order Evaluations on Tree-Decomposable Graphs*. Theor. Comput. Sci. 109(1&2), pp. 49–82, doi:10.1016/0304-3975(93)90064-Z.

[12] Elias Dahlhaus (2002): *Minimal elimination ordering for graphs of bounded degree*. Discrete Applied Mathematics 116(1-2), pp. 127–143, doi:10.1016/S0166-218X(00)00331-0.

[13] Rina Dechter (1999): *Bucket Elimination: A Unifying Framework for Reasoning*. Artif. Intell. 113(1-2), pp. 41–85, doi:10.1016/S0004-3702(99)00059-4.

[14] Fabio Gadducci, Marino Miculan & Ugo Montanari (2006): *About permutation algebras, (pre)sheaves and named sets*. Higher-Order and Symbolic Computation 19(2-3), pp. 283–304, doi:10.1007/s10990-006-8749-3.

[15] Nicklas Hoch, Kevin Zemmer, Bernd Werther & Roland Siegwart (2012): *Electric vehicle travel optimization-customer satisfaction despite resource constraints*. In: IEEE IVS, pp. 172–177, doi:10.1109/IVS.2012.6232240.

[16] Ton Kloks (1994): *Treewidth, Computations and Approximations*. Lecture Notes in Computer Science 842, Springer, doi:10.1007/BFb0045375.

[17] Jürg Kohlas & Marc Pouly (2011): *Generic Inference: A Unifying Theory for Automated Reasoning*. John Wiley & Sons, Inc., doi:10.1002/9781118010877.ch2.

[18] Alberto Martelli & Ugo Montanari (1972): *Nonserial Dynamic Programming: On the Optimal Strategy of Variable Elimination for the Rectangular Lattice*. J. Math. Anal. Appl. 40, pp. 226–242, doi:10.1016/0022-247X(72)90046-7.

[19] Ugo Montanari (1971): *On the Optimal Approximation of Discrete Functions with Low-dimensional Tables*. In: IFIP Congress (2), pp. 1363–1368.

[20] Ugo Montanari & Francesca Rossi (1991): *Constraint Relaxation may be Perfect*. Artif. Intell. 48(2), pp. 143–170, doi:10.1016/0004-3702(91)90059-S.

[21] A. M. Pitts (2013): *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science 57, Cambridge University Press, doi:10.1017/CBO9781139084673.

[22] Neil Robertson & Paul D. Seymour (1984): *Graph minors. III. Planar tree-width*. J. Comb. Theory, Ser. B 36(1), pp. 49–64, doi:10.1016/0095-8956(84)90013-3.

[23] Mihalis Yannakakis (1981): *Computing the Minimum Fill-In is NP-Complete*. *SIAM Journal on Algebraic Discrete Methods* 2(1), pp. 77–79, doi:10.1137/0602010.