Università di Pisa
Dipartimento di Informatica

Technical Report : TR-99-02

Directed Hypergraphs as a Modelling Paradigm

Giorgio Gallo, Maria Grazia Scutella'

February 18, 1999

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

Directed Hypergraphs as a Modelling Paradigm

Giorgio Gallo and Maria Grazia Scutellà
Dipartimento di Informatica, Università di Pisa,
Corso Italia, 40, 56125 Pisa, Italy
gallo@di.unipi.it, scut@di.unipi.it

January, 1999

Abstract

We address a generalization of graphs, the directed hypergraphs, and show that they are a powerful tool in modelling and solving several relevant problems in many application areas. Such application areas include Linear Production Problems, Flexible Manufacturing Systems, Propositional Logic, Relational Databases, and Public Transportation Systems.

1 Introduction

Graph theory has proved to be a powerful conceptual framework for the analysis, modelling and solution of combinatorial and discrete optimization problems arising in many different areas such as Mathematics, Operations Research, Computer Science and Engineering.

Undirected hypergraphs ([3], [4], [5]), as a generalization of undirected graphs, have further widened the scope of applicability of graph theory, although they have probably proved to be more useful as modelling tools rather than as algorithmic ones.

Still many problems can be found in different application areas, which, although featuring an underlying graph structure, cannot be represented as graph problems without losing some of their peculiar characteristics. This fact has led to the introduction of new special types of graphs such as, for instance, the so-called " labelled graphs" and the "And-Or graphs" (see, for example, [36], [32], [28], [6], [10], [30], [37], [20], [43], [8], [34], [35], [17]).

Here we want to address a further generalization of graphs, the directed hypergraphs, which include as particular cases both labelled graphs and And-Or graphs. We shall show that they are a powerful tool not only in modelling but also in solving several relevant problems in many application areas. Such application areas include Linear Production Problems, Flexible Manufacturing Systems, Propositional Logic, Relational Databases, and Public Transportation Systems.

The plan of the paper is the following. In Section 2, some problems that have motivated the introduction of directed hypergraphs are presented. Then, in Section 3, the main concepts for a theory of directed hypergraphs which encompasses both structural and algorithmic aspects are reviewed. Finally, in Section 4, a number of interesting applications of the hypergraph paradigm are fully described.

2 Representing Many-to-One relations

The main motivation behind the introduction of directed hypergraphs is the need, in many applications, to represent Many-to-One relations, for which standard graphs are hardly suited. In fact a graph, whose arcs are of the type (x, y), with x and y single nodes, needs an added structure or some additional informations to represent Many-to-One relations (typical examples are the "And-Or graphs" and the " labelled graphs"). In directed hypergraphs instead, arcs are of the type (X, y), where X is a subset of nodes, so they can be naturally used to represent Many-to-One relations. In some cases there is the need to represent One-to-Many relations, and hyperarcs of the type (x, Y), where x is a singleton and Y is a subset of nodes, are used.

In the following we shall mention some of the settings in which the use of directed hypergraphs is justified. Throughout the paper, for short, we shall call directed hypergraphs simply hypergraphs.

2.1 Formal languages

A typical production rule of the grammar of a *context free language* is

$sentence \rightarrow nounphrase\ verbphrase.$

Its meaning is that a sentence is obtained by putting together (concatenating) in the order a noun phrase and a verb phrase. The hyperarc $(\{nounphrase, verbphrase\}, sentence)$ is a representation of this production rule when the language's terms are considered as nodes of a hypergraph. With this representation, a grammar, i.e. a set of production rules over a set of terms (either variables or constants) becomes a hypergraph, where the terms are the nodes and the production rules are the hyperarcs. It is worth mentioning that it is in this setting that the first Shortest Hyperpath algorithm has been introduced [27].

A clause in a *propositional Horn formula* is of the form:

$p_1 p_2 \dots p_r \rightarrow p_0,$

meaning that the truth of all the propositions from p_1 to p_r implies the truth of proposition p_0. This clause can be represented as the hyperarc $(\{p_1, p_2, \dots, p_r\}, p_0)$ so that, like for the context free languages, any Horn formula can be written as a hypergraph, where the propositions are the nodes and the clauses are the hyperarcs.

2.2 Production and Manufacturing Systems

In a *simple linear production system* there is a set of goods, some produced within the system and others coming from outside, and relations connecting each produced good to the set of goods needed for its production. A hyperarc, (X, y), may represent the fact that the goods in the set X are needed to produce y. Different hyperarcs with the same head y can be used to represent different technologies which, with different inputs, may be used to produce the same type of good.

Linear production systems can be represented by means of *Leontief matrices*, and actually the analysis of Leontief systems and flows is one of the areas in which hypergraphs have proven to be particularly useful, both from a modelling and from an algorithmic point of view [26].

Similar problems, although at a lower scale, arise in manufacturing. An *assembly system* is one in which some items (parts) have to be assembled in order to produce a given product. An assembly operation, which has two or more parts in input and an item (either the final product or one of its subcomponents) as the output, can be naturally represented as a hyperarc. We shall see that the problem of finding an assembly plan (i.e. a sequence of feasible assembly operations leading to the desired final product), which is optimal with respect to a given criterion, can be modelled and solved by means of hypergraph techniques.

Another interesting problem arising in manufacturing is the *cutting stock problem.* Again, hypergraphs have proven to be useful in generating "cutting patterns". As it will be shown in Section 4, the generation of optimal cutting patterns bears a strong resemblance with the generation of an optimal assembly plan.

2.3 Transportation Models

Mass Transit Systems can be quite naturally modelled by means of graphs. The *line stops* (bus stops, subway stations, ...) and the portions of a line between two consecutive stops are modelled as nodes and as arcs, respectively. Other nodes are included in the model to represent the origins and the destinations of the passengers' traffic (typically these nodes are the centroids of the zones in which the urban area is partitioned), while arcs connecting such nodes to the ones representing the line stops are used to model the walks to and from the line stops. A model of this type would be adequate if each passenger had one clearly defined best choice for the trip to his/her destination. In this case the trip of a passenger from his/her origin to the bus stop would be modelled by the arc from the origin node to the node representing the chosen line at that stop. But this is not what happens in practice. In fact, often more lines, at an almost equivalent travelling cost, are available to reach a given destination, and the passenger goes to the stop and embarks the bus which arrives first. In this case, a hyperarc of the type (x, Y) would represent in a suitable way the trip.

2.4 Relational Databases

A *Relational Database* can be thought of as a table (or possibly a collection of tables), with a column for each attribute and a row for each record or tuple. Let U be the set of all the attributes; each attribute $a \in U$ has a domain, $dom(a)$, which is the set of all possible values it can take. A subset of U, $R = \{a_1, a_2, \ldots, a_q\}$, is called a *relation scheme*, while a relation over R, $r(R)$, is a subset of the Cartesian product $dom(a_1) \times dom(a_2) \times \ldots \times dom(a_q)$. If μ is an element of the relation $r(R)$, we will denote by $\mu(X)$, for any $X \subseteq R$, its projection on the subspace of the attributes in X.

Given a relation scheme, R, a *functional dependency* (*fd*) is a constraint imposed on R, which takes the form $X \rightarrow y$, where X is a subset of R, and y is one of its elements.

A relation over R is said to satisfy the functional dependency $X \rightarrow y$ if, for any two elements μ and σ of the relation, the following implication holds true:

$$\mu(X) = \sigma(X) \Rightarrow \mu(y) = \sigma(y),$$

i.e. there is a unique set of values which the attribute y can take for any set of values taken by the attributes in X.

The set of the functional dependencies over a relation $r(R)$ can be represented by means of a directed hypergraph. We shall show that computations on such a hypergraph may be used either for checking the Database integrity or, under some additional conditions, for determining minimal keys to access the data [29].

3 Hypergraphs and related concepts

3.1 Basic concepts

A directed *hypergraph*, or simply hypergraph, is a pair $H = (V, E)$, where $V = \{1, 2, \ldots, n\}$ is the set of the nodes and E is the set of the hyperarcs, of cardinality m. A *hyperarc* is a pair,

$e = (T_e, h_e)$, with $T_e \subseteq V$ and $h_e \in V \setminus T_e$; T_e is the *tail* of e, while h_e is its *head*. Tailless hyperarcs, i.e. of type $e = (\emptyset, h_e)$, and headless hyperarcs, i.e. of type $e = (T_e, \emptyset)$, are admitted.

While the size of a standard graph is uniquely defined by n and m, the size of a hypergraph depends also on the cardinality of its hyperarcs; so, the *size* of H can be defined as the sum of the cardinalities of its hyperarcs:

$size(H) = \sum_{e \in E} |T_e|$.

A *path* P_{st} from s to t in H is a sequence $P_{st} = (v_1 = s, e_1, v_2, e_2, \ldots, e_q, v_{q+1} = t)$, where $s \in T_{e_1}$, $h_{e_q} = t$ and $h_{e_i} \in T_{e_{i+1}}$, $i = 1, 2, \ldots, q-1$.

If $s = t$, then P_{st} is a *directed cycle*. When no directed cycle exists, H is said to be cycle free or *acyclic*.

A *directed hyperpath*, or simply hyperpath, P_{Rt} from the root set R ($\subseteq V$) to the sink t ($\in V$) in H is a minimal acyclic sub-hypergraph of H containing both the nodes in R and node t, such that each node, with the exception of the nodes in R, has exactly one entering hyperarc.

The definition of hyperpath can be extended as follows. A (directed) *hypertree* T_R rooted at R in H is an acyclic sub-hypergraph of H containing the nodes in R, such that each node, with the exception of the nodes in R, has exactly one entering hyperarc.

The set R is called the *root* set, while the remaining nodes are called the *non-roots*. Any non-root v not contained in the tail of any hyperarc of T_R is said to be a *leaf* of the hypertree. By the definition, for any non-root v there is a unique directed hyperpath in T_R from R to v.

An *undirected hyperpath* (*hypertree*) is a " permutation" of a hyperpath P_{Rt} (hypertree T_R), i.e. it is obtained by a permutation of some of the hyperarcs of P_{Rt} (T_R), where the permutation of a hyperarc e is a hyperarc e' such that $T'_e \cup h'_e = T_e \cup h_e$.

As an example, figure 1 shows a hyperpath from $R = \{1, 2\}$ to $t = 6$ (on the top), and one of its permutations.

Given a set of nodes, S, and a node t, referred to as the source set and the sink, respectively, a *St-cut* $C_{St} = (V_S, V_t)$ in H is a partition of V into two subsets V_S and V_t such that $S \subseteq V_s$ and $t \in V_t$. Given a St-cut C_{St}, its *cutset*, E_{St}, is the set of all the hyperarcs e such that $T_e \subseteq V_S$ and $h_e \in V_t$. The *cardinality* of a St-cut is the cardinality of its cutset.

3.2 Connectivity and minimum cost hyperpath computation

Let us introduce the relation *connected to*, which will be denoted by the symbol $\succ$: given a subset R of nodes and a node y, we shall say that R is *connected to* y, and write $R \succ y$, if and only if a directed hyperpath exists in H from the set R to the node y.

It is easy to check that relation $\succ$ satisfies the following set of *connectivity axioms*:

1. $y \in R \subseteq V \Rightarrow R \succ y$ (reflexivity)
2. $R \succ y$ and $Z \subseteq V \Rightarrow (R \cup Z) \succ y$ (augmentation)
3. $R \succ y, \forall y \in Y$, and $Y \succ z \Rightarrow R \succ z$ (transitivity)

Starting from the set of facts $R \succ y, \forall (R, y) \in E$, and making use of the connectivity axioms, in $O(size(H))$ time it is possible to find all (and only those) pairs (R, y), with $(R \cup \{y\}) \subseteq V$, for

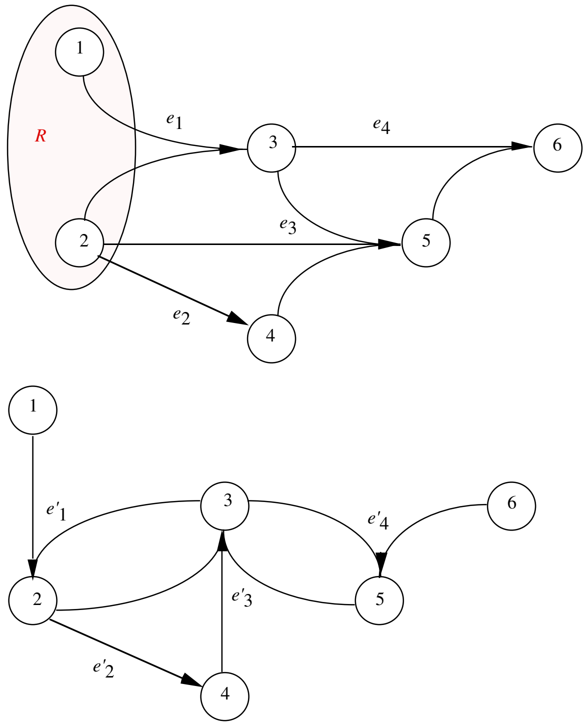

Figure 1: A hyperpath and one of its permutations

which $R \succ y$ (*connectivity procedure*). It is easy to see that R is not connected to y, i.e. $R \not\succ y$, if and only if a Ry-cut of cardinality 0 exists in H [14].

Assume that each hyperarc e in H is assigned a real cost $c(e)$. Given a hyperpath P_{Rt} from R to t, by *cost function* we mean a node function C which assigns a cost to each node in P_{Rt} depending on the costs of its hyperarcs. $C(t)$ is the cost of P_{Rt} under the chosen cost function.

Additive cost functions are those for which the node costs satisfy the following recursive equations:

$$\begin{cases} C(v) = 0, & \text{if } v \in R; \\ C(v) = c(e) + F(\{C(w) : w \in T_e\}), & \text{if } v = h_e \text{and } e \in P_{Rt}, \end{cases} \tag{1}$$

where F is a nondecreasing real set function.

Two particularly interesting cases of additive cost functions, which will be used in the following sections, are the *distance*, $Dist(v)$, and the *value*, $Val(v)$.

The former is obtained by setting:

$$F = \max\{Dist(w) : w \in T_e\}, \tag{2}$$

while for the latter it is:

$$F = \sum_{w \in T_e} a(w,e) Val(w), \tag{3}$$

where, for each e and for each $w \in T_e$, $a(w,e)$ is a nonnegative real coefficient.

The problem of finding minimum cost hyperpaths from a given root set R to all the nodes t for which a connection from R to t exists is a natural generalization of the shortest path tree problem for standard graphs.

At least in general, this problem is NP-hard. In fact, Italiano and Nanni [24] proved that the particular problem of finding minimum cardinality hyperpaths is NP-hard.

However, in the case of additive cost functions, efficient polynomial time algorithms exist. In such a case, the problem is equivalent to the problem of finding a solution to the following Generalized Bellman's Equations, where by $BS(v)=\{e \in E : v = h_e\}$ we denote the *Backward Star* of v:

$$\begin{cases} C(v) = 0, & \text{if } v \in R; \\ C(v) = \min\{c(e) + F(\{C(w) : w \in T_e\}) : e \in BS(v)\}, & \text{otherwise.} \end{cases} \tag{4}$$

Assume that $c(e)$ is non-negative, for each $e \in E$; assume also that each directed cycle of H is non-decreasing, where a directed cycle $(v_1 = s, e_1, v_2, e_2, \ldots, e_q, v_{q+1} = s)$ is said to be *non-decreasing* if, for any real z, it is:

$$c(e_q) + F_C^{v_q}(c(e_{q-1}) + F_C^{v_{q-1}}(\ldots + F_C^{v_2}(c(e_1) + F_C^{v_1}(z))\ldots)) \geq z, \tag{5}$$

with $F_C^{v_i}(z)$ denoting the restriction of $F_C(T_{e_i})$ to the case in which all the nodes of T_{e_i} have cost zero except node v_i, which has cost z, for each hyperarc e_i.

Under these additional conditions, which ensure that no node cost can be decreased through a directed cycle, and thus play the same role as the non-negative cycle condition in standard graphs, several polynomial time algorithms have been devised for the solution of the Generalized Bellman's Equations (3). More precisely, algorithms have been proposed which run in $O(n \times size(H))$ time, in the general case, and in $O(max\{n^2, size(H)\})$ time or in $O(max\{mlogn, size(H)\})$ time, if $c(e) + F(\{C(w) : w \in T_e\}) \geq C(w)$, for each $w \in T_e$ and for each $e \in E$ (Dijkstra's like condition) [14]. Observe that this last condition holds for additive cost functions such that $F(\{C(w) : w \in T_e\}) \geq Max\{C(w) : w \in T_e\}$, which is the case considered by Knuth in [27].

In the simple case in which the hypergraph is acyclic, the minimum cost hyperpaths rooted at R can be computed in $O(size(H))$ time by visiting the nodes according to a "topological order".

In the case of hypergraphs whose hyperarcs are of type (x, Y), where x is a singleton and Y is a subset of nodes, the results just reviewed about connectivity and minimum cost hyperpath computations still hold true. It is enough to reverse the directions of all the hyperarcs, thus obtaining a new hypergraph in which all the hyperarcs are of type (X, y).

3.3 Minimum cost hyperflows

Assume now that a positive real *multiplier* $\mu_v(e)$ is associated with each $v \in T_e$, $\forall e \in E$, and that a real demand vector b is associated with V. A flow on H (*hyperflow*) is a function $f : E \rightarrow \Re$ which satisfies the following *hyperflow conservation* constraints:

$$\sum_{v=h_e} f(e) - \sum_{v \in T_e} \mu_v(e) f(e) = b(v), \forall v \in V (conservation).$$

The hyperflow is said to be *feasible* if it satisfies the following *capacity* constraints, where u is an upper capacity vector associated with E:

$$0 \leq f(e) \leq u(e), \forall e \in E (capacity).$$

The *minimum cost hyperflow problem* consists in finding a feasible hyperflow on H which minimizes the function $\sum_{e \in E} c(e)f(e)$, where, as before, $c(e)$ denotes the cost associated with e. It is a quite general problem, which includes all the bounded linear programming problems. In particular, it generalizes the minimum cost flow problem on standard and on " generalized " graphs, and it extends the uncapacitated minimum cost flow problem on hypergraphs studied by Jeroslow et al. [26].

The *maximum hyperflow problem*, which consists in maximizing the total amount of hyperflow pushed into a certain sink node, given a source node, can be formulated in a similar way.

Let A denote the $n \times m$ incidence matrix of H; then the minimum cost hyperflow problem can equivalently be stated in the following matrix formulation:

$$\begin{aligned} \min \quad & \sum_{e \in E} c(e)f(e) \\ & Af = b; \\ & 0 \leq f(e) \leq u(e), \forall e \in E. \end{aligned}$$

If $\pi(v)$ denotes the dual variable or *potential* which is associated with node v, and $\lambda(e)$ is the dual variable associated with hyperarc e, then the dual problem, named the *potential problem on hypergraphs*, can be formulated as follows:

$$\begin{aligned} \max \quad & \sum_{v \in V} b(v)\pi(v) - \sum_{e \in E} u(e)\lambda(e) \\ & \pi(h_e) - \sum_{v \in T_e} \pi(v)\mu_v(e) - \lambda(e) \leq c(e), \forall e \in E; \\ & \lambda(e) \geq 0, \forall e \in E. \end{aligned}$$

In order to solve the problem, a *Hypergraph Simplex Algorithm* has been proposed in [38]. This algorithm relies on the characterization of the basis matrices of the above Linear Programming formulation in terms of spanning hypertree.

A *spanning hypertree* of H is an undirected hypertree, $T_R = (V, E_T)$, such that:

1. $E_T \subseteq E$;
2. $(T_e \cup \{h_e\}) \not\subseteq R$, $\forall e \in (E \setminus E_T)$.

In the spanning hypertree definition, property (2) is to impose a kind of minimality of the root set R; in fact, if a hyperarc e exists such that $(T_e \cup \{h_e\}) \subseteq R$, we can use it to span one of the nodes of R, and decrease by one the cardinality of the root set.

Examples of spanning hypertrees are shown in figure 2, where, with a little abuse of notation, a hypertree is represented by means of one of its top-down traversals, i.e. a traversal from the roots toward the leaves. Note that, for a given hypergraph with n nodes, it is $|R| + |E_T| = n$ for all spanning hypertrees, whereas the cardinalities of R and E_T may vary from hypertree to hypertree.

In the minimum cost hyperflow problem, each basis A_B corresponds to a pair (T_R, E_X) characterized as follows:

T_R is a spanning hypertree of the sub-hypergraph, say H^*, corresponding to the columns of A_B;

$T_R = (\{5,6\}, e_{10}, 4, e_4, 2, e_3, 1, e_6, 3)$ $\quad$ $T_{R'} = (\{1\}, e_2, 3, e_6, 4, e_3, 2, e_4, 5, e_{10}, 6)$

Figure 2: Spanning hypertrees

E_X is the set of the *external hyperarcs*, i.e. the basic hyperarcs out of the spanning hypertree; its cardinality is the same as the cardinality of the root node set R.

The correspondence between bases and pairs (T_R, E_X) is not a one-to-one correspondence. In fact, the above structure corresponds to a basis if and only if a particular matrix A_R, called the *root matrix*, whose size is the same as the size of the root node set (and of the external hyperarc set), is non-singular. An interesting property of A_R is that both the rows and the columns of A_R and of its inverse can be interpreted in terms of hyperflows and of potentials. The bases of the minimum cost hyperflow problem can thus be completely characterized in terms of hypergraph concepts.

This basis characterization generalizes the one for "standard " and for "generalized " graphs, where bases correspond to spanning trees with empty or with a singleton root node set (i.e., A_R has size 0 or 1, respectively).

By exploiting the basis characterization in terms of spanning hypertrees, the Hypergraph Simplex Algorithm solves linear systems of type $A_B f = b^*$ and $\pi A_B = c^*$, where A_B is an $n \times n$ basis and b^* and c^* are n-vectors, directly on the hypergraph.

This is obtained by performing labelling procedures on the basis spanning hypertree, and by properly using the inverse of the root matrix and its hypergraph characterization. Also the determination of the leaving hyperarc and the updating of the basis hypergraph structure and of the inverse of the root matrix are performed directly on the hypergraph at each step. So, like for the standard Network simplex algorithm, most of the computations performed at each pivot operation have direct hypergraph interpretations [38].

An implementation of the Hypergraph Simplex Algorithm is described in [13], where the results of a large experimentation are presented. In this experimentation, the primal version of a state-of-the-art LP code, such as CPLEX (Version 2.1), has been used as a benchmark. In figure 3, the running times of the Hypergraph Simplex Algorithm (HySimpleX) and of CPLEX are given as a function of the hypergraph parameters. Such a figure shows clearly that the Hypergraph Simplex Algorithm becomes increasingly faster than CPLEX as the number of the hyperarcs, m, increases, for given n (number of nodes) and k (maximum cardinality of the tails).

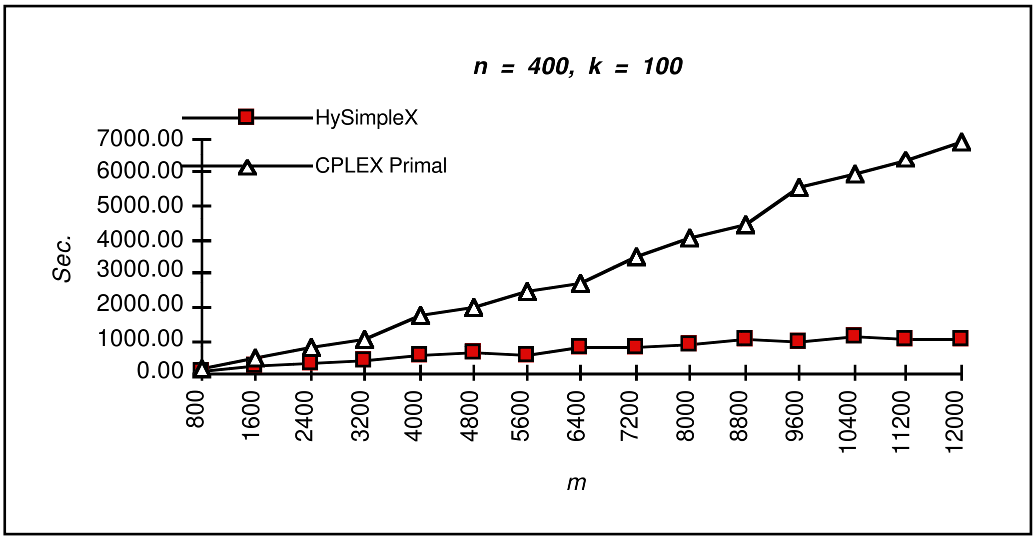

Figure 3: The running times of Hypergraph Simplex and of CPLEX

4 Hypergraph applications

4.1 Formal languages

4.1.1 Context free languages

The problem A *context free grammar* $G = (V, T, P)$ is a triple where V is a set of symbols of variables, T a set of terminal terms, and P a set of productions of the type:

$$y \to x_1 x_2 \dots x_k,$$

where y is a variable symbol, and $x_1 x_2 \dots x_k$ is a string of symbols which can be either terminal terms or variable symbols. Usually there is a distinguished variable symbol, the start symbol, which represents the type of strings which the grammar is supposed to generate. Consider, as an example, the grammar defined as follows [22]:

$$V = \{sentence, nounphrase, verbphrase, adjective, verb, adverb, noun\}$$

$$T = \{the, little, boy, ran, quickly\}$$

$$\begin{aligned} P = \{&sentence \to nounphrase\ verbphrase, \\ &nounphrase \to adjective\ nounphrase, \\ &nounphrase \to adjective\ noun, \\ &verbphrase \to verb\ adverb, \\ &adjective \to the, \\ &adjective \to little, \\ &noun \to boy, \\ &verb \to ran, \\ &adverb \to quickly\}. \end{aligned}$$

The meaning of the first production is that a sentence is obtained by putting together (concatenating) in the order a noun phrase and a verb phrase.

A typical problem which is of interest in this context is the problem of checking whether a given string of terminal terms is a *sentence*, i.e. it can be legally assigned to the variable sentence (similarly for any other variable); another one is to generate all the sentences which are "grammatically correct".

Knuth [27] has introduced the concept of *superior context free grammar* (*scfg*). A *scfg* is one in which the productions are of the type:

$$y \to g(x_1, x_2, \ldots, x_k),$$

where $y, x_1, x_2, \ldots, x_k$ are variable symbols, parentheses and commas are terminal terms, and g is a terminal symbol corresponding to a real "superior function". When $k = 0$, the function is a constant.

A function $g(x_1, x_2, \ldots, x_k)$ from $\Re_+^k$ to $\Re_+$ is said to be a *superior function* if it is monotone nondecreasing in each variable, and if $g(x_1, x_2, \ldots, x_k) \geq max\{x_1, x_2, \ldots, x_k\}$, for all $x_1, x_2, \ldots, x_k$.

An example of *scfg* is given next:

$$V = \{A, B, C\}$$

$$T = \{a, b, c, d, e, f, \text{``},\text{''}, (,)\}$$

$$P = \{A \to a, A \to b(B, C), B \to c(A), B \to d(A, C, A), C \to e, C \to f(B, A)\},$$

where a, b, c, d, e and f are superior functions.

The problem considered by Knuth is the one of finding, for a given variable, the smallest value which it can be assigned in the language generated by the grammar. In the example, if the functions are defined as follows:

$$a = 4,$$

$$b(x, y) = max\{x, y\},$$

$$c(x) = x + 1,$$

$$d(x, y, z) = x + max\{y, z\},$$

$$e = 9,$$

$$f(x, y) = 0.5(x + y + max\{x, y\}),$$

then, the smallest value that A, B and C can assume is 4, 5 and 7 respectively.

The hypergraph formulation A *scfg* can be suitably represented as a hypergraph, where V is the set of the nodes, and there is one hyperarc for each production. Namely, the production $y \to g(x_1, x_2, \ldots, x_k)$ is represented by the hyperarc $(\{x_1, x_2, \ldots, x_k\}, y)$, and g can be viewed as a weighting function associated with the hyperarc (it gives the value of the head node once known the values of the tail nodes). When g is a constant, then the associated production can be thought of as a tailless hyperarc.

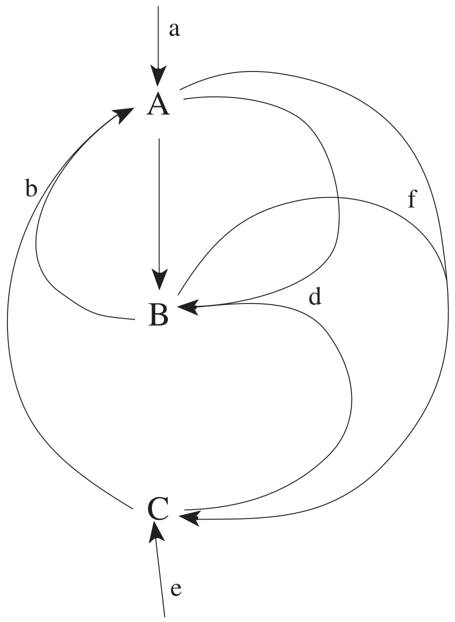

Figure 4: A hypergraph for *scfg*

By considering our sample example, we get the hypergraph in figure 4.

The problem of finding the smallest value corresponding to a given variable can be efficiently solved by computing a minimum cost hyperpath, in the above introduced hypergraph, from a dummy node, which is the tail node of the arcs corresponding to constants, towards the node corresponding to the variable. In fact, by definition, the superior functions are non-negative additive cost functions which satisfy the requirement concerning non-decreasing directed cycles. The minimum cost hyperpath computation can thus be performed via the algorithms in [14]. One of these, called SBT-Dijkstra by the authors, is essentially equivalent to the generalization of Dijkstra's algorithm proposed by Knuth in [27].

4.1.2 Propositional Horn languages

The problem Let P be a set of n atomic propositions, which can be either true or false, and denote by t a proposition which is always true, and by f a proposition which is always false.

Let C be a set of m clauses, each of the form $p_0 \leftarrow p_1 p_2 \dots p_q$, where, for $i = 0, 1, \dots, q$, $p_i \in P \cup \{f, t\}$. The meaning is that p_0 must be true when all the propositions $p_1, \dots, p_q$ are true. If this is the case, i.e. either p_0 is true or p_i is false for at least one index $i \in \{1, \dots, q\}$, then the clause is true; otherwise (p_0 is false and $p_1, \dots, p_q$ are true) the clause is false.

The disjunction p_0 is the consequence of the clause, while the conjunction $p_1 p_2 \dots p_q$ is called the implicant. If $p_0 = f$ the meaning of the clause is that $p_1, \dots, p_q$ can not be all true, while, if $q = 1$ and $p_q = t$, then p_0 must be true.

A *truth evaluation* is a function $v : P \rightarrow \{false, true\}$. If there is a truth evaluation which makes all the clauses true, then C is said to be satisfiable, otherwise it is unsatisfiable. The *Horn Satisfiability problem* (HORN-SAT) is defined as follows:

Input: A set P of n propositions, and a set C of m clauses over $P \cup \{f, t\}$;
Output: "yes" if C is satisfiable, " no" otherwise.

It is well known that, while the general satisfiability problem is NP-complete ([7], [19]), HORN-SAT is polynomial; in fact, it can be solved in linear time ([23], [8]). Unfortunately Max-HORN-SAT, i.e. the variant requiring the maximization of the number of satisfied clauses, remains NP-hard [25].

The hypergraph formulation In order to represent any given instance π in HORN-SAT, we can introduce a hypergraph, H_π, with one node for each element of $P \cup \{f, t\}$, and a hyperarc e for each clause $p_0 \leftarrow p_1 p_2 \dots p_q$, with $h_e = p_0$ and $T(e) = \{p_1, p_2, \dots, p_q\}$, which represents both the consequence and the implicant of the clause (the labelled graphs introduced by Dowling and Gallier to represent HORN-SAT [8] have a direct interpretation in terms of hypergraphs).

It is easy to see that the satisfiability problem is equivalent to the problem of finding a directed hyperpath in H_π from node t to node f, i.e. to check whether node f is connected to node t in H_π. Therefore, any instance of HORN-SAT can be solved in linear time through the connectivity procedure mentioned in 3.2 which, starting from t, finds all the nodes p which are connected to t, i.e. for which a directed hyperpath from t to p exists.

Max-HORN-SAT, too, can be approached via hypergraph computations. A hypergraph-based approach to Max-HORN-SAT is presented in [12], where hypergraph formulations of the problem are used both to find tight relaxations and to generate cutting planes.

4.2 Relational Databases

Database concepts A *Relational Database* can be thought of as a table (or possibly a collection of tables), with a column for each attribute. An example taken from the paper by Bala and Martin [1], on which this section is based, is reported in figure 5.

Student ID	First Name	Last Name	Login Name	Machine Name	Internet Address
MA1836	David	Crockett	da.croc	suna	da.croc@suna.cs.edu
Ma1845	Gregorio	Esparza	gr.espa	vax1	gr.espa@vax1.cs.edu
BS1848	Susan	Bennet	su.benn	suna	su.benn@suna.cs.edu
.	.	.	.	.	.
.	.	.	.	.	.
MA1812	Daniel	Cloud	da.clou	sunb	da.clou@sunb.cs.edu

Figure 5: A Relational Database

Let U be the set of all the attributes of the database (U={Student ID, First Name, Last Name, Login Name, Machine Name, Internet Address} in our example).

Each attribute $a \in U$ has a domain, $dom(a)$, which is the set of all possible values it can take.

A subset of U, $R = (a_1, a_2, \dots, a_q)$, is called a *relation scheme*, while a *relation* over R, $r(R)$, is a subset of the Cartesian product $dom(a_1) \times dom(a_2) \times \dots \times dom(a_q)$. In particular, a relation over the set U is called a *universal relation*.

Given a relation scheme, R, a *functional dependency* (*fd*) is a constraint imposed on R which takes the form $X \to y$, where X is a subset of R and y is an element of R.

A relation over R is said to satisfy the functional dependency $X \to y$ if, for any two elements μ and σ of the relation, $\mu(X) = \sigma(X) \to \mu(y) = \sigma(y)$, where, given an element μ, $\mu(X)$ denotes its *projection* on the subspace of the attributes in X, for any $X \subseteq R$. In other words, there is a unique set of values which the attribute y can take for any set of values taken by the attributes in X.

Continuing with our example, the following is a list of functional dependencies which can reasonably be imposed on the relation scheme U:

f_1: {Student ID} $\to$ {First Name}

f_2: {Student ID} $\to$ {Last Name}

f_3: {First name, Last Name} $\to$ {Login Name}

f_4: {Login Name, Machine Name} $\to$ {Student ID}

f_5: {Login Name, Machine Name} $\to$ {Internet Address}

f_6: {Internet Address} $\to$ {Student ID}

f_7: {Internet Address} $\to$ {Login Name}

f_8: {Internet Address} $\to$ {Machine Name}.

Given a set of functional dependencies F, a functional dependency f is implied by F if it is the logical consequence of some of the $fd's$ in F. As an example, {Student ID} $\to$ {Login Name} is a logical consequence of f_1, f_2 and f_3.

The *closure* of F, F^*, is the set of all the $fd's$ which are implied by F. The following set of axioms is complete and sound for its determination; that means that, by iterating the application of these axioms, it is possible to derive all the $fd's$ implied by F:

1. $y \in X \subseteq U \Rightarrow X \to y$ (reflexivity)
2. $X \to y$ and $Z \subseteq U \Rightarrow (X \cup Z) \to y$ (augmentation)
3. $X \to y, \forall y \in Y$, and $Y \to z \Rightarrow X \to z$ (transitivity)

Given a relation scheme R, on which a set of $fd's$, F, is defined, $X \subseteq R$ is a *superkey* for R if the fd $X \to y$ is in F^*, $\forall y \in R$. The subset X is said to be a *key* for R if it is a minimal superkey, i.e. none of its proper subsets is a superkey for R.

The concept of key is crucial to databases because keys provide a reasonable way to access a database and to retrieve its informations. In principle, we would like to have keys of small cardinality in order to avoid the need of checking too large portions of a database at each access. Unfortunately, the problem of finding a minimum cardinality key is NP-hard [29].

A further basic concept in databases is *decomposition*. Although, in principle, a database can be made by only one single relation, like the one of our example, this is not a practical solution, at least in general. One of the reasons is that we may have to access with high frequency only small portions of the database, and we do not want, at each access, to have to operate on all the informations it contains.

What it is usually done is to decompose the main relation scheme, say R, into smaller relation schemes, R_1, R_2,..., R_q. Of course we do not want to lose any relevant information, which implies that the following properties must be satisfied:

1. $\bigcup_{i=1}^{q} R_i = R$;

2. $(\bigcup_{i=1}^{q} F_{R_i}^*)^* = F^*$,

where F denotes the set of $fd's$ defined on R, and, for each i, F_{R_i} is the subset of the $fd's$ $(X \to y) \in F$ such that $(X \cup \{y\}) \subseteq R_i$.

A decomposition satisfying the above requirements is called a *dependency preserving decomposition.* An interesting problem is to check whether a given decomposition is dependency preserving; this question can be answered in polynomial time, as we shall see later.

The hypergraph formulation Let us consider a relation scheme R and a set of functional dependencies, F, defined on R. F can be modelled in terms of a hypergraph, $H_F = (R, E_F)$, where $E_F = \{(X, y) : X \to y \in F\}$.

Since the axioms which define the closure of F are identical to the connectivity axioms, which specify the relation *connected to* in directed hypergraphs, there is an isomorphism between functional dependencies implied by F and directed hyperpaths in H_F. As a straightforward consequence, many of the database problems which have been introduced before can be reformulated and efficiently solved in terms of directed hyperpath computations.

As a first example, the closure membership problem, i.e. the problem of checking whether a given fd $X \to y$ is in the closure of F, can be solved in linear time by means of a hypergraph visit starting from X. Such a visit finds all the nodes connected to X: if y is connected to X, then $X \to y$ belongs to the closure of F.

A visit algorithm can also be used to find whether a decomposition $\{R_1, R_2, \ldots, R_q\}$ of R is dependency preserving. It is enough to consider the sub-hypergraph of H_F containing only the hyperarcs whose correspondent dependencies are completely contained in one of the schemes R_1, R_2,..., R_q, and, for each dependency $X \to y$, for which the set $X \cup \{y\}$ is split between different schemes, to check whether there is a directed hyperpath from X to y. Again, the connectivity procedure can be used.

A further problem which, under some additional conditions, can be solved by making use of hyperpath computations, is the problem of finding a key. Consider the hypergraph H_F previously defined, extended with a dummy node, say δ, and a hyperarc (δ, y) for each node y. Assign cost 1 to all the dummy hyperarcs (δ, y), and cost 0 to all the original hyperarcs of H_F.

Then it is possible to prove that, if H_F is cycle-free, and the set of the minimum cost hyperpaths outgoing from the dummy node δ is computed, the set X of the attributes corresponding to the nodes for which the hyperarc (δ, y) belongs to a computed minimum cost hyperpath is a key. Furthermore, this key is unique.

An application to Decision Support Systems is described in [2], where directed hypergraphs are called “metagraphs”.

4.3 Production and Manufacturing Systems

4.3.1 Linear Production Systems

The problem A *linear production system* describes an economic system, where goods are consumed (inputs) or produced (outputs) by the system activities ([11], [21], [42]). The system is said to be *simple* if each activity produces exactly one output. Generally, the goods are classified as follows:

primary goods: they are goods like plant equipment and labor (sometimes, they are also referred to as factors of production), which are not outputs of activities of the system;

intermediate goods: they are goods produced by the system activities;

target goods: they are the goods required by consumers.

Consider for instance the simple linear production system which describes the production of cars. In such a system, primary goods like aluminium, steel and labor are required to produce engines, whereas steel, labor, glass and some plastic components must be provided as inputs to make up the body of the cars; finally, engines, bodies and tires, when properly assembled, allow one to produce cars.

The hypergraph formulation By introducing one node for each good, each activity of a simple linear production system can be represented by means of a hyperarc, linking the inputs to the output. The entire economic system can thus be described by a directed hypergraph, as shown in figure 6 for the example concerning the production of cars.

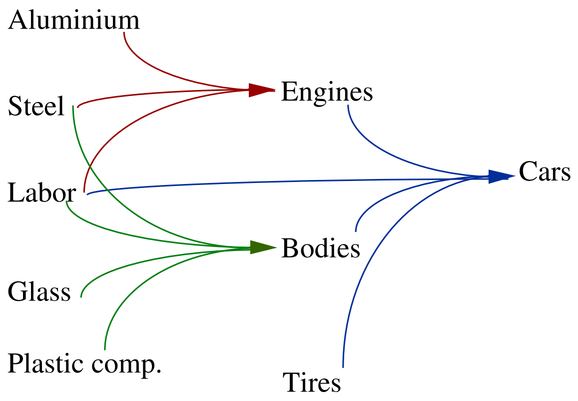

Figure 6: A hypergraph for car production

When a cost and a capacity are associated with each system activity, representing the production cost at the unitary level and the maximum amount of output which can be produced, respectively, then the problem of producing given amounts of the target goods at minimum cost can be modelled in terms of a minimum cost hyperflow problem on the linear production hypergraph. In the above model, different hyperarcs having the same head can be used both to model activities using alternative technologies for producing the same good.

Similarly, the problem of finding the maximum amount of target good that can be produced by the system can be formulated as a maximum hyperflow problem on the linear production hypergraph.

Moreover, when no capacity constraints are imposed on the production activities, the so-called *efficiency prices* relative to the economic system (see, for instance, [21]) can be found simply by solving an uncapacitated minimum cost hyperflow problem. Being the problem uncapacitated, it is equivalent to solving a minimum cost hyperpath problem, with a *value* cost function.

4.3.2 Assembly problems

The problem In robotic assembly systems, items are produced by assembling together parts given in input, according to a properly defined assembly plan, in order to satisfy a given criterion.

More formally, let Q be the set of the input (distinct) parts. An *assembly* is an item obtained by properly connecting some of the parts in Q by means of *feasible assembly operations*, where a feasible assembly operation $\omega = (\{\pi_1, \pi_2\}, \pi)$ is the operation to construct the item π by assemblying together the two items π_1 and π_2.

Let Π denote the set of all the assemblies which can be obtained from the parts in Q by means of sequences of feasible assembly operations, and Ω be the corresponding set of feasible assembly operations.

Given an item $\pi^* \in \Pi$, by *assembly plan* relative to π^* we mean a minimal set of feasible assembly operations which, starting from the parts in Q, leads to the production of π^*, where by minimal we mean that none of its proper subsets is an assembly plan.

An *optimal assembly plan problem* $(Q, \Pi, \Omega, \pi^*, \phi)$ is the problem of finding an assembly plan relative to π^* which satisfies a given criterion ϕ, usually written as a function to be minimized.

The hypergraph formulation Assembly plans can be formulated in terms of hypergraphs ([33], [15], [16]). In fact, if we associate one node with each assembly in Π, and a hyperarc with each feasible assembly operation $\omega = (\{\pi_1, \pi_2\}, \pi)$, having $\{\pi_1, \pi_2\}$ as its tail and π as its head, then there is a one-to-one correspondence between the set of the feasible assembly plans relative to π^* and the set of the hyperpaths from Q to π^* in the obtained *assembly hypergraph* $H = (V, E)$, where $V = \Pi$ and $E = \Omega$.

Several optimal assembly plan problems can thus be formulated and solved in terms of computations on H, which is acyclic by construction.

As a first case, assume that each feasible assembly operation is assigned a real cost, and that the *total production cost* of an assembly plan is the sum of the costs of all the operations performed (if an operation is performed $p > 1$ times, then its contribution to the total cost is p times its cost). From the hypergraph interpretation just introduced, it follows easily that an assembly plan which minimizes the production cost can be computed by finding a minimum value hyperpath in H from the node set Q to π^*, when the coefficients $a(w, e)$ are all set equal to 1 (see equations (3) in Section 3.2). Being H acyclic, the problem can thus be solved in linear time, according to the results reviewed in 3.2.

As a second interesting case, assume that each feasible assembly operation ω has an associated real value, $p(\omega)$, representing the probability that the execution of ω produces a defective item. Then, if we denote by $P(\pi)$ the probability that there is at least one fault in the assembly π, it is easy to show that the operation $\omega = (\{\pi_1, \pi_2\}, \pi)$ may introduce a fault in π with probability:

$$P(\pi) = P(\pi_1) + P(\pi_2) - P(\pi_1)P(\pi_2) + p(\omega)(1 - P(\pi_1))(1 - P(\pi_2)).$$

By interpreting $p(\omega)$ as a cost associated with ω, an assembly plan which minimizes the fault probability relative to π^* can be computed by finding a minimum cost hyperpath in H, from the nodes in Q to node π^*, with respect to the additive cost function where function F is defined as follows:

$$F = [P(\pi_1) + P(\pi_2) - P(\pi_1)P(\pi_2)](1 - p(\omega)).$$

This problem, too, can thus be solved in linear time.

A more interesting (and difficult indeed) case is the minimum makespan problem. Assume that each feasible assembly operation $(\{\pi_1, \pi_2\}, \pi) \in \Omega$ is assigned an integer duration, $d(\{\pi_1, \pi_2\}, \pi)$, and that k identical machines (robots) are available to perform the operations. The *Minimum Makespan Assembly Plan Problem* (MMAP) consists in finding an assembly plan relative to π^* which minimizes the time (*makespan*) needed to produce one unit of π^* on the k available machines.

Two particular easy cases are the one in which $k = 1$ (i.e. all the operations are performed in sequence) and the one in which $k = \infty$ (i.e. any number of parallel operations can be performed).

The case $k = 1$ is a special case of the minimum production cost problem introduced above, and therefore it can be solved in linear time by finding a minimum value hyperpath in H from the node set Q to π^*. Once an optimal assembly plan has been found, the corresponding assembly operations can be scheduled in any order on the machine, provided that the precedence constraints imposed by the plan are satisfied.

As shown in [15], the other extreme case, that is $k = \infty$, reduces to the problem of finding a minimum distance hyperpath in H from the nodes in Q to π^* (see equations (2) in Section 3.2). Also this problem can thus be solved in linear time. In this case, scheduling the operations in the optimal assembly plan can be obtained by the following simple rule: assign to the first available machine the first available operation, where the operations are ordered by going from the lower levels of the optimal hyperpath towards π^*.

The general case ($1 < k < \infty$) is NP-hard. In fact it has as a particular case the problem of scheduling n jobs on k identical machines, by satisfying a tree precedence relation, with the minimum makespan objective function. This is the case of instances with only one feasible assembly plan. In this case (i.e. one single assembly plan), a heuristics has been described in [15], which approximates the optimum makespan with a relative error bounded from above by $\frac{k-1}{k}$. Such a heuristics has been extended to the case of many feasible assembly plans in [16]. It has been proved that, when all the assembly plans relative to π^* have the same total duration, i.e. the sum of the durations of all the operations is the same for each assembly plan, then the bound $\frac{k-1}{k}$ still holds.

A special case, for which a polynomial time algorithm exists, is described in [16].

4.3.3 Cutting stock problems

The problem Let be given an unlimited number of pieces of stock, characterized by different *stock lengths* s_k, $k = 1, 2, \ldots, K$, and assume that we want to cut out from such pieces of stock d_1 pieces of length b_1, d_2 pieces of length b_2, $\ldots$, d_I pieces of length b_I (all the input data are assumed to be integer).

The *One-Dimensional Cutting Stock* Problem (CSP) consists in cutting the pieces of stock in such a way that the length requirements be satisfied, and the overall residual length be minimized (in alternative formulations, the total number of the used pieces of stock is minimized).

Two main approaches are generally followed for the solution of CSP: heuristics, and linear programming relaxations [9]. LP-based approximation algorithms first solve the linear programming relaxation of some CSP model, and then search for an integer solution by means of a more or less sophisticated strategy, usually by simply rounding up. Extensive computational experiments reveal that the gap between the optimal (integer) value and the linear programming bound is usually small ([39], [40], [41]). Therefore, efficient approaches to solve linear programming relaxations of CSP are useful in practice.

A classical approach to formulate the linear programming relaxation of CSP is based on the definition of cutting patterns. A *cutting pattern* (or simply pattern) is an integer vector $(a_1, a_2, \ldots, a_I)$, denoting that a piece of stock is cut in such a way to obtain exactly a_i pieces of length d_i, $i = 1, 2, \ldots, I$. An *elementary pattern* is a pattern having 1 in one only position, and 0 otherwise (hereafter P_i will denote the i^{th} elementary pattern, having 1 inposition i, $i = 1, 2, \ldots, I$).

A cutting pattern $(a_1, a_2, \ldots, a_I)$ is said to be *feasible* if $\sum_i a_i d_i$ is less than or equal to some stock length s_k, producing a *leftover* of $s_k - \sum_i a_i d_i$.

A linear programming relaxation of CSP consists in choosing feasible patterns in such a way to satisfy the stock requirements and minimize the overall leftover, accepting also fractions of patterns.

The hypergraph formulation Any instance P of CSP can be modelled by means of a hypergraph H_P defined as follows. The set of the nodes of H_P is in one-to-one correspondence with the feasible patterns of P. To define the hyperarc set, assume that the order relation $d_1 < d_2 < \ldots < d_I$ holds among the required lengths. Then, each feasible pattern $A = (a_1, a_2, \ldots, a_j, 0, \ldots, 0)$, where a_j, $j \leq I$, is the maximum (with respect to the given ordering) non zero component, can be uniquely defined according to the following recursive rule:

$$(a_1, a_2, \ldots, a_j, 0, \ldots, 0) = (a_1, a_2, \ldots, a_j - 1, 0, \ldots, 0) + P_j, \tag{6}$$

where P_j is the j^{th} elementary pattern. Therefore, by recoursively applying rule (6), each pattern A can be viewed as the result of the successive composition of elementary patterns.

Accordingly, let us introduce a hyperarc of H_P for each feasible pattern A, which has the node corresponding to A as the head, and the nodes corresponding to the two patterns in (6) as the tail nodes.

A particular case is when $a_1 = a_2 = \ldots = a_j - 1 = 0$ and $a_j = 2$, i.e. $(a_1, a_2, \ldots, a_j - 1, 0, \ldots, 0)$ coincides with P_j; in this case, the hyperarc has the node corresponding to P_j as the only tail node, and a multiplier equal to 2 is associated with the hyperarc, denoting that two occurencies of the elementary pattern P_j need to be used (*multiple hyperarcs*).

By construction, H_P is a layered hypergraph; therefore it is acyclic.

CSP can be formulated in terms of a minimum cost hyperflow on H_P, when extended with a headless hyperarc $(v, \emptyset)$ for each node v. In the hyperflow model, the flow on a headless hyperarc $(v, \emptyset)$ will represent how many times the feasible pattern corresponding to v is used to satisfy the length requirements.

In order to obtain such a hyperflow interpretation, let us associate a hyperflow multiplier equal to 1 with each node $v \in T(e)$, for each hyperarc e in H_P, with the exception of multiple hyperarcs, having 2 as the hyperflow multiplier (recall that they model feasible patterns made up of the same two elementary patterns). In addition, let us associate hyperflow demands with the nodes as follows: each node v corresponding to the i^{th} elementary pattern, and so to the i^{th} length d_i, has a hyperflow demand $b(v) = -b_i$; for each other node v, it is $b(v) = 0$.

Then, if integer constraints are imposed on the hyperflow variables, each feasible hyperflow on H_P uniquely identifies a feasible way to cut the pieces of stock in order to satisfy the length requirements: the cutting stock solution is given by the hyperflows associated with the headless hyperarcs, i.e. the hyperflow on a headless hyperarc $(v, \emptyset)$ gives the number of times we use the pattern corresponding to v.

An example is provided in figure 7 for an instance where $s = 5$ is the unique stock length, $d = (1, 2)$ is the ordered list of the required lengths, and the stock requirement vector is $b = (3, 2)$, i.e. we want 3 pieces of length 1 and 2 pieces of length 2. The hyperflow demands are the numbers in the [] brackets. The hyperflow corresponds to the use of the two patterns (1,2) and (1,0).

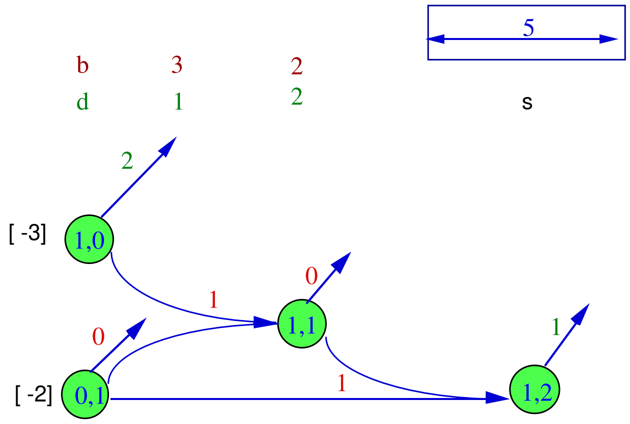

Figure 7: A hyperflow for CSP

Given a node v in H_P, corresponding to some pattern $A = (a_1, a_2, \ldots, a_I)$, let us associate the cost

$$c((v, \emptyset)) = min\{s_k : s_k \geq \sum_i (a_i d_i), k = 1, \ldots, K\} - \sum_i (a_i d_i)$$

with the headless hyperarc $(v, \emptyset)$ outgoing from v, in order to represent the minimal residual length obtained using pattern A once. The remaining hyperarcs of H_P are given zero cost.

For example, in our "sample instance", the headless hyperarcs $((1, 0), \emptyset)$, $((0, 1), \emptyset)$ and $((1, 1), \emptyset)$ are given the cost 4,3 and 2, respectively, whereas the other hyperarcs have a zero cost.

Then, it is easy to see that the (uncapacitated) integer minimum cost hyperflow problem on H_P models CSP. In fact, an integer hyperflow, i.e. a feasible cutting strategy, that minimizes the overall residual length of the cut pieces of stock is looked for.

By relaxing the integrality constraints, the resulting (linear) minimum cost hyperflow problem on H_P models the linear programming relaxation of CSP [18].

4.4 Public Transportation Models

The problem A *transit system* can be modelled as a special network in which there are three sets of nodes and three sets of arcs:

N_c: nodes representing the centroids of the zones in which the area is partitioned,

N_s: nodes corresponding to the geografical location of the bus stops,

N_l: nodes corresponding to the line stops,

E_{cs}: arcs connecting, in either direction, centroids and stops,

E_{sl}: arcs connecting, in either direction, nodes in N_s and nodes in N_l,

E_{ll}: arcs representing line segments between two consecutive line stops.

For each bus stop, i, let L_i denote the set of all the lines which stop at i, $v_{i0} \in N_s$ be the node representing the geografical location of the stop, and $v_{ij} \in N_l$ be the node corresponding to the stop of line $j \in L_i$.

Consider a passenger waiting at i, which wants to minimize the expected travel time from i to a certain destination s. In practice, being the actual travel time known in advance only through a rather rough estimate, it may happen that, instead of a single preferable line, there is a set of lines, say $L^*_{is} \subseteq L_i$, which are almost undistinguishable from the traveller's point of view, in the sense that he/she is willing to board the first carrier of these lines arriving at stop i. This set of lines is called an *attraction set* for all the passengers travelling from i to s. Generally, for each stop i, many possible attraction sets exist, depending on the destination, and, for a given destination, on the lines' frequencies and on the buses' speed.

A typical problem arising in transportation planning is then the *Traffic Assignment* problem, which is the problem of determining the passengers' flow on the lines under the assumption that each passenger tries to minimize his/her expected travel time. That is, what is needed is to compute the minimum expected travel time, for each origin/destination pair.

Another interesting and much harder problem indeed is the capacitated case. Here we assume that some arcs have capacities and hence that only a limited number of travellers can pass through them. As an example, consider a bus line in a rush hour: the passenger willing to board on the 8:00 bus may be obliged to wait for the 8:40 because the bus capacity has been saturated.

The hypergraph formulation Standard graphs are not a sufficiently powerful means to model the passenger behaviour and to solve the Traffic Assignment problem. In fact, the requirement that, for each pair (i, s) and each possible attraction set L^*_{is}, the entire set of lines in L^*_{is} has to be considered, cannot be captured by means of standard graph notations. Instead, directed hyperarcs of the type $(v_{i0}, \{v_{ij} : j \in L^*_{is}\})$, can be appropriately added to the transit network to represent the attraction sets. Such hyperarcs will be called *boarding hyperarcs*.

An example is provided in figure 8, where a portion of a city map is suitably modelled by means of boarding hyperarcs. In this example there are 4 bus stops (A, B, C and D), and 4 lines (L_1, L_2, L_3 and L_4). The hypergraph represents the set of the possible routes a passenger going from A to B can take. Arcs (D,B) and (C,B) represent walking portions of the trip.

Consider the following notation, where $L \subseteq L_i$ denotes a subset of lines stopping at i:

f_j: the frequency of line j;

$\pi_j(L)$: the probability that a bus serving line $j \in L$ will arrive at stop i before the buses serving any other line in L;

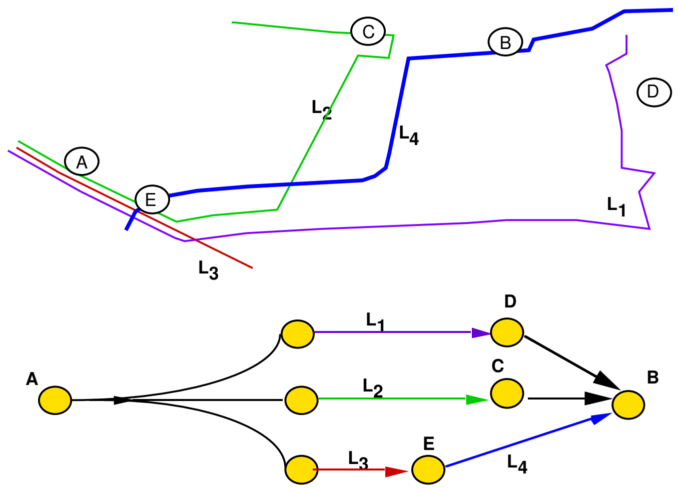

Figure 8: A transit hypergraph

$t_j(i,s)$: the expected travel time between stop i and destination s if line j is used at stop i, not including the waiting time at i;

$w(L)$: the average waiting time for a traveller whishing to travel on any of the lines in L.

The expected travel time from i to s for a passenger considering the attraction set L^*_{is} is then

$$T_{is}(L^*_{is}) = w(L^*_{is}) + \sum_{j \in L^*_{is}} t_j(i,s)\pi_j(L^*_{is}),$$

where, under reasonable hypotheses on the distribution of passenger and bus arrivals at the stops ([34], [35]), it is:

$$\pi_j(L) = \frac{f_j}{\sum_{h\in L} f_h},\ w(L) = \frac{1}{2\sum_{h\in L} f_h}.$$

As far as the travel times $t_j(i,s)$ are concerned, they are composed of walking times, in-vehicle travel times and waiting times associated with transfers from one line to another one. These times can be associated with the arcs of the transit network: the lengths of in-vehicle arcs (E_{ll}) are the corresponding buses' travel times, the lengths of walking arcs (E_{cs}) are walking times, and the lengths of leaving arcs (E_{sl}) are set to 0. On the other hand, the waiting times are associated with the boarding hyperarcs.

At least in principle, the Traffic Assignment problem can then be solved by

1. building the hypergraph representing the transit network;
2. changing the directions on all the arcs, obtaining a new hypergraph in which all the hyperarcs are of the type (X, y), where X is an attraction set for travellers embarking at stop y;
3. solving a minimum cost hyperpath computation for each node s which is the destination of some trip.

The main problem with this approach is in the huge number of distinct attraction sets. As shown by Nguyen and Pallottino [35], all the attraction sets at each stop can be considered in an implicit way at low (polynomial) computational cost. In fact they have proved that, for each origin/destination pair (i, s), the optimal attractive set can be obtained by a simple greedy procedure.

To tackle the capacitated case, Marcotte and Nguyen [31] have introduced the concept of *travel strategy*. The idea is that, at each node in its trip, the traveller faces a certain number of alternatives, one for each of the arcs in the node's forward star. The traveller tries first the one corresponding to the minimum cost path to his/her destination; if such an arc is not available due to capacity limitation, he/she tries the one corresponding to the second minimum cost path, and so on until an available arc is found. Each of these arcs has attached a weight, i.e. the probability of finding the arc available, which depends on the total number of travellers wanting to use that arc. This situation, too, can be represented by hyperarcs: the set of the alternatives faced by a traveller passing through node x can be modelled by a hyperarc of the type (x, Y).

With this notation, a travel strategy is a hyperpath, and the average cost of a hyperpath (travel strategy), σ, from x to a given destination node s is given by

$$t_\sigma(x, s) = \sum_{j \in Y} (\tau(x, j) + t_\sigma(j, s))\pi_Y(x, j),$$

where (x, Y) is the unique hyperarc exiting from x in the hyperpath σ, $\pi_Y(x, j)$ is the probability for the traveller of passing through node j after leaving node x, and $\tau(x, j)$ is the transit time associated with the arc (x, j).

Using the concept of hyperpath it is possible to derive equilibrium conditions of the same type used in the uncapacitated traffic assignment models. In fact, the well known Wardrop equilibrium conditions can be restated for the capacitated case as follows: a traffic flow is an equilibrium flow when, for each origin/destination, the expected travel time on all the used hyperpaths is the same, and this common value is not higher than the expected travel time on any unused hyperpath.

References

[1] M. Bala and K. Martin. A mathematical programming approach to data base normalization. Technical report, Graduate School of Business, University of Chicago, 1993.

[2] A. Basu and R.W. Blanning. Metagraphs: A tool for modeling decision support systems. *Management Science*, 40(12):1579–1600, 1994.

[3] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, 1973.

[4] C. Berge. Minimax theorems for normal hypergraphs and balanced hypergraphs - a survey. *Annals of Discrete Mathematics*, 21:3–19, 1984.

[5] C. Berge. *Hypergraphs: Combinatorics of Finite Sets.* North-Holland, Amsterdam, 1989.

[6] H. Boley. Directed recursive label node hypergraphs: a new representation language. *Artificial Intelligence*, 9:49–85, 1977.

[7] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3-th ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[8] W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 3:267–284, 1984.

[9] H. Dyckhoff. A typology of cutting and packing problems. *European J. Oper. Res.*, 44:145–159, 1990.

[10] A.L. Furtado. Formal aspects of the relational model. *Information Systems*, 3:131–140, 1978.

[11] D. Gale. *The Theory of Linear Economic Models.* McGraw-Hill, New York, NY, 1960.

[12] G. Gallo, C. Gentile, D. Pretolani, and G. Rago. Max Horn Sat and the minimum cut problem on directed hypergraphs. *Math. Programming*, 80:213–237, 1998.

[13] G. Gallo, N. Licheri, and M.G. Scutellà. The Hypergraph Simplex approach: some experimental results. *Ricerca Operativa*, 78(anno XXVI):21–54, 1996.

[14] G. Gallo, G. Longo, S. Nguyen, and S. Pallottino. Directed hypergraphs and applications. *Discrete Appl. Math.*, 40:177–201, 1993.

[15] G. Gallo and S. Pallottino. Hypergraph models and algorithms for the Assembly Problem. Technical Report TR-6/92, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, 1992.

[16] G. Gallo and M.G. Scutellà. Minimum makespan assembly plans. Technical Report TR-98-10, submitted to Operations Research, Dipartimento di Informatica, Università di Pisa, Pisa, Italy, 1998.

[17] G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *J. Logic Programming*, 7:45–61, 1989.

[18] M. Gambale, M. Nonato, and M.G. Scutellà. The cutting stock problem: a new model based on hypergraph flows. *Ricerca Operativa*, 74(anno XXV):73–97, 1995.

[19] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, San Francisco, CA, 1979.

[20] S. Gnesi, U. Montanari, and A. Martelli. Dynamic programming as graph searching: an algebraic approach. *J. Assoc. Comput. Mach.*, 28:737–751, 1981.

[21] J.M. Henderson and R.E. Quandt. *Microeconomic Theory, a Mathematical approach.* McGraw-Hill, New York, NY, 1971.

[22] J.E. Hopcroft and J.D. Ullman. *Formal languages and their relation to automata.* Addison-Wesley, 1969.

[23] A. Itai and J. Makowsky. On the complexity of Herbrand's theorem. Technical Report T. R. 243, Dept. Comp. Sci., Israel Inst. of Technology, Israel, 1982.

[24] G.F. Italiano and U. Nanni. On line maintenance of minimal directed hypergraphs. In *Proc. 3 Convegno Italiano di Informatica Teorica*, pages 335–349. World Science Press, 1989.

[25] B. Jaumard and B. Simeone. On the complexity of the maximum satisfiability problem for Horn formulas. *Inform. Process. Lett.*, 26:1–4, 1987.

[26] R.G. Jeroslow, R.K. Martin, R.R. Rardin, and J. Wang. Gainfree Leontief substitution flow problems. *Math. Programming*, 57:375–414, 1992.

[27] D.E. Knuth. A generalization of Dijkstra's algorithm. *Inform. Process. Lett.*, 6(1):1–5, 1977.

[28] G. Levi and F. Sirovich. Generalized and/Or graphs. *Artificial Intelligence*, 7:243–259, 1976.

[29] C.L. Lucchesi and S.L. Osborn. Candidate keys for relations. *J. Comput. System Sci.*, 17:270–279, 1978.

[30] D. Maier. Minimum covers in the relational data base model. *J. Assoc. Comput. Mach.*, 27:664–674, 1980.

[31] P. Marcotte and S. Nguyen. Hyperpath formulations of traffic assignment problems. In P. Marcotte and S. Nguyen, editors, *Equilibrium and Advanced Transportation Modelling*, pages 175–200. Kluwer Academic, MA, 1998.

[32] A. Martelli and U. Montanari. Additive And/Or graphs. *Proceedings IJCAI*, 3:1–11, 1977.

[33] L.S. Homem De Mello and A.C. Sanderson. And/Or graph representation of assembly plans. *IEEE Transactions on Robotics and Automation*, 6:188–199, 1990.

[34] S. Nguyen and S. Pallottino. Equilibrium traffic assignment for large scale transit network. *European J. Oper. Res.*, 37:176–186, 1988.

[35] S. Nguyen and S. Pallottino. Hyperpaths and shortest hyperpaths. In B. Simeone, editor, *Combinatorial Optimization*, volume 1403 of *Lecture Notes in Mathematics*, pages 258–271. Springer-Verlag, Berlin, 1989.

[36] N.J. Nilsson. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, NY, 1971.

[37] N.J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA, 1980.

[38] R.Cambini, G. Gallo, and M.G. Scutellà. Flows on hypergraphs. *Math. Programming*, 78:195–217, 1997.

[39] G. Scheithauer and H. Terno. About the gap between the optimal values of the integer and continuous relaxation one-dimensional cutting stock problem. In *Oper. Res. Proc.* Springer-Verlag, Berlin, 1992.

[40] G. Scheithauer and H. Terno. The modified integer round-up property for the one-dimensional cutting stock problem. *European J. Oper. Res.*, 84:562–571, 1995.

[41] G. Scheithauer and H. Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Oper. Res. Lett.*, 20:93–100, 1997.

[42] A. Takayama. *Mathematical Economics*. Cambridge University Press, 1985.

[43] J.D. Ullman. *Principles of Database Systems*. Computer Science Press, Rockville, MD, 1982.