Tree–Based Fixpoint Iteration for Disjunctive Logic Programs

Dietmar Seipel
University of Tübingen
Sand 13, D – 72076 Tübingen, Germany
seipel@informatik.uni-tuebingen.de

Abstract

Non–Horn programs containing disjunctive rules allow for more advanced applications of deductive databases. But the inference process dealing with rapidly increasing amounts of disjunctive facts becomes much more intricate, because of the problems of finding resolvable atoms in the facts and of subsumption elimination in sets of facts.

We present an efficient implementation of the fixpoint computation for the disjunctive consequence operator T_P^S based on the Δ–iteration technique and on a compact data structure of so–called clause trees for disjunctive facts. These trees speed up the combinatorial search in a T_P^S–hyperresolution step. Furthermore, the necessary operations of subsumption elimination can be performed efficiently based on them.

The presented techniques can be used for computing the extended generalized closed world assumption of a disjunctive logic program and the generalized disjunctive well–founded semantics of a disjunctive normal logic program as well.

1 Introduction

Deductive databases use logic programming as a powerful, declarative language for specifying queries to relational databases. There has been a lot of research on processing definite horn clauses with definite facts, cf. [BaRa 86], [Ul 88,89], [CeGoTa 90], where the desired result is the least fixpoint of the definite consequence operator T_P, which equals the minimal Herbrand model $\mathcal{M}_P$, cf. [Llo 87]. The possibly recursive program is evaluated iteratively in a set–oriented fashion based on the primitive operations of relational algebra, which can be executed efficiently on databases.

A very frequently used logic program P is the one that specifies the transitive closure of a graph given by an arc–relation. It consists of two definite logic rules:

$$\begin{array}{lcl} path(X,Y) & :- & arc(X,Z), path(Z,Y). \\ path(X,Y) & :- & arc(X,Y). \end{array}$$

But, when applied to a graph given by indefinite information for arc, e.g. the three disjunctive facts

$$arc(a,b) \vee arc(a,c), \quad arc(b,d) \text{ and } arc(c,d),$$

it is necessary to use the general disjunctive consequence operator T_P^S instead of the definite one for deriving all the definite (e.g. $path(a,d)$) and disjunctive (e.g. $path(a,b) \vee path(a,c)$) facts logically implied by P.

More advanced applications require even the support of disjunctive rules in addition to disjunctive facts: The deduction of independencies in probabilistic reasoning, cf. [Pea 88], and the overlapping of rectangles in spatial databases, cf. [AbWiPa 93], are examples falling into this category. In general, a disjunctive logic rule has a head given by a disjunction of positive atoms and a body given by a conjunction of positive and negated atoms:

$$a_1(X_1) \vee \ldots \vee a_k(X_k) \quad :- \quad b_1(Y_1) \wedge \ldots \wedge b_m(Y_m) \wedge \neg b_{m+1}(Y_{m+1}) \wedge \ldots \wedge \neg b_{m+n}(Y_{m+n}).$$

where $a_i(X_i)$ and $b_i(Y_i)$ are positive atoms with argument vectors X_i and Y_i, respectively. The rule is a definite horn clause, iff $k = 1$ and $n = 0$.

Indefinite rules without negated body literals (i.e. $k > 1$, $n = 0$) specify the set of logically implied disjunctive facts (i.e. positive ground clauses), the so–called minimal model state $\mathcal{MS}_P$, cf. [LoMiRa 92]. Efficient techniques for computing $\mathcal{MS}_P$ iteratively as the least fixpoint of the consequence operator T_P^S are currently investigated. Indefinite rules with negation (i.e. $k > 1$, $n > 0$) can be evaluated by repeated computation of minimal model states w.r.t. certain disjunctive transformations of P and of closed world assumptions, cf. well–founded semantics [VaGeRoSchli 91], generalized well–founded semantics [BaLoMi 89], generalized disjunctive well–founded semantics [LoMiRa 92], stationary semantics [Prz 90] and stable model semantics [GeLi 88].

We will review some of the notations and theoretical foundations for disjunctive logic programming of [LoMiRa 92]. Then, we will describe a bottom–up procedure that iteratively derives the disjunctive facts of $\mathcal{MS}_P$ and organizes them in a tree structure that is used for a fast hyperresolution of further disjunctive facts by T_P^S. For simplicity, we will use an example with definite rules and disjunctive facts. The presented ideas, however, are applicable to disjunctive rules as well.

2 Foundations of Disjunctive Logic Programs

Syntactically, a **deductive database** $DDB = (DB, P)$ is given by a set DB of facts, which resemble a relational database, and a logic program P, which consists of a set of rules for deriving new facts from DB.

(i) Given some atoms $p_i(X_i)$, where $1 \leq i \leq m$, $m \in I\!N_+$, and some literals $L_j(Y_j)$, where $1 \leq j \leq n$, $n \in I\!N_0$. A disjunctive **fact** is given by

$$p_1(X_1) \vee \ldots \vee p_m(X_m).$$

A disjunctive normal **rule** r is given by

$$r: \underbrace{p_1(X_1) \vee \ldots \vee p_m(X_m)}_{\text{head}} \; :- \; \underbrace{L_1(Y_1) \wedge \ldots \wedge L_n(Y_n)}_{\text{body}}.$$

where the atoms $p_1(X_1), \ldots, p_m(X_m)$ form its head and the literals $L_1(Y_1), \ldots, L_n(Y_n)$ form its body. r is disjunctive, iff all its body literals L_i are positive. r is normal, iff $m = 1$. The head atoms are called **dependent** on the body literals. A disjunctive logic program is called **recursive**, iff there is a rule such that a body literal transitively depends on a head atom.

The disjunctive normal logic program containing all the **ground instances** of rules in P is denoted by $ground(P)$.

The **semantics** of a deductive database $DDB = (DB, P)$ is defined according to the bottom–up or naive evaluation algorithms, cf. [LoMiRa 92], as the least fixpoint of a consequence operator T_P^S.

(ii) The **disjunctive Herbrand base** DHB_P of a disjunctive logic program P is given by the set of positive ground clauses

$$\mathrm{DHB}_P \;=\; \{\; A_1 \vee \ldots \vee A_k \mid k \geq 1,\; A_i \in \mathrm{HB}_P,\; 1 \leq i \leq k \;\}.$$

A **Herbrand state** S of P is a subset $S \subseteq \mathrm{DHB}_P$. The canonical form and the expansion of S are given by

$$\begin{aligned} can(S) \;&=\; \{\; C \in S \mid \text{there is no proper subclause } C' \text{ of } C \text{ in } S \;\}, \\ exp(S) \;&=\; \{\; C \in \mathrm{DHB}_P \mid \text{there is a subclause } C' \text{ of } C \text{ in } S \;\}, \end{aligned}$$

respectively, i.e. $can(S) \subseteq S \subseteq exp(S)$. S is called expanded, iff $exp(S) = S$.
The **set difference** with subsumption of two Herbrand sets S_1 and S_2 is given by

$$S_1 \setminus_s S_2 \;=\; \{\; C \in S_1 \mid C \text{ is not subsumed by any clause in } S_2 \;\}.$$

(iii) The **disjunctive consequence operator** $T_P^S : 2^{\mathrm{DHB}_P} \longrightarrow 2^{\mathrm{DHB}_P}$ of a disjunctive logic program P maps Herbrand states to Herbrand states:

$$\begin{aligned} T_P^S(D) \;=\; \{\; C \in \mathrm{DHB}_P \mid\; & C' \;\text{:-}\; B_1, \ldots, B_n \text{ is in } ground(P), \\ & B_1 \vee C_1, \ldots, B_n \vee C_n \text{ are in } D,\;\; C \text{ is the smallest factor of } C' \vee C_1 \vee \ldots \vee C_n \}. \end{aligned}$$

A **model state** S of a disjunctive logic program P is an expanded Herbrand state of P such that every minimal Herbrand model of S is a Herbrand model of P.

Example 2.1 ([LoMiRa 92]) *Given the disjunctive logic program*

$$\begin{aligned} P \;=\; \{\;\; & getto(X,Y) \vee getto(X,Z) \;\text{:-}\; path(X,Y,via(Z)), at(X); \\ & at(X) \;\text{:-}\; getto(Y,X); \\ & path(was, ny, via(phil)) \vee at(ny) \;\}. \end{aligned}$$

For the state $S_1 \;=\; \{\; path(was, ny, via(phil)) \vee at(ny) \;\}$ *we have*

$$\begin{aligned} T_P^S(S_1) \;=\; \{\; & path(was, ny, via(phil)) \vee at(ny), \\ & getto(was, ny) \vee getto(was, phil) \vee path(was, ny, via(phil)) \vee at(ny) \;\} \end{aligned}$$

and for the state $S_2 \;=\; T_P^S(S_1)$ *we have*

$$\begin{aligned} T_P^S(S_2) \;=\; T_P^S(S_1) \;\cup\; \{\; & at(ny) \vee getto(was, phil) \vee path(was, ny, via(phil)), \\ & at(phil) \vee at(ny) \vee getto(was, ny) \vee path(was, ny, via(phil)) \;\}. \end{aligned}$$

For every **definite** logic program P the following sets are equal, cf. [Llo 87]: the logical consequences $\{\; A \in \mathrm{HB}_P \mid P \models A \;\}$ of P, the minimal Herbrand model $\mathcal{M}_P$ of P, the least fixpoint $lfp(T_P)$ of the definite consequence operator T_P, and $T_P \uparrow \omega$.

In general a **disjunctive** logic program P does not have a unique minimal model, but a set $\mathcal{MM}_P$ of minimal models. The following theorem relates model states to the disjunctive consequence operator and shows that – as for Herbrand models of definite logic programs – every intersection of model states again is a model state.

Theorem 2.2 (Model States) *Given a disjunctive logic program P.*

1. *Characterization: An expanded Herbrand state D of P is a model state of P, iff $T_P^S(D) \subseteq D$.*
2. *Intersection Property: The intersection $\cap_{i \in J} MS_i$ of every non–empty set $\{MS_i\}_{i \in J}$ of model states of P is a model state of P too.*

Thus, there is a unique **minimal model state** $\mathcal{MS}_P$ of a disjunctive logic program P, namely the intersection $\cap_{i \in J} MS_i$ of the set $\{MS_i\}_{i \in J}$ of all model states of P.

According to [LoMiRa 92], the following Herbrand states are equal:

1. the logical (disjunctive) consequences $\{ C \in \mathrm{DHB}_P \mid P \models C \}$,
2. the minimal model state $\mathcal{MS}_P$ of P,
3. the expanded least fixpoint $exp(lfp(T_P^S))$ of T_P^S,
4. the expansion $exp(T_P^S \uparrow \omega)$ of the ω–power $T_P^S \uparrow \omega$ of T_P^S, and
5. the set $\{ C \in \mathrm{DHB}_P \mid \forall M \in \mathcal{MM}_P : M \models C \}$ of clauses implied by all minimal models of P.

For a definite logic program P, the minimal Herbrand model $\mathcal{M}_P$ is equal to the canonization $can(\mathcal{MS}_P)$ of the minimal model state.

3 Optimization Techniques for the Fixpoint–Computation

A well–known paradigm of recursive query processing is Δ–iteration or semi–naive iteration, cf. [BaRa 86], [GüKiBa 87]. Up to now it has been used mainly for the fixpoint computation of definite consequence operators. It can of course be used for the disjunctive consequence operator as well. In [KöKiThöGü 93] it has been shown that Δ–iteration combined with subsumption techniques is sound and complete for all operators which are monotonous w.r.t. the subsumption ordering.

Fixpoint computation dealing with rapidly increasing amounts of disjunctive facts is rather intricate, because of the problems of finding resolvable atoms in the facts and of subsumption elimination in fact sets. The disjunctive consequence operator T_P^S can be efficiently implemented based on a compact data structure of so–called clause trees for disjunctive facts. This will speed up the combinatorial search in a T_P^S–hyperresolution step. The necessary operations for subsumption elimination during the Δ–iteration process can be performed efficiently based on the clause trees.

3.1 The Tree–Based Disjunctive Consequence Operator

The disjunctive consequence operator T_P^S applied to a Herbrand state S searches for all disjunctive facts $B_1 \vee C_1, \ldots, B_n \vee C_n$ in S that can be hyperresolved with a ground instance $C' :\!- B_1, \ldots, B_n$ of a rule $r \in P$, deriving the smallest factor of $C' \vee C_1 \vee \ldots \vee C_n$.

Example 3.1 *Given the disjunctive logic program P with the transitive closure rule*

$$p(X,Y) :\!- p(X,Z), p(Z,Y)$$

and the Herbrand state

$$S = \{ \quad p(a,b) \vee p(f,g),\ p(a,b) \vee p(f,h),$$
$$p(a,b) \vee p(a,d) \vee p(a,c) \vee p(a,\mathsf{1}),\ \ldots\ ,\ p(a,b) \vee p(a,d) \vee p(a,c) \vee p(a,\mathsf{m}),$$
$$p(b,c) \vee p(g,\mathsf{1}),\ \ldots\ ,\ p(b,c) \vee p(g,\mathsf{n}) \quad \}$$

From each of the facts $p(a,b) \vee p(a,d) \vee p(a,c) \vee p(a,\mathsf{i})$, $\mathsf{1} \leq \mathsf{i} \leq \mathsf{m}$, *and each of the facts* $p(b,c) \vee p(g,\mathsf{j}), \mathsf{1} \leq \mathsf{j} \leq \mathsf{n}$, *the* T_P^S*–operator derives* $n \cdot m$ *new facts*

$$p(a,d) \vee p(a,c) \vee p(a,\mathsf{i}) \vee p(g,\mathsf{j})$$

using the same ground instance of the transitive closure rule $p(a,c) :- p(a,b), p(b,c)$

In the following disjunctive facts will be considered as sets of atoms.
Assume for each body atom B_i, $1 \leq i \leq n$, facts $F_i^j \in S$, $1 \leq j \leq k_i$, with common intersection F_i such that $B_i \in F_i$. We will try to store F_i just once with pointers to the k_i difference sets $F_i^j \setminus F_i$. Thus, once we have found that $B_i \in F_i$, we know that each F_i^j has the form $B_i \vee C_i^j$, where $C_i^j = F_i^j \setminus \{B_i\}$. Therefore the F_i^j can be used for $\prod_{i=1}^n k_i$ hyperresolutions producing facts that all have the common subclause $F_1' \vee \ldots \vee F_n'$, where $F_i = F_i' \vee B_i$. In the example above we had $B_1 = p(a,b)$, $B_2 = p(b,c)$ and $k_1 = m$, $k_2 = n$.
As suggested, repeated application of the T_P^S–operator tends to rapidly increase the amount of intersections between derived disjunctive facts.

The **state overlap ratio** $o_S(S)$ is given by the ratio of the sums of the cardinalities of the clauses in S and the total number of atoms in S:

$$o_S(S) := \frac{\sum_{C \in S} |C|}{|\bigcup_{C \in S} C|}.$$

When searching S for facts during resolution, each atom in S will be looked at $o_S(S)$ times on average. In the above example we had $o_S(S) = (\ 4 + 4 \cdot m + 2 \cdot n\ )/(\ 6 + n + m\ )$, which converges to 3 for large $n = m$.

The **hasse diagram** $\mathcal{H}_{S'}$ for the superset $S' = \{\ C_1 \cap \ldots \cap C_k \mid k \in I\!N_+, C_i \in S, 1 \leq i \leq k\ \}$ of a finite Herbrand state S contains all the mentioned intersection information. But the cardinality $|S'|$ can be of exponential size $2^{|S|}$. E.g. for the Herbrand state $S = \{C_1, \ldots, C_n\}$ containing the clauses $C_i = p_1 \vee \ldots \vee p_{i-1} \vee p_{i+1} \vee \ldots \vee p_n$ the set S' contains all the subsets of $\{p_1, \ldots, p_n\}$ except the full one. Thus, we are looking for a suitable small subset S'' of S' ($S \subseteq S'' \subseteq S'$) that captures a suitable subset of common intersections of clauses in S. For every resolution step the whole data structure has to be searched for occurrences of B_i. This suggests to reduce the number of edges as well as the number of nodes.

Example 3.2 *Given the Herbrand state* $S = \{\ o \vee p \vee t,\ o \vee q \vee t,\ p \vee q\ \}$*. The hasse diagram* $\mathcal{H}_{S'}$ *for the superset* $S' = S \cup \{o \vee t,\ p,\ q\ \}$ *containing all intersections is given by figure 1, as well as two possible subtrees* $\mathcal{T}_1$ *and* $\mathcal{T}_2$*, containing all the leaf nodes of* $\mathcal{H}_{S'}$*.*

A tree representation of Herbrand states will turn out to be a reasonable choice for supporting deduction efficiently.

Definition 3.3 *Given a Herbrand state* S*. Let* $\mathcal{A}_S = \cup_{C \in S} C$ *denote all the atoms occurring in the clauses of* S*. A* **clause tree** $\mathcal{T} = (V, E)$ *with a node labeling* $l : V \longrightarrow 2^{\mathcal{A}_S}$ **represents** S*, iff*

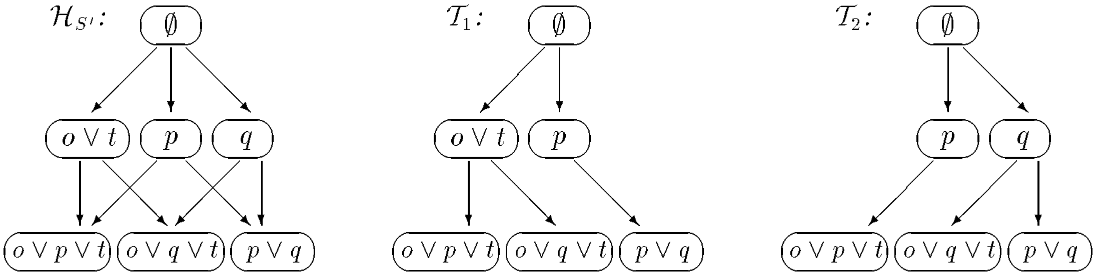

Figure 1: The Hasse–diagram $\mathcal{H}_{S'}$ and Steiner subtrees $\mathcal{T}_1$, $\mathcal{T}_2$

(i) *each node $v \in V$ represents a clause $clause(v)$ as follows: $clause(v)$ is the union of the sets $l(v')$ which label the nodes v' on the path from the root to v,*

(ii) *the clauses $clause(v)$ of the leaves $v \in V$ are in S, and every clause $C \in S$ is represented by exactly one leaf of the clause tree.*

For tree construction of $\mathcal{T} = (V, E)$ we use as heuristic the minimizing of the sum of the arc lengths:

$$weight(\mathcal{T}) = \sum_{e \in E} length(e)$$

where $length(e) = \mid clause(v_2) \setminus clause(v_1) \mid$, for $(v_1, v_2) \in E$, i.e. we are looking for a minimal Steiner tree $\mathcal{T}$, cf. [Vo 90], of the hasse diagram $\mathcal{H}_{S'}$. The price for this compact data structure is of course a lack of symmetry in the treatment of intersections compared to $\mathcal{H}_{S'}$. In example 3.2 clause tree $\mathcal{T}_1$ has the weight 6 and clause tree $\mathcal{T}_2$ has the weight 7. $\mathcal{T}_1$ is a minimal Steiner tree of the hasse diagram $\mathcal{H}_{S'}$.

Example 3.4 *For the transitive closure example 3.1 the Steiner tree $\mathcal{T}$ – given in figure 2 – representing the Herbrand state S is unique – which usually is not the case. The node identifiers are omitted, only the labels are displayed.*

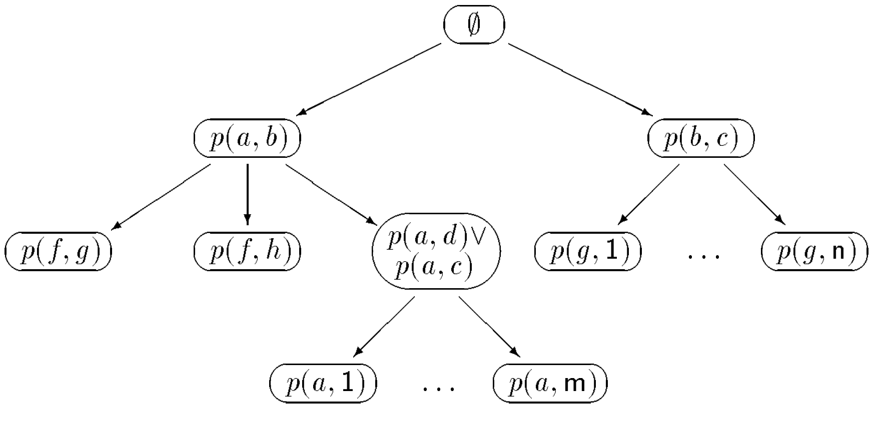

Figure 2: Tree representation by a minimal Steiner tree

Based on clause tree representations $\mathcal{T}$ of a Herbrand state S the T_P^S–operator is evaluated in two steps. The first one, tree search, determines nodes in the tree, such that there is a ground instance of a rule from P whose body atoms are contained in the tree nodes. The second one, tree resolution, resolves the clauses for the leaf nodes in the subtrees of the determined nodes w.r.t the ground instance of the rule found in tree search.

tree search
: search for all disjunctive facts $B_1 \vee C_1, \ldots, B_n \vee C_n$ in the labeled tree $\mathcal{T}$, such that there is a ground instance $C' \text{ :- } B_1, \ldots, B_n$ of a rule $r \in P$ as follows: for each rule r given by $p_1(X_1) \vee \ldots \vee p_m(X_m) \text{ :- } q_1(Y_1) \wedge \ldots \wedge q_n(Y_n)$ successively look for nodes v_i $(1 \leq i \leq n)$ in the clause tree, such that some $B_i \in \mathit{clause}(v_i)$ instantiates $q_i(Y_i)$, and no $B_i \in \mathit{clause}(v_i'')$ does so for all predecessors v_i'' of v_i in $\mathcal{T}$;

tree resolution
: let v_i^j $(1 \leq j \leq k_i)$ be the leaves in the subtree $\mathcal{T}_{v_i}$ of $\mathcal{T}$ rooted at v_i: for each successful search, i.e. a combination $v_1^{j_1}, v_2^{j_2}, \ldots, v_n^{j_n}$ of leaves in $\mathcal{T}_{v_1}, \mathcal{T}_{v_2}, \ldots, \mathcal{T}_{v_n}$, respectively, derive a new disjunctive fact C as the smallest factor of $C' \vee C_1 \vee \ldots \vee C_n$, where $C_i = \mathit{clause}(v_i^j) \setminus \{B_i\}$ (note, that the B_i choosen in tree search is contained in $\mathit{clause}(v_i) \subseteq \mathit{clause}(v_i^j)$);

Interleaved with the T_P^S–operator, the insertion of the portion of newly derived facts which are not subsumed by already known facts in the old clause tree can be performed:

tree update
: determine a new clause tree representation $\mathcal{T}$ with a labeling l for the updated set S of disjunctive facts by refining – cf. figure 3 – the clause tree $\mathcal{T}$ at node v_i, for some arbitrarily picked $i \in \{1, \ldots, n\}$: insert a node v_i' labeled by $l(v_i) \setminus \{B_i\}$ in the tree $\mathcal{T}$ (if not already present); replace the edge (v_i'', v_i) in $\mathcal{T}$ between v_i and its immediate predecessor v_i'' by two new edges (v_i'', v_i'), (v_i', v_i); generate an (optimal) representing subtree $\mathcal{T}_{v_i'}$ for the set of all $\prod_{i=1}^n k_i$ resolvents rooted at the node v_i' (correct, since $\mathit{clause}(v_i')$ is a subclause of each resolvent !).

The – relatively large – clause $D = C' \vee \mathit{clause}(v_1) \setminus \{B_1\} \vee \ldots \vee \mathit{clause}(v_n) \setminus \{B_n\}$ is a common subclause of all the $\prod_{i=1}^n k_i$ derived resolvents C. It yields exactly one node in the new subtree $\mathcal{T}_{v_i'}$. Only the difference clauses $C \setminus D$ have to be considered, when building $\mathcal{T}_{v_i'}$, which is a great gain in efficiency.

The **tree overlap ratio** $o_T(\mathcal{T})$ of a clause tree $\mathcal{T}$ is given by the ratio of the weight of $\mathcal{T}$ and the total number of atoms in S:

$$o_T(\mathcal{T}) := \frac{\mathit{weight}(\mathcal{T})}{|\bigcup_{C \in S} C|}.$$

When searching the clause tree $\mathcal{T}$ for facts C from S during resolution, each atom occuring in S will be looked at $o_T(\mathcal{T})$ times on average. Since

$$|\bigcup_{C \in S} C| \leq \mathit{weight}(\mathcal{T}) \leq \sum_{C \in S} |C|,$$

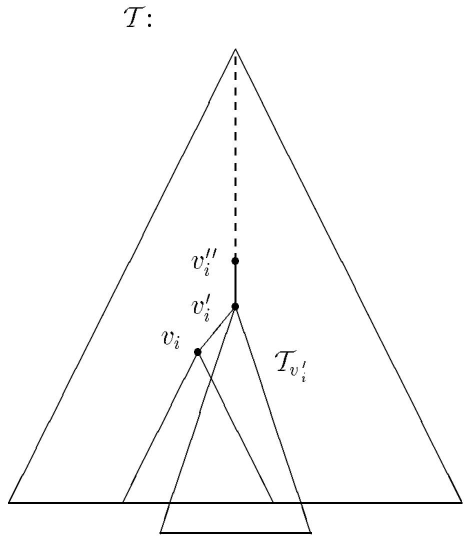

Figure 3: Tree–refinement

it holds

$$1 \leq o_T(\mathcal{T}) \leq o_S(S).$$

In the above example 3.4 we had $o_T(\mathcal{T}) = 1$, i.e. the tree overlap ratio is minimal.

We were looking for fast algorithms constructing tree representations $\mathcal{T}$ with small $o_T(\mathcal{T})$. Unfortunately, the minimal Steiner problem in graphs is $\mathcal{NP}$–complete, cf. [Vo 90]. So, we have implemented two heuristic methods:

(i) The first one is based on constructing minimal spanning trees in the hasse diagram $\mathcal{H}_{S'}$. These trees can be transformed easily to Steiner trees. In graph theory, this is considered a good heuristic for the general minimal Steiner tree problem.

(ii) But, a clause set S has much more structure that can be used for tree construction. The second technique forgets about the hasse diagram and builts the tree $\mathcal{T}$ inductively by a heuristic for the insertion of a clause C from S into an intermediate tree.

In our test runs, the achieved tree overlap ratios of the two methods were about equal. But the second method worked much faster. It took only about 10% of the run time. Thus, for larger Herbrand states containing more than 50 different atoms the first method was too slow and could no longer be used.

3.2 The Tree–Based Δ–Iteration Technique

Semi–naive or Δ–iteration was originally defined for definite logic programs. For extending it to the fixpoint computation of a disjunctive logic program P, we need the Δ–version of the disjunctive consequence operator T_P^S.

Definition 3.5 *Given a disjunctive logic program P and two Herbrand states D and ΔD of P. The disjunctive* Δ**–consequence operator** $T_P^S : 2^{\mathrm{DHB}_P} \times 2^{\mathrm{DHB}_P} \longrightarrow 2^{\mathrm{DHB}_P}$ *is defined as*

$$\begin{aligned} T_P^S(D, \Delta D) \;=\; \{\; C \in DHB_P \mid\; & C' :\!-\; B_1, \ldots, B_n \textit{ is in ground}(P), \\ & B_1 \vee C_1,\; \ldots,\; B_n \vee C_n \textit{ are in } D \cup \Delta D, \quad \textit{some } B_i \vee C_i \textit{ is in } \Delta D, \\ & C \textit{ is the smallest factor of } C' \vee C_1 \vee \ldots \vee C_n \;\}. \end{aligned}$$

The algorithm Δ–ITERATION–1 of figure 4 formulates the disjunctive Δ–iteration based on the Δ–version of T_P^S. When applied to the empty Herbrand state $S = \emptyset$, it yields the least fixpoint of T_P^S.

> **Algorithm** Δ–ITERATION–1 (P, S)
>
> { initialization }
>
> $\Delta S := S$;
>
> { iteration }
>
> **while** ($|\Delta S| > 0$)
>
> $\quad \Delta S \;:=\; can(\; prune(T_P^S(S, \Delta S))\;) \setminus_s S$;
>
> $\quad S \;:=\; can(\; S \cup \Delta S\;)$;
>
> **end**

Figure 4: Disjunctive Δ–iteration

The basic operations of Δ–ITERATION–1 are the following. From a computational point of view, T_P^S derives a multiset S' of disjunctive facts, which has to be pruned by removing duplicates. The time complexity of **pruning** is $n \log n$, where $n = |S'|$ is the number of facts in S'. Δ**–computation** w.r.t. subsumption is the computation of the set $S_1 \setminus_s S_2$ of all disjunctive facts in a Herbrand state S_1 which are not subsumed by a disjunctive fact in another Herbrand state S_2. It takes time $n_1 \cdot n_2$, where $n_i = |S_i|$. **Canonization** is the computation of all disjunctive facts in a Herbrand state S'' which are not properly subsumed by other – i.e. shorter – disjunctive facts in the same state. It takes time n^2, where $n = |S''|$.

Based on these considerations, we derive a more efficient version of the disjunctive Δ–iteration by rearranging the basic operations.

> **Algorithm** Δ–ITERATION–2 (P, S)
>
> { initialization }
>
> $\Delta S := S$;
>
> { iteration }
>
> **while** ($|\Delta S| > 0$)
>
> $\quad S_1 \;:=\; T_P^S(S, \Delta S)$;
>
> $\quad S_2 \;:=\; prune(S_1)$;
>
> $\quad S_3 \;:=\; S_2 \setminus_s S$;
>
> $\quad \Delta S \;:=\; can(S_3)$;
>
> $\quad S \;:=\; can(S \cup \Delta S)$;
>
> **end**

Figure 5: Improved disjunctive Δ–iteration

Experiments with Δ–ITERATION–2, cf. tables 2 and 3 revealed, that the comparatively cheap pruning operation reduces the multiset S_1 of derived facts to about 30% of its size. Only to the resulting facts Δ–computation has to be applied yielding a further reduction to about 1 % of the size of S_1 in time $O(|S_2| \cdot |S|)$, where normally S is much smaller than S_2. Finally, the most expensive operation *can* is delayed as far as possible. It is applied to the smallest Herbrand state S_3 and reduces it to about 0.5% of the size of S_1.

Based on a tree representation $\mathcal{T}$ of the intermediate Herbrand states S, it can be determined very efficiently if a Herbrand state S subsumes a given disjunctive fact, cf. the following PROLOG–rules:

```
tree_subsumes([Root|Subtrees],D_Fact) :-
    subsumes(Root,D_Fact),
    member(Subtree,Subtrees),
    tree_subsumes(Subtree,D_Fact).
tree_subsumes([Root],D_Fact) :-
    subset(Root,D_Fact).
```

A clause tree is given by a clause *Root* denoting its root and a list *Subtrees* of zero or more subtrees of *Root*. A clause C_1 *subsumes* another clause C_2 if C_1 is a subclause of C_2. Thus, *tree_subsumes* essentially searches for a complete path in the tree representation from the root to a leaf, such that all labeling clauses of the nodes on the path are subclauses of the investigated clause. Then the clause represented by the determined leaf is in S and subsumes the investigated disjunctive fact.

3.3 Experimental Results

We have compared PROLOG–implementations of the basic version of algorithm Δ–ITERATION–2 and a tree–based version of algorithm Δ–ITERATION–2 using clause trees for resolution and subsumption. We achieved very encouraging results for the path example, even without incorporating all possible optimization techniques. I.e. tree update was done by rooting a representing clause tree for the derived new facts at the root of the old clause tree, instead of refining the old tree or merging the old and the derived clause tree, which might yield a tree with a smaller weight.

Example 3.6 *Table 1 shows the ratio of the evaluation times for the fixpoint computations with the basic version (t_1) and the tree–based version (t_2) of* Δ–ITERATION–2 *for different initial Herbrand states S in the running transitive closure example. We have measured the state overlap ratios $o_S(S)$, the tree overlap ratios $o_T(\mathcal{T})$ and the compactification ratios $o_S(S)/o_T(\mathcal{T})$ at the beginning (superscript i), at the end (superscript f) and at the time where the compactification ratio $o_S(S)/o_T(\mathcal{T})$ was maximal (superscript m).*

$o_S^{(i)}$	$o_S^{(i)}/o_T^{(i)}$	$o_S^{(f)}$	$o_S^{(f)}/o_T^{(f)}$	$o_S^{(m)}$	$o_S^{(m)}/o_T^{(m)}$	t_1/t_2
1.54	1.54	18.26	2.50	23.53	2.66	1.69
1.73	1.73	22.23	2.72	34.55	2.98	2.14
1.88	1.88	25.64	2.87	44.52	3.23	2.65
2.00	2.00	28.68	3.00	54.50	3.41	3.49

Table 1: Comparison of basic and tree–based Δ–ITERATION–2

For Herbrand states with many overlapping facts, i.e. large state overlap ratios $o_S(S)$, the tree–based version becomes more and more efficient and superior to the basic version of Δ–ITERATION–2.

For disjunctive rules with more than two body atoms we have achieved a speed–up for the tree–based version of more than 10. When compared to the initial version Δ–ITERATION–1 – where the canonization operation is performed before the Δ–computation by $\setminus_s$ – the speed–up is even much higher.

We have also analysed the tree–based version of Δ–ITERATION–2 for the disjunctive transitive closure example by measuring the following: the times for the basic operations of the iteration steps, the numbers of tuples after the different operators, the state and tree overlap ratios, the compactification ratios after the performed iteration steps.

Example 3.7 *The tree–based version of* Δ–ITERATION*–2 for the disjunctive transitive closure example for a Herbrand state with* 11 *facts yielded the results given by the tables 2, 3 and 4.*

step	1	2	3	4	
T_P^S	0.26	2.69	28.71	11.22	42%
prune	0.01	0.53	11.46	4.13	12.6%
$\setminus_s$	0.05	1.21	19.68	11.33	32%
can	0.27	2.45	1.13	0.01	3.5%
can	0.55	5.17	4.19	0.02	9.9%

Table 2: Time for tree–based Δ–ITERATION–2

step	1	2	3	4	
T_P^S	100	2.209	33.406	12.667	100%
prune	50	891	8.612	5.235	31%
$\setminus_s$	44	208	151	0	0.9%
can	44	132	47	0	0.5%
can	0	3	30	0	14%

Table 3: Numbers of tuples after the different operators

step	0	1	2	3	4
o_S	1.75	8.36	28.92	31.08	31.08
o_T	1.00	3.64	10.12	10.77	10.77
o_S/o_T	1.75	2.30	2.86	2.89	2.89

Table 4: State and tree overlap ratios, compactification ratios

Most of the run time – 42% *– is spent for the disjunctive* T_P^S*–operator. Pruning also takes a considerable amount –* 13% *– of the run time, since it has to reduce a huge number (about* 50.000*) of tuples to about* 30%*.* Δ*–computation, although it has been speeded up by using trees, still is very expensive and takes a further* 32% *of the run time. The canonization operations are much cheaper than in the case where they are applied before the* Δ*–computation.*

4 Summary and Outlook

Disjunctive, normal logic programs P can be evaluated by repeated computation of minimal model states w.r.t. certain disjunctive transformations of P to compute the generalized disjunctive well–founded semantics.
Tree–based Δ–iteration techniques can be used for computing the extended generalized closed world assumption (EGCWA) of a disjunctive logic program as defined in [LoMiRa 92]. The support for negation sets for conjunctive facts can be efficiently derived by an iteration from the minimal model state $\mathcal{MS}_P$. Furthermore, the positive and the negative consequence operator $\mathcal{T}_S^D$ and $\mathcal{F}_S^D$, respectively, used for defining the generalized disjunctive well–founded semantics can be speeded up by tree–based Δ–iteration.

We have developed a tree–based evaluation of disjunctive normal logic programs under the generalized disjunctive well–founded semantics. It has been implemented as a meta–interpreter in PROLOG. The basic operators are the disjunctive consequence operator T_P^S, the Δ–operator $\setminus_s$ with subsumption and the canonization operator *can* for Herbrand states. Tree–based versions of these operators are used to built a Δ–iteration technique for the computation of the least fixpoint of a disjunctive logic program. Based on this, the generalized disjunctive well–founded semantics of disjunctive normal logic programs can be computed.

Tree–resolution has been analysed for the frequently occurring definite transitive closure rules with disjunctive facts. For general disjunctive logic programs involving disjunctive rules the method is applicable as well, and yields even more efficient evaluation. The state overlap ratio gets even higher, since the common subclause of the resolvents contains the whole disjunctive rule head.

Incorporating aspects of other data structures like TRIES, cf. [AhHoUl 83], for a bit vector representation of Herbrand states, hash tables and inverted lists is currently investigated.

References

[AbWiPa 93] *A. Abdelmoty, M. Williams, N. Paton:* Deduction and Deductive Databases for Geographic Data Handling, Proc. Intl. Symposium on Advances in Spatial Databases 1993 (SSD'93), pp. 443-464.

[AhHoUl 83] *A. Aho, J. Hopcroft, J. Ullman:* Data Structures and Algorithms, Addison–Wesley, 1983.

[BaLoMi 89] *C. Baral, J. Lobo, J. Minker:* Generalized Well–Founded Semantics for Logic Programs, Proc. Intl. Conf. on Automated Deduction 1989, pp. 102-116.

[BaRa 86] *F. Bancilhon, R. Ramakrishnan:* An Amateur's Introduction to Recursive Query Processing Strategies, Proc. ACM SIGMOD 1986, pp. 16-52.

[CeGoTa 90] *S. Ceri, G. Gottlob, L. Tanca:* Logic Programming and Databases, Springer, 1990.

[CheWa 92] *W. Chen, D.S. Warren:* A Goal–Oriented Approach to Computing Well–Founded Semantics, Proc. Joint Intl. Conf. and Symp. on Logic Programming 1992, pp. 589-603.

[CheWa 93] *W. Chen, D.S. Warren:* Query Evaluation under the Well–Founded Semantics, Proc. ACM PODS 1993.

[Da 92] *S.K. Das:* Deductive Databases and Logic Programming, Addison–Wesley, 1992.

[Fu 91] *U. Fuhrbach:* Computing Answers for Disjunctive Logic Programs, Proc. ILPS Workshop on Disjunctive Logic Programs 1991.

[GeLi 88] *M. Gelfond, V. Lifschitz:* The Stable Model Semantics for Logic Programming, Proc. Intl. Conf. and Symp. on Logic Programming 1988, pp. 1070-1080.

[GüKiBa 87] *U. Güntzer, W. Kießling, R. Bayer:* On the Evaluation of Recursion in (Deductive) Database Systems by Efficient Differential Fixpoint Iteration, Proc. Intl. Conf. on Data Engineering 1987, pp. 120-129.

[KöKiThöGü 93] *G. Köstler, W. Kießling, H. Thöne, U. Güntzer:* The Differential Fixpoint Operator with Subsumption, Proc. Intl. Conf. on Deductive and Object–Oriented Databases 1993.

[Llo 87] *J.W. Lloyd:* Foundations of Logic Programming, Springer, second edition, 1987.

[LoMiRa 92] *J. Lobo, J. Minker, A. Rajasekar:* Foundations of Disjunctive Logic Programming, MIT Press, 1992.

[Pea 88] *J. Pearl:* Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, 1988.

[Prz 90] *T.C. Przymusinski*: Stationary Semantics for Disjunctive Logic Programs and Deductive Databases, Proc. North–American Conf. of Logic Programming, MIT Press, 1990, pp. 42-59.

[SeiAr 93] *D. Seipel, H. Argenton*: Evaluation Techniques for Disjunctive Logic Programs, Proc. Symposium on Operations Research 1993.

[Ul 88,89] *J.D. Ullman*: Principles of Database and Knowledge-Base Systems, Volume I,II, Computer Science Press, 1988/89.

[VaGeRoSchli 91] *A. Van Gelder, K.A. Ross, J.S. Schlipf*: The Well–Founded Semantics for General Logic Programs, JACM, vol. 38(3), 1991, pp. 620-650.

[Vo 90] *S. Voß*: Steiner–Probleme in Graphen, Anton Hain, 1990.