Solving Cardinality Constraints in (Constraint) Logic Programming

Dietmar Seipel[1] and Ulrich Geske[2]

[1] University of Würzburg, Dept. of Computer Science
Am Hubland, D – 97074 Würzburg, Germany
seipel@informatik.uni-wuerzburg.de

[2] GMD–FIRST Berlin
Kekuléstraße 7, D – 12489 Berlin, Germany
geske@first.gmd.de

Abstract

We investigate *cardinality constraints* of the form $M \hookrightarrow_{\Theta} K$, where M is a set and Θ is one of the comparison operators "$=$, $\leq$", or "$\geq$": a model of such a constraint is required to contain "exactly, at most", or "at least", respectively, K elements of M.

Applications dealing with cardinality constraints may for instance arise in assignment problems, such as *course planning*, or in *games*: E.g. there may be rules like "every student has to take 3 of the courses of a given set M".

We present a *calculus* represented by a definite logic program, which allows for very efficiently reasoning with sets of cardinality constraints. We have implemented the calculus in Prolog, and we have compared it to other methods that we have encoded in constraint logic programming and in disjunctive logic programming.

Keywords. cardinality constraints, (constraint) logic programming

1 Introduction

Practical applications of deductive databases [1] often require the use of extended features like *disjunctive information*, *aggregation operators*, *default negation*, *computation of cheapest models*; several extensions of deductive databases have been proposed for these reasons. Applications dealing with *cardinality constraints* may for instance arise in assignment problems, such as course planning, or in games. Also cardinality constraints for *binary relationship types* in *entity–relationship models* can be expressed easily using the concept of cardinality constraints.

Constraint logic programming (CLP) [6] offers very efficient methods for solving problems involving cardinality constraints. Cardinality constraints were introduced in CLP by Van Hentenryck and Deville [2], where, syntactically, a cardinality constraint is a formula $\#(t_l, t_u, [\varphi_1, \ldots, \varphi_n])$ with terms t_l, t_u and formulas φ_i. Its declarative semantics is as follows: it evaluates to true under an interpretation I (including a variable assignment), iff k formulas φ_i are true, where $I(t_l) \leq k \leq I(t_u)$ and $I(t_l), I(t_u)$ are natural numbers. Such a construct is extremly usefull for avoiding standard backtracking, which occurs for disjunctive constraints. Cardinality constraints avoid nondeterminism by delaying the decision for one of the disjuncts. In [2] results of benchmark tests are given which demonstrate the efficiency of cardinality constraints compared to disjunctive–constraint formulas, especially for larger problems.

Research in disjunctive logic programming let to methods for dealing with disjunctive knowledge. Moreover, if cardinality constraints have to be mixed with other formalisms for representing incomplete knowldege, then it might even be necessary to embed them into such a richer framework. In the disjunctive logic programming paradigm we could represent generic incomplete information such as "every student has to finish two small labs or one big lab, except if he enrolled himself before a certain deadline".

In deductive databases, reasoning with the well–known consequence operator $\mathcal{T}_\mathcal{P}$ of van Emden and Kowalski (cf. [4]) is *sound* and *complete* for deriving all atomary consequences of a definite logic program $\mathcal{P}$; it is not necessary to derive other formulas, such as disjunctions of literals, as intermediate results. In [10] we have shown that cardinality constraints can be derived based on the *disjunctive* consequence operator $\mathcal{T}^s_\mathcal{P}$ of Minker and Rajasekar (cf. [5], [8]), which generalizes $\mathcal{T}_\mathcal{P}$. In this paper we would like to investigate whether this approach can support the use of cardinality constraints in CLP. The expectation is, that this calculus has properties which are useful in special cases and which may be comparable or more efficient for certain classes of problems.

The rest of the paper is organized as follows. In Section 2 we define the theoretical background of cardinality constraints. In Section 3 we are dealing with different calculi for reasoning with cardinality constraints. Some experimental results obtained for reasoning with cardinality constraints are reported in Section 4. Finally, Section 5 presents some conclusions.

2 Basic Definitions and Notations

In this section we will define the concept of cardinality constraints. Cardinality constraints have also been considered by Niemelä and Simmons (cf. [7]), who have extended their system Smodels to cope with cardinality constraints.

Syntax of Cardinality Constraints

Given a universe $\mathcal{A}t$ of atoms, a subset $M \subseteq \mathcal{A}t$, and natural numbers $K, K_1, K_2 \in \langle\, 0, |M|\, \rangle$.[1]

[1] By $I\!N_+$ we denote the set $\{\, 1, 2, 3, \ldots \}$ of positive natural numbers, whereas $I\!N_0$ denotes the set $\{\, 0, 1, 2, \ldots \}$ of all natural numbers. $\langle\, n, m\, \rangle$ denotes the interval $\{\, n, n+1, \ldots, m \,\}$ of natural numbers.

1. A *cardinality constraint* is given by $M \hookrightarrow_{\Theta} K$, where $\Theta \in \{=, \leq, \geq\}$ is a comparison relation on $\mathbb{N}_0$.
2. An *interval–cardinality constraint* is given by $M \hookrightarrow_{\in} \langle K_1, K_2 \rangle$, where $\langle K_1, K_2 \rangle$ is an interval of natural numbers.

E.g., $\{a, b, c\} \hookrightarrow_{=} 2$ is a cardinality constraint.

Semantics of Cardinality Constraints

An interpretation is a set $I \subseteq \mathcal{A}t$ of ground atoms. I satisfies a cardinality constraint $c = M \hookrightarrow_{\Theta} K$, for short $I \models c$, if $|M \cap I|\, \Theta\, K$. I.e., c imposes a condition on the cardinality of the selected subset $M \cap I$ of M. We further define $M \hookrightarrow_{\in} \langle K_1, K_2 \rangle$ by

$$I \models M \hookrightarrow_{\in} \langle K_1, K_2 \rangle \text{ if } I \models M \hookrightarrow_{\geq} K_1 \text{ and } I \models M \hookrightarrow_{\leq} K_2.$$

Then it holds for instance $M \hookrightarrow_{=} K \iff M \hookrightarrow_{\in} \langle K, K \rangle$. Given a set $\mathcal{C}$ of cardinality constraints: I is a *model* of $\mathcal{C}$, for short $I \models \mathcal{C}$, if I satisfies all constraints $c \in \mathcal{C}$. $\mathcal{C}$ implies a single cardinality constraint c, $\mathcal{C} \models c$, if for all models I of $\mathcal{C}$ it holds $I \models c$. The set of all cardinality constraints that are implied by a set $\mathcal{C}$ of cardinality constraints is denoted by $\mathcal{C}^+$.

Definite Consequences of Cardinality Constraints

For a set $\mathcal{C}$ of cardinality constraints we define $\textsc{Def}(\mathcal{C}) = \langle T, F \rangle$ as

$$T = \{ A \in M \mid \text{there exists } M \hookrightarrow_{\geq} K \in \mathcal{C} \text{ such that } K = |M| \},$$
$$F = \{ A \in M \mid \text{there exists } M \hookrightarrow_{\leq} K \in \mathcal{C} \text{ such that } K = 0 \}.$$

The *definite consequences* of $\mathcal{C}$ are defined by $\textsc{Def}(\mathcal{C}^+) = \langle T^+, F^+ \rangle$: it can be shown that T^+ consists of all atoms $A \in \mathcal{A}t$ that are contained in all models I of $\mathcal{C}$, and F^+ consists of all atoms $A \in \mathcal{A}t$ that are contained in none of the models I of $\mathcal{C}$. E.g., for the set $\mathcal{C} = \{ c_1, c_2 \}$ of cardinality constraints, where $c_1 = \{a, b, c\} \hookrightarrow_{=} 2$ and $c_2 = \{b\} \hookrightarrow_{=} 0$, we get $T^+ = \{a, c\}$, since $\mathcal{C} \models \{a, c\} \hookrightarrow_{\geq} 2$, and $F^+ = \{b\}$, since $\mathcal{C} \models \{b\} \hookrightarrow_{\leq} 0$.

Non–Ground Cardinality Constraints

As a generalization, we could consider *non–ground cardinality constraints* $M \hookrightarrow_{\theta} K$ as well, where M can be a set of non–ground atoms with variable symbols. In that case, $M \hookrightarrow_{\theta} K$ must hold for all ground instances of M.

Non–ground cardinality constraints are for instance used in the *course planning example*: a course plan is given by a set I of atoms $e(St, Co, Se)$, which state that a student St is enrolled in a course Co in a given semester Se. There could be constraints such as "every student must be enrolled in at least 4 and at most 7 courses per semster":

$$\{ e(St, c_1, Se), \ldots, e(St, c_n, Se) \} \hookrightarrow_{\in} \langle 4, 7 \rangle,$$

where $\{c_1, \ldots, c_n\}$ is the set of all possible courses.

Computational Complexity

The computational complexity of the *satisfiability problem* for sets $\mathcal{C}$ of cardinality constraints of the from $M \hookrightarrow_= K$ over ground atoms is in $\mathcal{NP}$ (since for a given interpretation I of $\mathcal{C}$, the problem of testing whether $I \models \mathcal{C}$ is polynomial). As a consequence of this and a recent result of Kaye [3], the satisfiability problem is also $\mathcal{NP}$–complete.

The Mine Sweeper Example

The well–known game Mine Sweeper takes place on an $N \times N$ board, where mines are located on some of the fields. The goal of the player is to investigate the fields of the board, such that he never investigates a field with a mine. Each time a field is investigated, the system informs the player about the number of mines on the neighbour fields. Of course, the intial investigations cannot rely on any knowledge, but after a few investigations in many cases it becomes possible to infer that some neighbour fields do not contain a mine, or that other fields must contain a mine.

Consider the following 6×6–board, which was obtained by investigating the upper left corner with the coordinates $(1, 1)$. Since the neighbours of $(1, 1)$ do not contain any mines, they have been investigated as well, etc., until finally a border line with non–zero fields was reached, cf. Figure 1. This board gives

Initial board:

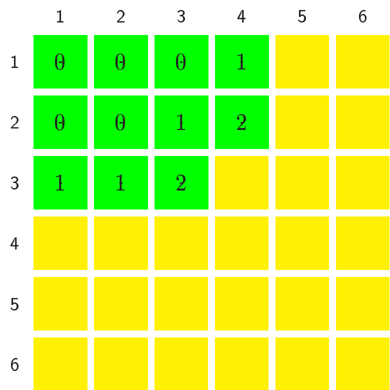

Fig. 1. Mine Sweeper Board after investigating the upper left corner $(1, 1)$

rise to the set $\mathcal{C} = \{ c_1, \ldots, c_6 \}$ of cardinality constraints, which are derived by scanning the 6 fields that are marked by non–zero numbers from left to right (and looking at their neighbours). E.g., since the field $(3, 3)$ is marked by "2", two of its non–investigated neighbours, namely $(2, 4)$, $(3, 4)$, $(4, 4)$, and $(4, 3)$, must contain a mine, which is modelled by c_3:

$$\begin{aligned}
(1,3) &\mapsto\ c_1 = \{(1,4),(2,4)\} \hookrightarrow_{=} 1, \\
(2,3) &\mapsto\ c_2 = \{(1,4),(2,4),(3,4)\} \hookrightarrow_{=} 1, \\
(3,3) &\mapsto\ c_3 = \{(2,4),(3,4),(4,4),(4,3)\} \hookrightarrow_{=} 2, \\
(3,2) &\mapsto\ c_4 = \{(4,3)\} \hookrightarrow_{=} 1, \\
(4,2) &\mapsto\ c_5 = \{(4,3),(5,3),(5,2),(5,1)\} \hookrightarrow_{=} 2, \\
(4,1) &\mapsto\ c_6 = \{(5,2),(5,1)\} \hookrightarrow_{=} 1.
\end{aligned}$$

3 Reasoning with Cardinality Constraints

For reasoning with cardinality constraints we will compare techniques from (constraint) logic programming.

3.1 Cardinality Constraints in Constraint Logic Programming

Cardinality constraints can be represented in CLP by assigning $\{0, 1\}$–variables X_A to the atoms $A \in \mathcal{A}t$. E.g., the cardinality constraint

$$c_3 = \{(2,4),(3,4),(4,4),(4,3)\} \hookrightarrow_{=} 2$$

from the Mine Sweeper example in the previous section can be represented by writing down all the cases which can arise:

$$\begin{array}{l}
(\ (\ X_{(2,4)} \#=\ 1,\ X_{(3,4)} \#=\ 1,\ X_{(4,4)} \#=\ 0,\ X_{(4,3)} \#=\ 0\ ) \\
;\ (\ X_{(2,4)} \#=\ 1,\ X_{(3,4)} \#=\ 0,\ X_{(4,4)} \#=\ 1,\ X_{(4,3)} \#=\ 0\ ) \\
;\ (\ X_{(2,4)} \#=\ 1,\ X_{(3,4)} \#=\ 0,\ X_{(4,4)} \#=\ 0,\ X_{(4,3)} \#=\ 1\ ) \\
;\ (\ X_{(2,4)} \#=\ 0,\ X_{(3,4)} \#=\ 1,\ X_{(4,4)} \#=\ 1,\ X_{(4,3)} \#=\ 0\ ) \\
;\ (\ X_{(2,4)} \#=\ 0,\ X_{(3,4)} \#=\ 1,\ X_{(4,4)} \#=\ 0,\ X_{(4,3)} \#=\ 1\ ) \\
;\ (\ X_{(2,4)} \#=\ 0,\ X_{(3,4)} \#=\ 0,\ X_{(4,4)} \#=\ 1,\ X_{(4,3)} \#=\ 1\ )\ ).
\end{array}$$

Most CLP systems contain a special construct, a so–called *global constraint*, for facilitating the encoding of such relations. For instance in the CHIP system the global constraint among/5 may be used for expressing cardinality constraints:

$$\mathsf{among}(\ [K_1, K_2, L],\ [X_1, X_2, \ldots, X_n],\ A_3,\ A_4,\ A_5\ ).$$

It expresses conditions on values of sets and subsets of specified domain variables $X_1, X_2, \ldots, X_n$. The first argument states that a value from the list A_4 of integers (in increasing order) occurs at least K_1 times and at most K_2 times in any subset of length L of the set $\{X_1, X_2, \ldots, X_n\}$. The arguments A_3 (which is a list of constants) and A_5 (which denotes the *search heuristic*) are not important for our considerations. E.g., the cardinality constraint c_3 given above can be encoded by the global constraint

$$\mathsf{among}(\,[2,2,4],\ [X_{(2,4)}, X_{(3,4)}, X_{(4,4)}, X_{(4,3)}],\ [0,0,0,0],\ [1],\ \mathsf{all}\,).$$

Given the different solutions to a set $\mathcal{C}$ of cardinality constraints, then the definite consequences in T^+ and F^+ are given by all atoms $A \in \mathcal{A}t$ such that $X_A = 1$ and $X_A = 0$, respectively, in all solutions. E.g., Table 1 shows the different solutions to the set $\mathcal{C} = \{\, c_1, \ldots, c_6 \,\}$ of cardinality constraints from the Mine Sweeper example in the previous section. It holds $T^+ = \{\, (4,3) \,\}$ and $F^+ = \{\, (3,4), (5,3) \,\}$, i.e., the field $(4,3)$ must contain a mine, and the fields $(3,4)$ and $(5,3)$ are free of mines.

$X_{(1,4)}$	$X_{(2,4)}$	$X_{(3,4)}$	$X_{(4,4)}$	$X_{(4,3)}$	$X_{(5,3)}$	$X_{(5,2)}$	$X_{(5,1)}$
1	0	0	1	1	0	1	0
1	0	0	1	1	0	0	1
0	1	0	0	1	0	1	0
0	1	0	0	1	0	0	1

Table 1. CLP Solutions for Mine Sweeper

3.2 Cardinality Constraints in Definite Logic Programming

In this section we will present a definite logic program $\mathcal{P}_{cc} = \{\, r_1, \ldots, r_6 \,\}$ representing a *calculus* for reasoning with cardinality constraints. The predicate symbols of $\mathcal{P}_{cc}$ are written in infix notation:

$$\begin{aligned} r_1 &= (M_2 \setminus M_1 \hookrightarrow_{=} K_2 - K_1) \leftarrow \\ &\qquad (M_1 \hookrightarrow_{=} K_1) \wedge (M_2 \hookrightarrow_{=} K_2) \wedge (M_1 \subseteq M_2), \\ r_2 &= (M_2 \setminus M_1 \hookrightarrow_{\geq} K_2 - K_1) \leftarrow \\ &\qquad (M_1 \hookrightarrow_{\geq} K_1) \wedge (M_2 \hookrightarrow_{\leq} K_2), \\ r_3 &= (M_2 \setminus M_1 \hookrightarrow_{\leq} K_2 - K_1) \leftarrow \\ &\qquad (M_1 \hookrightarrow_{\leq} K_1) \wedge (M_2 \hookrightarrow_{\geq} K_2) \wedge (M_1 \subseteq M_2). \end{aligned}$$

Cardinality constraints $M \hookrightarrow_{\theta} K$ for different comparison operators θ are related as follows:

$$\begin{aligned} r_4 &= (M \hookrightarrow_{=} K) \leftarrow (M \hookrightarrow_{\geq} K) \wedge (M \hookrightarrow_{\leq} K), \\ r_5 &= (M \hookrightarrow_{\geq} K) \leftarrow (M \hookrightarrow_{=} K), \\ r_6 &= (M \hookrightarrow_{\leq} K) \leftarrow (M \hookrightarrow_{=} K). \end{aligned}$$

Rule r_1 states the following: if K_1 elements are selected from a subset M_1 of M_2 and K_2 elements are selected from M_2, then – of course – $K_2 - K_1$ elements are selected from $M_2 \setminus M_1$. The rules r_2 and r_3 are variants of r_1 for reasoning with mixed comparison operators θ.

Soundness

It is easy to prove that $\mathcal{P}_{cc}$ is *sound* in the following sense: if a cardinality constraint c is derived from a given set $\mathcal{C}$ of cardinality constraints by repeated applications of rules in $\mathcal{P}_{cc}$, then $\mathcal{C} \models c$.

Reduction of Cardinality Constraints

Before reasoning with a set $\mathcal{C}$ of cardinality constraints using the calculus given by $\mathcal{P}_{cc}$ it is very helpful to *simplify* the set $\mathcal{C}$ first by removing from $\mathcal{C}$ all atoms $A \in T \cup F$, where $\textsc{Def}(\mathcal{C}) = \langle\, T, F\, \rangle$.

The *reduction* of $\mathcal{C}$ w.r.t. $\langle\, T, F\, \rangle$, where $T \subseteq \mathcal{A}t$ and $F \subseteq \mathcal{A}t$ are arbitrary sets of atoms, is defined as

$$\begin{aligned}\textsc{Red}_{\langle\, T,F\, \rangle}(\mathcal{C}) = \{\; M' \hookrightarrow_\theta K' \mid{} & \text{there exists } M \hookrightarrow_\theta K \in \mathcal{C} \\ & \text{such that } M' = M \setminus (T \cup F),\; K' = K - |M \cap T| \;\}.\end{aligned}$$

The reduction removes all atoms in $T \cup F$ from the cardinality constraints in $\mathcal{C}$ and adapts the values K to K'. Especially for $\langle\, T, F\, \rangle = \textsc{Def}(\mathcal{C})$ we define $\textsc{Red}(\mathcal{C}) = \textsc{Red}_{\langle\, T,F\, \rangle}(\mathcal{C})$.

It can be shown that reducing w.r.t. definite consequences yields implied cardinality constraints: i.e., $\textsc{Red}(\mathcal{C}) \subseteq \mathcal{C}^+$, and, more generally, $\textsc{Red}_{\langle\, T,F\, \rangle}(\mathcal{C}) \subseteq \mathcal{C}^+$, if $T \subseteq T^+$ and $F \subseteq F^+$, where $\textsc{Def}(\mathcal{C}^+) = \langle\, T^+, F^+\, \rangle$. E.g., for the set $\mathcal{C} = \{\, c_1, c_2, c_3, c_4\, \}$, where $c_1 = \{a, b, c\} \hookrightarrow_= 2$, $c_2 = \{b\} \hookrightarrow_= 0$ (as in a previous example), $c_3 = \{b, d, e\} \hookrightarrow_\leq 1$, and $c_4 = \{a, d, f\} \hookrightarrow_\geq 2$, we get $T^+ = \{a, c\}$, $F^+ = \{b\}$, and $\textsc{Red}_{\langle\, T^+,F^+\, \rangle}(\mathcal{C}) = \{\, \{\} \hookrightarrow_= 0,\, \{d, e\} \hookrightarrow_\leq 1,\, \{d, f\} \hookrightarrow_\geq 1\, \}$.

Fixpoint Iteration with $\mathcal{P}_{cc}$ and $\mathcal{C}$

The set $\textsc{Def}(\mathcal{C}^+) = \langle\, T^+, F^+\, \rangle$ of all definite conclusions of a given set of cardinality constraints can be *approximated* by alternatively applying the *definite consequence operator* $\mathcal{T}_{\mathcal{P}_{cc}}$ and the reduction operator: we start with $\mathcal{C}_1 = \textsc{Red}(\mathcal{C})$ and $\langle\, T_1, F_1\, \rangle = \textsc{Def}(\mathcal{C})$, and we iteratively compute $\mathcal{C}'_{n+1} = \mathcal{T}_{\mathcal{P}_{cc}}(\mathcal{C}_n)$ and

$$\mathcal{C}_{n+1} = \textsc{Red}(\mathcal{C}'_{n+1}),\; \langle\, T_{n+1}, F_{n+1}\, \rangle = \textsc{Def}(\mathcal{C}'_{n+1}).$$

Then, it holds $\cup_{n \in I\!N_+} T_n \subseteq T^+$ and $\cup_{n \in I\!N_+} F_n \subseteq F^+$. In general the sets T^+ and F^+ are only approximated: in Section 3.3 a set of cardinality constraints is given, where $\cup_{n \in I\!N_+} T_n \subsetneq T^+$.

However, when we applied the method to cardinality constraints derived from the Mine Sweeper problem, then it turned out that the sets T^+ and F^+ were obtained in almost all of the examples that we considered.

3.3 An Incompleteness Result

In this section, we will consider restricted calculi for reasoning about cardinality constraints of the form $M \hookrightarrow_= K$, which we will call *exactly–cardinality constraints*.

The question arises, whether there exists a definite logic program $\mathcal{P}^=_{cc}$ representing a *sound* and *complete* calculus for inferring exactly–cardinality constraints. This question has been answered in [10] by investigating the following generic set $\mathcal{C}_n$ of cardinality constraints:

$$\mathcal{C}_n = \{\, \{a_1, \ldots, a_n\} \hookrightarrow_= n-1\,\} \cup \{\, \{a_i, b_i, c\} \hookrightarrow_= 2 \,|\, 1 \leq i \leq n\,\},$$

where a_i, b_i, and c are pairwise different atoms, cf. Figure 2.

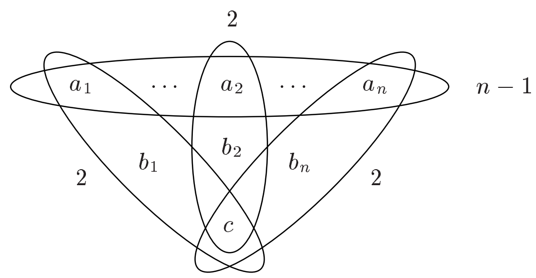

Fig. 2. Hypergraph for Cardinality Constraints

The following lemma has been proven in [10].

Lemma 3.1. *For the cardinality constraints*

$$\begin{aligned} cc_i &= \{a_i, b_i\} \hookrightarrow_= 1,\ 1 \leq i \leq n, \\ cc_{n+1} &= \{c\} \hookrightarrow_= 1, \\ cc_{n+2} &= \{b_1, \ldots, b_n\} \hookrightarrow_= 1, \end{aligned}$$

it holds $\mathcal{C}_n \models cc_i$, *for all* $i \in \langle\, 1, n+2 \,\rangle$.

However, for all strict subsets $\mathcal{C}' \subsetneq \mathcal{C}_n$ it holds $\mathcal{C}' \not\models cc_i$, for all $i \in \langle\, 1, n+2 \,\rangle$. This is a consequence of the following Theorem 3.1, which is a much stronger result that has been proven in [10].

A set $\mathcal{C}$ of exactly–cardinality constraints *covers* $c = M \hookrightarrow_= K$, for short $\mathcal{C} \rhd c$, if there exist constraints $M_i \hookrightarrow_= K_i \in \mathcal{C}$, $i \in \langle\, 1, k \,\rangle$, such that the sets M_i are pairwise disjoint, $M = \cup_{i=1}^{k} M_i$, and $K = \sum_{i=1}^{k} K_i$. It can be shown that $\mathcal{C} \rhd c$ always implies $\mathcal{C} \models c$. A cardinality constraint $M \hookrightarrow_= K$ is called *trivial*, if $M = \emptyset$ and $K = 0$.

Theorem 3.1. *For all strict subsets* $\mathcal{C}' \subsetneq \mathcal{C}_n$*, where* $n \geq 2$*, and all non–trivial cardinality constraints* $M \hookrightarrow_= K$ *it holds:*

$$(\mathcal{C}' \models M \hookrightarrow_= K) \Rightarrow (\mathcal{C}' \rhd M \hookrightarrow_= K).$$

Moreover it can be seen that $\mathcal{C}' \rhd M \hookrightarrow_= K$ implies that $M \hookrightarrow_= K \in \mathcal{C}'$ for non–trivial cardinality constraints.

Now we can answer the question of whether there exists a finite definite logic program $\mathcal{P}_{cc}^=$, such that the cardinality constraints cc_i, $i \in \langle\, 1, n+2 \,\rangle$, can be inferred from $\mathcal{C}_n$ using the calculus given by $\mathcal{P}_{cc}^=$. Every rule in $\mathcal{P}_{cc}^=$ contains only a finite number of atoms of the form $M \hookrightarrow_= K$ in its body. Thus, there exists a rule $r_{max} \in \mathcal{P}_{cc}^=$ with a maximal number n_{max} of atoms $M \hookrightarrow_= K$ in its body. As a consequence of Theorem 3.1, if $n > n_{max}$ and $\mathcal{P}_{cc}^=$ represents a sound calculus for reasoning with exactly–cardinality constraints, then $\mathcal{P}_{cc}^=$ is not able to derive the constraints cc_i, $i \in \langle\, 1, n+2 \,\rangle$, from the set $\mathcal{C}'$.

3.4 The Mine Sweeper Example

The calculus given by $\mathcal{P}_{cc}$ can be applied to the Mine Sweeper example of Section 2. Obviously, the cardinality constraint c_4 indicates that $(4,3)$ contains a mine. Further conclusions can be drawn using rule r_1. From c_1 and c_2 we can derive $c_7 = \{(3,4)\} \hookrightarrow_= 0$, since the set $\{(1,4),(2,4),(3,4)\}$ contains a single mine (cf. c_2) which must be located on one of the fields in the subset $\{(1,4),(2,4)\}$ (cf. c_1), i.e., $(3,4)$ cannot contain a mine. From c_5 and c_6 we can derive $c_8 = \{(4,3),(5,3)\} \hookrightarrow_= 1$, and from c_4 and c_8 we can derive $c_9 = \{(5,3)\} \hookrightarrow_= 0$, i.e., $(5,3)$ cannot contain a mine. Summarizing, we obtain $T_1 = \{(4,3)\}$, $F_1 = \emptyset$, and $T_2 = \emptyset$, $F_2 = \{(3,4)\}$, and $T_3 = \emptyset$, $F_3 = \{(5,3)\}$, and

$$\mathcal{C}_3 \doteq \{\, \{(1,4),(2,4)\} \hookrightarrow_= 1,\ \{(2,4),(4,4)\} \hookrightarrow_= 1,\ \{(5,2),(5,1)\} \hookrightarrow_= 1 \,\},$$

where trivial cardinality constraints (such as $\{(1,4)\} \hookrightarrow_\geq 0$) and weaker cardinality constraints (such as $\{(1,4),(2,4)\} \hookrightarrow_\geq 1$), which are also derived, are not shown. Now the iteration stops, since $\mathcal{C}_4 = \mathcal{C}_3$.

Investigating the fields $(3,4)$ and $(5,3)$ and marking $(4,3)$ as a mine might result in the intermediate board shown in Figure 3. Now the only conclusions can be drawn from the cardinality constraints obtained by looking at the fields $(3,3)$ and $(3,4)$:

$$\begin{aligned}
(3,3) &\mapsto\ c_{10} = \{(2,4),(4,4)\} \hookrightarrow_= 1,\\
(3,4) &\mapsto\ c_{11} = \{(2,4),(2,5),(3,5),(4,5),(4,4)\} \hookrightarrow_= 1.
\end{aligned}$$

Thus we can derive $c_{12} = \{(2,5),(3,5),(4,5)\} \hookrightarrow_= 0$, i.e., the fields $(2,5)$, $(3,5)$, and $(4,5)$ cannot contain a mine.

Investigating these fields might result in the final board, cf. Figure 3. At this point it turns out that no further definite conclusions can be drawn as to whether any of the non–investigated fields contains a mine or not. It is only possible to derive for instance the indefinite conclusion that either $(2,4)$ or $(4,4)$ contains a mine.

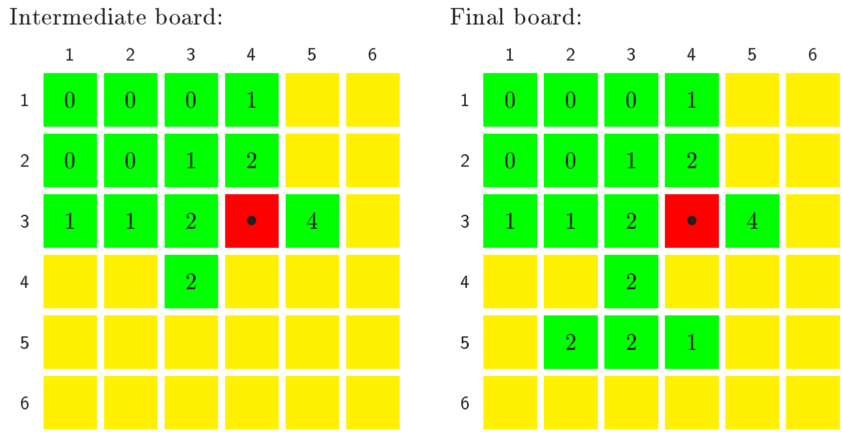

Fig. 3. Mine Sweeper Board after two further steps

3.5 Cardinality Constraints in Disjunctive Logic Programming

The definite logic program $\mathcal{P}_{cc}$ only derives cardinality constraints as intermediate results. Note, however, that other calculi which are deriving more general intermediate results could be able to derive all definite conclusions in T^+ and F^+. In [10] a transformation from a set $\mathcal{C}$ of cardinality constraints to a *positive–disjunctive logic program* $\mathcal{P}_\mathcal{C}$ without function symbols is given, such that the following holds for exactly–cardinality constraints of the form $\{A\} \hookrightarrow_= K$, where $K \in \{0, 1\}$:

- If $\mathcal{C} \models \{A\} \hookrightarrow_= 1$, then A can be derived from $\mathcal{P}_\mathcal{C}$.
- If $\mathcal{C} \models \{A\} \hookrightarrow_= 0$, then A can be derived from the dual program $\mathcal{P}^d_\mathcal{C}$.

This means that it is possible to compute the set $\textsc{Def}(\mathcal{C}^+) = \langle\, T^+, F^+ \,\rangle$ of *definite consequences* of $\mathcal{C}$ based on $\mathcal{P}_\mathcal{C}$.

4 Experimental Results

We have investigated a collection of test sets $\mathcal{C}$ of cardinality constraints of various sizes. In the following we report on sets that were derived from intermediate constellations of the game Mine Sweeper: each cardinality constraint $M \hookrightarrow_= K$ expresses how many of the non–investigated neighbours of a field on an $N \times N$ board contains a mine. Thus, the cardinalities of the sets M can range from 0 to 8, since each field has 8 neighbours. Larger sets of cardinality constraints can be obtained by enlarging the size N of the board.

We generated about 320 sets $\mathcal{C}$ of cardinality constraints (for boards of the size $N = 8$). In Figure 4 the average running times are given. On the x–axis we show the size

$$|\mathcal{C}|_n = \sum_{M \hookrightarrow_\theta K \in \mathcal{C}} |M|^n$$

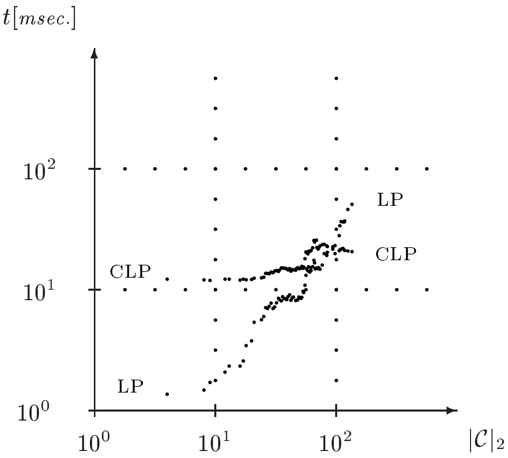

Fig. 4. Average Running Times for Reasoning with Cardinality Constraints

of the problems, and on the y–axis we show the running times in milliseconds. Our interest was to investigate, whether a problem that is hard to solve for one calculus is hard to solve for the other, too. It turned out that the computation time needed when working with $\mathcal{P}_{cc}$ (LP) was comparable to the time needed for computing with constraint logic programming (CLP); $\mathcal{P}_{cc}$ was faster for smaller examples, whereas CLP was faster for bigger examples. Thus, for smaller examples it is a good strategy first to apply $\mathcal{P}_{cc}$ and see if it returns any results, and then to apply CLP only in case that no results have been obtained. Only for 0.6% (i.e., 2 out of 320) of the test cases, CLP was able to derive definite conclusions, when $\mathcal{P}_{cc}$ did not return any conclusions.

In CLP we have encoded the problem by writing down all the alternative instantiations (Test 1) and by using the among/5 construct (Test 2). Surprisingly, for about 66% of the benchmark problems Test 1 runs faster than or as fast as Test 2. Of course this result is only valid for the problem size of the Mine Sweeper game and for the available implementation of among/5. To have an estimation of the influence of the number of cardinality constraints on the execution time, a much larger problem with 113 variables and 117 cardinality constraints was specified using the among/5 construct. The running time corresponded about linearly with the larger problem size compared to Test 2.

5 Conclusions

For reasoning with cardinality constraints the mixing of methods from *constraint logic programming*, which encode the constraints into *global constraints*, with a calculus for directly dealing with the constraints in logic programming proved itself to be very useful.

In [10] we have shown that cardinality constraints can also be solved using methods from *disjunctive logic programming*, which encode the constraints into disjunctive logic rules. This is very helpful for problems where cardinality constraints and incomplete, disjunctive knowledge are envolved.

In the future we plan to investigate the calculus and the methods from CLP for larger examples (e.g. cardinality constraints from larger Mine Sweeper boards) and also for non–ground cardinality constraints.

References

1. *S. Ceri, G. Gottlob, L. Tanca:* Logic Programming and Databases, Springer, 1990.
2. *P. Van Hentenryck, Y. Deville:* The Cardinality Operator: A new Logical Connective for Constraint Logic Programming, Proc. 8th Intl. Conference on Logic Programming 1991 (ICLP'91), MIT Press, 1991, pages 745–759.
3. *R. Kaye:* Minesweeper is NP–complete, Mathematical Intelligencer, 22(2), 2000, pages 9–15.
4. *J.W. Lloyd:* Foundations of Logic Programming, Second Edition, Springer, 1987.
5. *J. Lobo, J. Minker, A. Rajasekar:* Foundations of Disjunctive Logic Programming, MIT Press, 1992.
6. *K. Marriott, P. Stuckey:* Programming with Constraints – An Introduction, MIT Press, 1998.
7. *I. Niemelä, P. Simmons:* Extending the Smodels System with Cardinality and Weight Constraints, In Jack Minker (ed.): *Logic–Based Artificial Intelligence*, Kluwer, 2000, pages 491–522.
8. *D. Seipel, J. Minker, C. Ruiz*: Model Generation and State Generation for Disjunctive Logic Programs, *Journal of Logic Programming*, 32(1), 1997, pages 48–69.
9. *D. Seipel:* DisLog – A Disjunctive Deductive Database Prototype, Proc. 12th Workshop on Logic Programming (WLP'97), 1997, pages 136–143. DisLog is available at "http://www-info1.informatik.uni-wuerzburg.de/databases/DisLog".
10. *D. Seipel, U. Geske:* Cardinality Constraints in Disjunctive Deductive Databases, In *Workshop on Deductive Databases and Logic Programming (DDLP'2000)* at the International Conference on Applications of Prolog (INAP'2000), 2000.