Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Cutting Connections in Linear Connection Proofs

Bertram Fronhöfer

Institut für Informatik, TU München, D – 80290 München
fronhoef@informatik.tu-muenchen.de

Abstract

The paper contributes to the implementation of plan generation systems based on the Linear Connection Method. Analysing a previously published algorithm, we detected as one of the reasons for the explosion of its search space certain "cyclic rules"—Horn clause like expressions where the head literal reoccurs in the tail—which resulted from the way we transformed action specifications. We will present here an improved algorithm, which instead of using these cyclic rules, solves subgoals via insertion of actions into an already partially constructed plan. This improved algorithm also outperformed the old one when we ran it on some benchmarks.

1 Introduction

Goal-oriented plan generation is one of the classical topics of AI research. In its essence, a plan generation problem is given by a triple $(\mathbf{I}, \mathbf{A}, \mathbf{G})$: A *set of actions* $\mathbf{A}$ together with a description of a state of the world—the *initial situation* $\mathbf{I}$—and a (maybe partial) description of a desired future situation—the *goal* $\mathbf{G}$. A solution to such a plan generation problem—a *plan*—is a sequence of actions from $\mathbf{A}$, which when applied to the initial situation $\mathbf{I}$, generates a new situation in which the goal $\mathbf{G}$ is satisfied.

The task of plan generation can be viewed as an inference problem in the following way: We have to prove for a given plan generation problem $(\mathbf{I}, \mathbf{A}, \mathbf{G})$ a derived *specification theorem* which is roughly of the form: '$\mathbf{I}$ *and* $\mathbf{A}$ *imply* $\mathbf{G}$'. The exact form of the specification theorem will depend on the chosen logic and on the style of specifying or modelling actions and situations in this logic. The interest in generating plans automatically gives rise to the question of a suitable logic allowing for efficient automated proof search.

The first concrete proposal how to generate plans via theorem proving was made on the choice of classical first-order logic and resulted in the so-called *Situation Calculus* (see [11]), which immediately fell into disrepute due to bad practical performance in attempts to prove the respective specification theorems by use of automated theorem provers (see [8]). This failure of theorem proving was commonly attributed to what was called the Frame Problem: Lots of so-called Frame Axioms had to be specified, which increased tremendously the already inevitable exponential explosion of the search space. Since this problem is also inherent in other logics (e.g. modal logics) known in those days, the nonsuccess of Situation Calculus discredited the use of logic for plan generation in general, and henceforth planning systems were conceived without reference to a particular kind of logic[1].

This negative judgement about the suitability of logic for plan generation was challenged considerably in 1986 when W. Bibel proposed the *Linear Connection Method*—which produces so-called *Linear Connection Proofs*—as a new approach to plan generation (see [1] and Section 2). Since this approach worked without Frame Axioms it promised to overcome the shortcomings of the logic-oriented approaches to plan generation which were known till then. We also presented in [5] and [2] a proof search algorithm for Linear Connection Proofs, the so-called *Linear Backward Chaining* (LBC)-algorithm (see Section 2 for details), which performed quite competitively when compared with a real planning system (see [5] for details).

However, a closer analysis of the LBC-algorithm revealed the following unpleasant behaviour, which stems from the requirement to remention in the consequent of our action/implications those preconditions of the action which *survive* the action's application unaffectedly (see Section 2). This peculiarity of the way to specify actions with the Linear Connection Method is disastrous if we consider simple backward proof search algorithms. As we will show in Exam-

[1] Let us shortly mention that the rupture between logic and planning was mainly on deductive grounds, since for many planning systems efforts were made to define precise declarative semantics. A good example in this respect is the STRIPS approach for which semantics were given in [10], but STRIPS's connectives and a proof theory were never worked out.

ple 3 later, we are obliged to enter action/implications in extension steps at suchurementioned literals, thus giving rise, at least theoretically, to an additional search space explosion due to looping over these rementioned literals.

With the LBC-algorithm this becomes more apparent, because we transform action/implications into a set of so-called rules, which are special Horn clauses, and in case of a rementioned literal L we obtain a "cyclic rule", i.e. we get a Horn clause with head literal L, where L is also among the tail literals (see Example 2).

The attempt to avoid this additional growth of the search space led to another proof search algorithm—the so-called *Linear Insertion Planning* (LIP)-algorithm—which works without these cyclic rules. Experimental evaluation also showed that the LIP-algorithm generally outperforms the LBC-algorithm.

The plan of the paper is as follows:

In Section 2 we review the Linear Connection Method and its use for plan generation. We also recapitulate the LBC-algorithm.

In Section 3 we present the cyclic rules and show the rôle played by them with the LBC-algorithm by means the well-known Sussman anomaly.

In Section 4 we present the Linear Insertion Planning (LIP)-algorithm—an extension of the LBC-algorithm—works without these cyclic rules. In Section 5 we show the runtime behaviour of the LIP-algorithm with respect to the LBC-algorithm by means of some benchmark examples.

2 Linear connection proofs

As with Situation Calculus, a basic principle of the Linear Connection Method is to view plan generation as the task of proving that a goal situation can be deduced from an initial situation and from the formulae describing the actions. However, in contrast to Situation Calculus, where 'situation information' is encoded (as an additional argument) in every elementary proposition (literal) which may vary over time, with the Linear Connection Method elementary propositions are void of such 'situation information'.

Example 1 In order to present this approach, let us look at the following example of a plan generating proof, taken from [1], where a block b is moved from the top of a block a down on the table. It is specified in the following way: The *initial situation* will be given by the formula

$$Sit(s) \wedge T(a) \wedge O(b,a) \wedge C(b) \wedge E \qquad (\mathbf{I}_1)$$

which means, that we are in a situation s, denoted by $Sit(s)$, where a is on the table: $T(a)$, block b is on top of a : $O(b,a)$, the top of b is clear: $C(b)$ and the robot's hand is empty: E.

Our *action system* $\mathbf{A}_1$ is composed of two actions $\mathbf{A}_{11}$ and $\mathbf{A}_{12}$ which lift a block and put a block down respectively, and which are specified by the following formulae:

$$\forall w,x,y: \quad Sit(w) \wedge O(x,y) \wedge C(x) \wedge E \longrightarrow Sit(l(w,x,y)) \wedge H(x) \wedge C(y) \qquad (\mathbf{A}_{11})$$

$$\forall w',v: \quad Sit(w') \wedge H(v) \longrightarrow Sit(p(w',v)) \wedge C(v) \wedge T(v) \wedge E \qquad (\mathbf{A}_{12})$$

$(\mathbf{A}_{11})$ says: For any situation w, where a block x is on a block y : $O(x,y)$, the top of x is clear: $C(x)$ and the robot's hand is empty: E, there exists a situation $l(w,x,y)$ in which the robot holds block x in his hand: $H(x)$ and the top of block y is clear: $C(y)$.

$(\mathbf{A}_{12})$ says: For any situation w' where the robot holds block v : $H(v)$, there exists a situation $p(w',v)$ in which v is on the table: $T(v)$, nothing is on top of v: $C(v)$ and the robot's hand is empty: E.

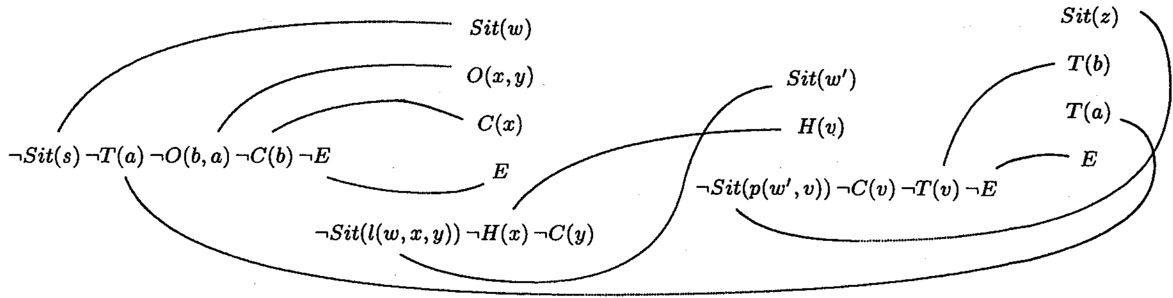
The *goal* is stated by the formula

$$\exists z: Sit(z) \wedge T(b) \wedge T(a) \wedge E \qquad (\mathbf{G}_1)$$

which says that we are asking for a situation z in which the blocks a and b are both on the table and the robot's hand is empty. With our approach, a plan which brings about the desired situation will be extracted from a proof of the goal formula from the given situation and the known actions, i.e. we must prove the specification theorem $T_{(\mathbf{I}_1,\mathbf{A}_1,\mathbf{G}_1)} :\equiv \mathbf{I}_1 \wedge \mathbf{A}_{11} \wedge \mathbf{A}_{12} \longrightarrow \mathbf{G}_1$.

A proof by means of the *Connection Method* is given by the matrix in Figure 1. The set of arcs represents the *spanning set of connections* which makes the matrix complementary. (The reader may consult [1] for a detailed presentation of the proof of this example.)

The horizontal row of literals on the left denotes the initial situation, the two towers in the middle denote the two applied actions—the left one picks up the block b and the right one puts the block b down on the table—and finally, the column of literals on the right describes the goal situation. (Note that a matrix resembles a set of clauses which are displayed vertically, i.e. we get a row of columns. Note as well that we used an affirmative translation of formulae, i.e. a clause/column represents a conjunction, and the entire matrix must be understood as a disjunction of



(Fig. 1): A Linear Connection Proof

clauses/columns. Moreover, we made use of matrices in non-normal form, i.e. a clause/column may have further matrices/sets of clauses as its elements. This is essential: A conversion of the matrix into normal form would destroy its 'linearity'.) In this proof the value of the variable z represents the plan we were looking for in form of the term $p(l(s,b,a),b))$, which must be read: first lift block b from top of block a and then put block b down on the table. ∎

Of course, we cannot expect every (classical) proof to yield a correct plan—since, unlike to Situation Calculus, our literals are not explicitly time-dependent, they can be reused again and again, although they should no longer be 'valid' after the execution of certain actions—but a notable feature of the proof above is that every instance of a literal is used at most once, i.e. it is involved in at most one connection. Proofs of that kind are called *Linear Connection Proofs* and it is claimed in [1] that this kind of 'linearity' is the necessary restriction to be imposed on proofs in order to generate correct plans. Obviously, sets of connections where each literal is involved in at most one connection were called *linear sets of connections*, and a complementary matrix with a linear set of connections was called a *linear matrix.* (The Linear Connection Method has immediately prompted the question about the underlying logic, a question which has stimulated a lot of research work. See [6] for a survey of publications on this topic.) Let us point out that plan generation with the Linear Connection Method works only if it goes hand in hand with a suitable specification philosophy of plan generation problems. The *formal/syntactic requirements* are that a *well-posed* plan generation problem $(\mathbf{I},\mathbf{A},\mathbf{G})$ must be given as follows:

- $\mathbf{I}$ is a finite set $\{F_1,\ldots,F_n\}$ of ground facts.
- $\mathbf{A}$ is a finite set $\{R_1,\ldots,R_k\}$ of *actions* where each R_i is a universally closed formula of the form

  $$A_{i1} \wedge \ldots \wedge A_{ig} \longrightarrow C_{i1} \wedge \ldots \wedge C_{ih}$$

  with facts $A_{i1},\ldots,A_{ig},C_{i1},\ldots,C_{ih}$. Due to the resemblance to Horn clauses (with multiple or conjunctive heads) such implications are also called *Horn bundles.* (Note that in classical logic this Horn bundle would be equivalent to a set of Horn clauses with heads $C_1,\ldots,C_h$ respectively and identical tails $A_1 \wedge \ldots \wedge A_g$. For the reader familiar with the Connection Method it is easy to see—e.g. in Figure 1—that simply replacing a Horn bundle by a set of Horn clauses would destroy the 'linearity' of the matrix.)
- $\mathbf{G}$ is a finite set $\{G_1,\ldots,G_m\}$ of ground facts.

The plan generation problem $(\mathbf{I},\mathbf{A},\mathbf{G})$ is *solvable* iff there is a Linear Connection Proof for the *specification theorem* $T_{(\mathbf{I},\mathbf{A},\mathbf{G})}$ (In contrast to Example 1 we dropped here the Sit-literals, because their main purpose is to record the generated plan, a job which can be achieved more conveniently otherwise in a concrete implementation.):

$$F_1 \wedge \ldots \wedge F_n \wedge R_1 \wedge \ldots \wedge R_k \longrightarrow G_1 \wedge \ldots \wedge G_m$$

A matrix obtained from a specification theorem will be called *Horn bundle matrix.*

In addition, we have to meet the following *non-formal/semantic requirements* about the intended meaning of the implications which specify the actions: The *antecedent* shall comprise all facts of the existing situation which are involved in the action—either as being necessary conditions for the action's application or as being facts which shall no longer be valid after the action has been carried out. The *consequent* shall comprise all facts which are either newly created by the action or which were involved in the action as preconditions, but are not affected by it.

This convention about the specification of actions entails that all those facts of a situation, which are not included in the antecedent shall survive the application of the action; a property which harmonizes perfectly with the working of Linear Connection Proofs.

This harmony between specification philosophy and formal proof concept is the ultimate reason why no Frame Axioms need to be (explicitly) given with this approach.

In [5] and [2]—to which we refer for details—we presented a proof search algorithm for Linear Connection Proofs which we called *Linear Backward Chaining (LBC)*. This algorithm virtually constructs Linear Connection Proofs by backward search from the goal (clause). It is a kind of tableau calculus which searches for a Linear Connection Proof of a specification theorem $T_{(\mathbf{I},\mathbf{A},\mathbf{G})}$ being derived from a plan generation problem $(\mathbf{I},\mathbf{A},\mathbf{G})$ which meets the requirements listed above. This proof search algorithm has much resemblance to a Horn clause interpreter. The main difference is that we have to assure that no literal/fact is connected twice, for which reason instead of the usual set of unit clauses we have to maintain a pool $POOL of currently unconnected literals, which is initialized by the facts from $\mathbf{I}$.

Moreover, we transform every Horn bundle from $\mathbf{A}$ (of the form $A_1 \wedge \ldots \wedge A_g \longrightarrow C_1 \wedge \ldots \wedge C_h$) into h *rules* which we write down in a PROLOG-like notation:

$$\begin{array}{lll} C_1 & \text{:-} & A_1,\ldots,A_g,\mathsf{NF}(C_2),\ldots,\mathsf{NF}(C_h). \\ C_2 & \text{:-} & A_1,\ldots,A_g,\mathsf{NF}(C_1),\mathsf{NF}(C_3),\ldots,\mathsf{NF}(C_h). \\ & \vdots & \\ C_{h-1} & \text{:-} & A_1,\ldots,A_g,\mathsf{NF}(C_1),\ldots,\mathsf{NF}(C_{h-2}),\mathsf{NF}(C_h). \\ C_h & \text{:-} & A_1,\ldots,A_g,\mathsf{NF}(C_1),\ldots,\mathsf{NF}(C_{h-1}). \end{array}$$

The evaluation of a literal $\mathsf{NF}(C)$ adds C to the $POOL. (The NF-predicate implements a kind of lemma generation facility: If one of the head literals of a Horn bundle satisfies a subgoal, then unit lemmata are generated from all the other head literals. Of course, in contrast to classical logic, these lemmata cannot be used again and again and we also disallow renaming of variables. i.e. we keep track of a global substitution. See [5] for implementation details.) The facts from $\mathbf{G}$ yield a goal clause, from which we start an ordinary backward chaining process, the only difference to Horn clause reasoning is that a literal/fact A is removed from the $POOL when it unifies with a subgoal A in a tableau extension step. (See Example 3 for a presentation of the working of the LBC-algorithm.)

We recall the following theorem from [5] and [2]:

Theorem 1 *The* LBC*-algorithm is a correct and complete proof procedure for generating Linear Connection Proofs.*

The simulation of the LBC-algorithm on top of a tableau-like theorem prover, for which we chose SETHEO[2], is obviously fairly straightforward and we refer to [5] for a detailed description.

3 The rôle of cyclic rules

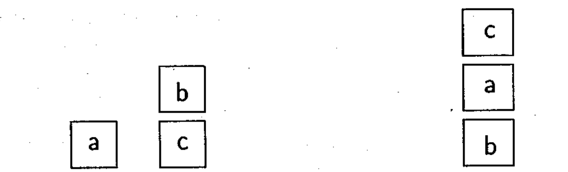
As we mentioned already in the introduction, with the transformation of action/implications into rules some cyclic ones may show up[3]. The most simple case is the following:

Example 2 For an action $\mathbf{A}$ specified by the Horn bundle $A \wedge B \longrightarrow A \wedge D$ our transformation yields the two rules

$$\begin{array}{lll} A \text{ :- } A,B,\mathsf{NF}(D). & & \text{(I)} \\ D \text{ :- } A,B,\mathsf{NF}(A). & & \text{(II)} \end{array}$$ ■

Here rule (I) is cyclic in the literal A. When working with classical logic such a rule would be a tautology and could be discarded without loosing completeness. However, it cannot be discarded with the LBC-algorithm as we will show with the well-known *Sussman Anomaly.* (This example also illustrates the working of the LBC-algorithm.)

Example 3 The so-called Sussman Anomaly is the following small example of blocks world planning: In a world consisting of three blocks a, b and c, the initial situation is given by the facts $O(a, table)$, $C(a)$, $O(c, table)$, $O(b, c)$ and $C(b)$ and the goal consists of the two facts $O(c, a)$ and $O(a, b)$.



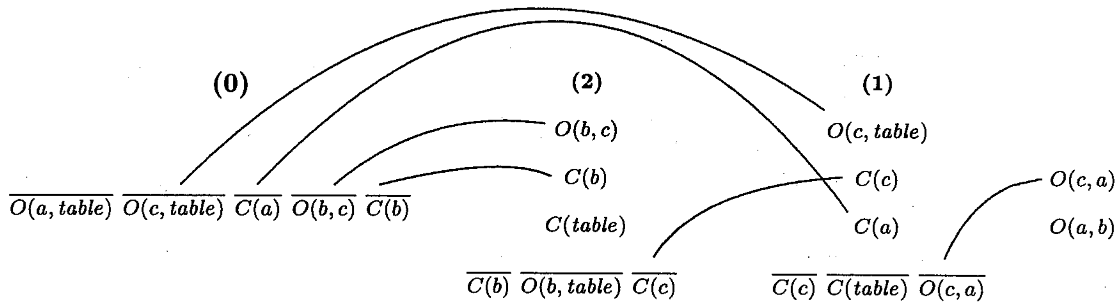
A single action which moves a block x from the top of a block y onto the top of a block z will do, i.e. the Horn bundle:

$$\forall x, y, z : O(x, y) \wedge C(x) \wedge C(z) \longrightarrow O(x, z) \wedge C(y) \wedge C(x)$$

(Since it is assumed that there is always enough room on the *table* to put down a block, i.e. $C(table)$ is always true, we exempted this literal from the linearity restriction.)

[2]SETHEO is a theorem prover for classical full 1-order logic, based on connection tableau (model elimination). It can be obtained via ftp. For further information see [7] or [9] and `http://wwwjessen.informatik.tu-muenchen.de/forschung/reasoning/setheo.html` .

[3]In the following we will deal exclusively with the influence of these cyclic rules on the search space. That these cyclic rules also show some resemblance to Frame Axioms has been discussed at length in [4].



(Fig. 2): Sussman Example

As well known, each of the two goal literals can easily be achieved separately, but none of the respective shortest and also most straightforward plans for achieving one of the goal literals can be extended to a shortest plan which achieves both goal literals altogether. (The shortest plan for both requires an interlacing of actions from the respective plans for the separate goal literals; see [12] for a detailed presentation of this example.) With LBC we run into this problem as well, because we solve one goal literal after the other. However, the cyclic rule

$$C(x) \text{ :- } O(x,y), C(x), C(z), \mathsf{NF}(C(y)), \mathsf{NF}(O(x,z)).$$

saves us in the end.

Assume we want to prove the literals $O(c,a)$ and $O(a,b)$ in this order. We may first select (the following instance of) the rule

$$O(c,a) \text{ :- } O(c, \mathit{table}), C(c), C(a), \mathsf{NF}(C(\mathit{table})), \mathsf{NF}(C(c)). \quad (1)$$

- where $O(c, \mathit{table})$ is a fact of the initial situation,
- $C(c)$ is solved with the rule

  $$C(c) \text{ :- } O(b,c), C(b), C(\mathit{table}), \mathsf{NF}(O(b, \mathit{table})), \mathsf{NF}(C(b)). \quad (2)$$

  whose subgoals $O(b,c)$, $C(b)$ and $C(\mathit{table})$ are in the initial situation
- and $C(a)$ is also in the initial situation.

This means that we moved c on top of a with two actions: 'moving b from top of c down on the table' followed by 'moving c from the table on top of a' (see Figure 2). (For reasons of space and for convenience we will sometimes denote negative literals by overlining instead of prefixing them with $\neg$.) *But now we are stuck!* To move a on b—in order to solve the second goal literal $O(a,b)$, for which we would choose the rule

$$O(a,b) \text{ :- } O(a,y), C(a), C(b), \mathsf{NF}(C(y)), \mathsf{NF}(C(a)).$$

—we would have to remove c from top of a first—i.e. to solve the subgoal $C(a)$—in order to be able to move a. But LBC has no means to destroy $O(c,a)$. The only way out is backtracking: We now solve $C(a)$ with the cyclic rule

$$C(a) \text{ :- } O(a, \mathit{table}), C(a), C(b), \mathsf{NF}(C(\mathit{table})), \mathsf{NF}(O(a,b)).$$

which keeps a clear and moves as a side effect a on top of b, thus yielding already our second goal literal $O(a,b)$. ■

4 The LIP proof search procedure

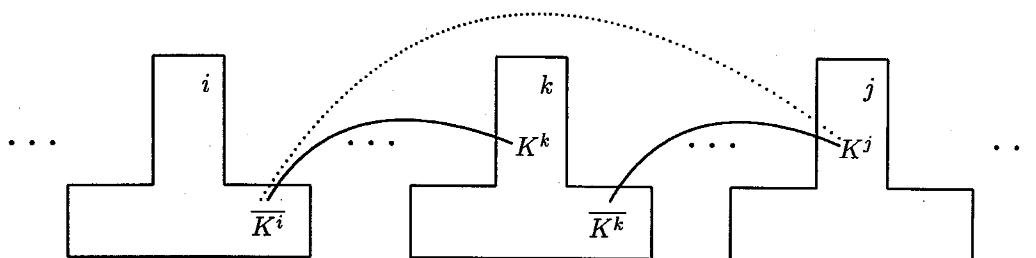
The idea of how to get rid of these cyclic rules can be seen as follows with the Sussman example: Instead of backtracking at the position where we are stuck, we enter the needed action $O(a, \mathit{table}) \wedge C(a) \wedge C(b) \longrightarrow C(a) \wedge C(\mathit{table}) \wedge O(a,b)$ at the literal $O(a,b)$, i.e. we choose the non-cyclic rule

$$O(a,b) \text{ :- } O(a, \mathit{table}), C(a), C(b), \mathsf{NF}(C(\mathit{table})), \mathsf{NF}(C(a)). \quad (3)$$

and solve its subgoal $C(a)^3$ via *cutting the connection* $(\overline{C(a)^2}, C(a)^1)$ between the rules (1) and (2), which means that we replace this connection by the two connections $(\overline{C(a)^2}, C(a)^3)$ and $(\overline{C(a)^3}, C(a)^1)$. (We sometimes use upper indices to refer to (the number of) the action in which a literal occurs. The number 0 refers to the initial situation.)

More generally, if we come across a subgoal K^k in the antecedent of action k, which is rementioned in this action's consequent, then instead of solving it via an extension step, we may look in the matrix constructed so far for a connection $(\overline{K^i}, K^j)$ which we cut, i.e. we replace it by the two connections $(\overline{K^i}, K^k)$ and $(\overline{K^k}, K^j)$ as is shown in Figure 3.

Of course, we are not completely free in our choice of connections to cut as will be seen with the LIP-algorithm below and its correctness proof.



(Fig. 3): Cutting a connection

The LIP-algorithm is obtained from the LBC-algorithm through adding the possibility to insert actions in a plan by cutting connections. This means that apart of SOLVEBYACTION we get a second subprocedure SOLVEBYCUTTING. Cutting connection $(\overline{K^i}, K^j)$ —see Figure 3— means that we insert action k somewhere between the action i and j. In the implementation this insertion will be simulated. To assure correctness, this entails that cutting connection $(\overline{K^i}, K^j)$ requires that, while solving the subgoals of action k, we have to exclude all facts which are generated by actions which depend on (the facts generated by) action j: If such facts are in the $POOL, we are not permitted to use them in extension steps and if they are already connected we may not use them via cutting these connections.

The LIP-algorithm is described in Figure 4 in a nondeterministic way and by use of an Algol-like notation: We have the two self explaining non-deterministic choice constructs

'choose begin
<alternative$_1$>
[] *<alternative$_2$>*
⋮
[] *<alternative$_n$>* **end'**

and

'select (*member* ∈ *set*) **begin** ... **end'**

and the construct FAIL which signals non-success of our choice.

We have two assignment statements $:=_{add}$ and $:=_{rem}$ which add resp. remove an element to resp. from a set (cf. the arithmetic assignments =+ and =− in C).

The algorithm uses the following global variables: $POOL: the set of all unconnected facts, $CONN'S: the set of all made connections, $MATRIX: the set of all used rules together with their number of introduction, $ACTION: the number of already introduced actions/rules, $EXCLUDED: the set of all actions/rules whose generated facts are currently disallowed.

We annotate facts F by an upper index which refers to the number of the action which generated F (F^0 means that F belongs to the initial situation). In case of a subgoal F we write an upper index as well to indicate the number of the action which requires it (F^∞ means that F belongs to the goal). We notate by $F^{[from,to]}$ that there is a connection from a tail literal F of action *from* to a head literal $\overline{F}$ of action *to* (the initial situation is considered an action without tail (and number 0) while the goal clause is considered an action without head (and number ∞))

For simplifying the specification of the LIP-algorithm, we assume to work with ground instances of formulae, thus avoiding to care about unifiers.

For a proof of the correctness and completeness of this algorithm we refer to [3].

5 Experimental evaluation

To get a feeling for the comparative performance of the LIP- and the LBC-algorithm, we ran both on a set of benchmark problems from the blocks world. These examples were generated by a random generator which is delivered with the UCPOP[4] planning system, and a comparison of runtimes between UCPOP and the LBC-algorithm is found in [5]. The problems are enumerated in the order of their random generation, and the number in the middle of the problems' names indicates how many blocks are involved in the problem: E.g. RBW-6-5 refers to the 5th generated problem with 6 blocks. The runtimes—for LBC and LIP—are displayed in Figure 5. Both algorithms were implemented on top of the SETHEO theorem prover (version 3.2 extended by some special built-ins) and were run on a HP 735/99.

[4] UCPOP is a partial order planner developed at the University of Washington, Seattle (see [13]). It is available via anonymous ftp from `~ftp/pub/ai/` at `cs.washington.edu`. In [5] we showed that the LBC-algorithm performs quite well with respect to UCPOP.

proc MAIN;
    \$POOL := \$MATRIX := \$CONN'S := \$EXCLUDED := $\emptyset$; \$ACTION := 0 ;
    'read a plan generation problem $(\mathbf{I}, \mathbf{A}, \mathbf{G})$ ' ;
    for each fact $F \in \mathbf{I}$ **begin** \$POOL $:=_{add} F^0$ **end**;
    for each fact $F \in \mathbf{G}$ **begin** SOLVESUBGOAL($F^\infty, \emptyset$) **end**;
    'extract plan from ($\mathbf{I}, \mathbf{G}$, \$MATRIX, \$CONN'S) and output it';
endproc MAIN

proc SOLVESUBGOAL(F^K, *NewFacts*) ;
    /* solves subgoal F in the tail of action-implication K
      (*NewFacts* are the new facts produced by the chosen rule) */
    choose begin
            if ($F^N \in$ \$POOL $\wedge$ $N \notin$ \$EXCLUDED) **then**
                \$POOL $:=_{rem} F^N$; \$CONN'S $:=_{add} F^{[K,N]}$; **return** (*extension*) ;
            fi;
        [] SOLVEBYACTION (F^K) ; **return** (*extension*) ;
        [] **if** $F \in$ *NewFacts* **then** SOLVEBYCUTTING (F^K) ; **return** (*cutting*) ; **fi**;
    end
endproc SOLVESUBGOAL

proc SOLVEBYACTION(F^K) ;
    /* solves subgoal F with a new action-implication (of number N) */
    N := \$ACTION :=+ 1 ;
    'select a ground rule ' *F:–Tail*, *NewFacts*. ' derived from an action in $\mathbf{A}$ ';
    ExAct := \$EXCLUDED ;
    SolvedByCutting := $\emptyset$;
    for each subgoal $G \in$ *Tail* **begin**;
        Type := SOLVESUBGOAL(G^N, *NewFacts*) ;
        if *Type* $\neq$ *extension* **then** *SolvedByCutting* $:=_{add} G$ **fi**;
    end;
    \$EXCLUDED := *ExAct* ;
    for each fact $G \in$ *NewFacts* **begin**
        if ($G \notin$ *SolvedByCutting*) **then** \$POOL $:=_{add} G^N$ **fi**;
    end;
    \$MATRIX $:=_{add}$ (N, '*F:–Tail*, *NewFacts*.') ; \$CONN'S $:=_{add} F^{[K,N]}$;
endproc SOLVEBYACTION

proc SOLVEBYCUTTING(F^K) ;
    /* solves a surviving subgoal F in the tail of action K by cutting an admissible connection $F^{[B,E]}$ */
    select $F^{[B,E]} \in$ \$CONN'S **begin**
        if $E \in$ \$EXCLUDED **then** FAIL **fi**;
        \$CONN'S $:=_{rem} F^{[B,E]}$; \$CONN'S $:=_{add} F^{[B,K]}$; \$CONN'S $:=_{add} F^{[K,E]}$;
        CLOSURE(K) ;
        if $K \in$ \$EXCLUDED **then** FAIL **fi**;
    end;
endproc SOLVEBYCUTTING

proc CLOSURE(B) ;
    \$EXCLUDED $:=_{add}$ all action numbers which are reflexive-transitively reachable
        from B in the relation between actions given by \$CONN'S ;
endproc CLOSURE

(Fig. 4): The LIP algorithm

Problem	LBC		LIP		
RBW-6-1	13.66	[108]	1.90	[366]	+
RBW-6-2	8.84	[104]	4.87	[363]	+
RBW-6-3	0.15	[76]	0.04	[218]	+
RBW-6-4	0.87	[83]	0.11	[252]	+
RBW-6-5	0.08	[85]	0.07	[255]	+
RBW-6-6	3.01	[109]	1.54	[321]	+
RBW-6-7	4.73	[103]	2.57	[361]	+
RBW-6-8	16.60	[108]	2.60	[364]	+
RBW-6-9	2.00	[108]	1.84	[374]	+
RBW-6-10	0.95	[88]	0.10	[252]	+
RBW-6-11	3.41	[119]	3.90	[428]	−
RBW-6-12	0.25	[95]	0.26	[311]	−

Problem	LBC		LIP		
RBW-7-1	124.18	[141]	119.74	[517]	+
RBW-7-2	2.01	[108]	1.93	[374]	+
RBW-7-3	91.88	[117]	9.22	[376]	+
RBW-7-4	11.46	[126]	6.46	[382]	+
RBW-7-5	19550.99	[128]	10205.84	[590]	+
RBW-7-6	0.02	[100]	0.03	[244]	−
RBW-7-7	1976.05	[120]	224.29	[440]	+
RBW-7-8	0.21	[105]	0.19	[282]	+
RBW-7-9	159.33	[134]	134.91	[510]	+
RBW-7-10	8.72	[120]	8.73	[442]	−

(Fig. 5): Problems with 6 and 7 blocks (runtimes, proof sizes and speed up)

Let us mention finally that both implementations are simulations, and going from the LBC to the LIP-algorithm we get quite some overhead due to the additional explicit storage of connections. (We added in brackets the proof size, i.e. the size of the simulation and not of the Linear Connection Proof, to give an further idea of the increase of effort, although lots of connections stem from the plan extraction routine which is more complicated with LIP than with LBC.) Further speed ups by linear factors can certainly be achieved through better data organisation.

6 Concluding remarks

We presented an improvement of plan/proof search for the Linear Connection Method. The new LIP-algorithm compared to the LBC-algorithm is a step from total order planning to partial order planning. A question to be investigated in the future is how the LIP-algorithm compares to already known partial order planning algorithms with respect to the search space.

Acknowledgements: We want to thank Johann Schumann for providing some useful built-ins in SETHEO which helped to speed up the LIP-algorithm.

References

[1] W. Bibel. A Deductive Solution for Plan Generation. *New Generation Computing*, 6:115–132, 1986.

[2] B. Fronhöfer. *The Action-as-Implication Paradigm: Formal Systems and Application*, volume 1 of *Computer Science Monographs*. CSpress, München, Germany, 1996. (revised version of Habilitationsschrift, TU München 1994).

[3] B. Fronhöfer. Cutting Connections in Linear Connection Proofs. Technical Report AR-96-01, Technische Universität München, 1996. available from ftp://ftp.informatik.tu-muenchen.de/local/lehrstuhl/jessen/Automated_Reasoning/Reports/AR-96-01.ps.gz.

[4] B. Fronhöfer. Cyclic Rules in Linear Connection Proofs. In G. Görz and S. Hölldobler, editors, *KI-96: Advances in Artificial Intelligence*, pages 67–70, Dresden, Germany, 1996. Springer, LNAI 1137.

[5] B. Fronhöfer. Situational Calculus, Linear Connection Proofs and STRIPS-like Planning: An Experimental Comparison. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 193–209, Terrasini, Palermo, 1996. Springer, LNAI 1071.

[6] B. Fronhöfer. Plan Generation with the Linear Connection Method. *INFORMATICA, Lithuanian Academy of Sciences*, 1997. (to appear).

[7] Ch. Goller, R. Letz, K. Mayr, and J. Schumann. Setheo V3.2: Recent Developments. In Alan Bundy, editor, *CADE'94*, pages 778–782, 1994.

[8] C. Green. Application of Theorem Proving to Problem Solving. In *IJCAI-1*, pages 219–239, 1967.

[9] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A High-Performance Theorem Prover. *JAR*, 8(2):183–212, 1992.

[10] V. Lifshitz. On the Semantics of STRIPS. In M.P. Georgeff and A.L. Lansky, editors, *Workshop on Reasoning about Actions and Plans*, pages 1–8. Morgan Kaufmann, 1986.

[11] J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[12] N. J. Nilsson. *Principles of Artificial Intelligence.* Springer, 1982.

[13] J.S. Penberthy and D. Weld. UCPOP: a sound, complete, partial order planner for ADL. In *KR-92*, pages 103–114, October 1992.