A Curiously Effective Backtracking Strategy for Connection Tableaux

Michael Färber[0000−0003−1634−9525]

Vrije Universiteit Amsterdam, The Netherlands
michael.faerber@gedenkt.at

Abstract. Automated proof search with connection tableaux, such as implemented by Otten's leanCoP prover, depends on backtracking for completeness. Otten's restricted backtracking strategy loses completeness, yet for many problems, it significantly reduces the time required to find a proof. Kaliszyk has implemented restricted backtracking using *restricted prover states* which are very efficient, but cannot perform complete strategies. I introduce a new restricted backtracking strategy based on the notion of *exclusive cuts*. This strategy is the most complete strategy that can be realised using Kaliszyk's restricted prover states. I implement the strategy in a new prover called *meanCoP* and show that it greatly improves upon the previous best strategy in leanCoP.

1 Introduction

Bibel's connection method [2] is a proof search method similar to Andrews's matings [1]. Compared to other proof search methods such as resolution, the connection method has several merits: It is goal-oriented, enabling natural conjecture-directed proof search. It can be used with relatively little effort for non-classical logics such as intuitionistic or modal logics [15, 17], and non-clausal search [18]. Finally, most connection calculi have only very few and simple rules, making it easy to certify proofs in proof assistants such as HOL Light [11, 4].

One of the most influential connection provers is Otten's leanCoP [15]. Its outstanding ratio between code size and effectiveness has made it a frequently used vehicle to experiment with new search strategies. leanCoP uses bounded depth-first search together with iterative deepening to explore larger and larger potential proofs. As the proof search is not confluent, leanCoP employs backtracking to preserve completeness.

This article studies backtracking that guides connection proof search. Otten showed that by restricting backtracking, the prover becomes significantly more effective for many problems as this reduces the search space, at the expense of losing completeness (section 3). Kaliszyk implemented Otten's restricted backtracking strategy very efficiently using a technique that I call *restricted prover states* (section 4). Upon implementing a new prover based on Kaliszyk's restricted states, I unexpectedly discovered a novel incomplete strategy, which I found to be the most complete strategy admissible by restricted prover states

(section 5). I implemented this strategy in a new prover called *meanCoP* based on Otten's leanCoP and Kaliszyk's restricted states (section 6). The new strategy improves upon the former best strategy in leanCoP dramatically (section 7).

2 Preliminaries

In this article, we will use classical first-order logic without equality. However, the techniques shown can be also applied to non-classical logics.

A term t is either a variable (denoted by x, y, z) or the application of a constant (denoted by a, b, c) to terms. An atom A is the application of a predicate (denoted by p, q, r) to terms. Predicates and constants have associated fixed arities. A literal L is an atom A or its negation $\neg A$. The complement $\overline{}$ of a literal is defined such that $\overline{A} = \neg A$ and $\overline{\neg A} = A$. A term substitution σ is a mapping from variables to terms. Applying a substitution σ to a literal L, denoted as σL, substitutes all variables of L with their mappings. Two literals L_1, L_2 can be unified under a substitution σ if $\sigma L_1 = \sigma L_2$.

A formula in conjunctive normal form (CNF) is a conjunction ($\wedge$) of disjunctions ($\vee$) of literals. A clause is a set of literals, and a matrix is a set of clauses. We interpret a clause as the disjunction of its literals, and we interpret a matrix as the conjunction of its (interpreted) clauses. It is easy to see that for each formula in CNF, there is an equivalent matrix.

Example 1. Consider the formula

$$(\mathsf{p}(x) \vee \mathsf{q}(x)) \wedge (\neg \mathsf{p}(y) \vee \mathsf{r}(y)) \wedge \neg \mathsf{p}(z) \wedge \neg \mathsf{r}(\mathsf{a}) \wedge \neg \mathsf{r}(\mathsf{b}) \wedge \neg \mathsf{q}(\mathsf{c}).$$

Its equivalent matrix is

$$M = \left[\begin{bmatrix} \mathsf{p}(x) \\ \mathsf{q}(x) \end{bmatrix} \begin{bmatrix} \neg \mathsf{p}(y) \\ \mathsf{r}(y) \end{bmatrix} \left[\neg \mathsf{p}(z) \right] \left[\neg \mathsf{r}(\mathsf{a}) \right] \left[\neg \mathsf{r}(\mathsf{b}) \right] \left[\neg \mathsf{q}(\mathsf{c}) \right] \right],$$

which we will use as running example throughout this paper.

In this paper, we treat proof search using the clausal connection tableaux calculus [13, 19].

Definition 2 (Connection Calculus). The axiom and the rules of the clausal connection calculus are given in Figure 1. The words of the connection calculus are tuples $\langle C, M, \mathit{Path} \rangle$, where M is a matrix, and C and Path are sets of literals or ε. C is called the *subgoal clause* and Path is called the *active path*. In the calculus rules, σ is a global (or rigid) term substitution; that is, it is applied to the whole derivation.

An application of a proof rule is called a proof step. A derivation for $\langle C, M, \mathit{Path} \rangle$ with the term substitution σ, in which all leaves are axioms, is called a *connection proof* for $\langle C, M, \mathit{Path} \rangle$. A connection proof for $\langle \varepsilon, M, \varepsilon \rangle$ is called a connection proof for M.

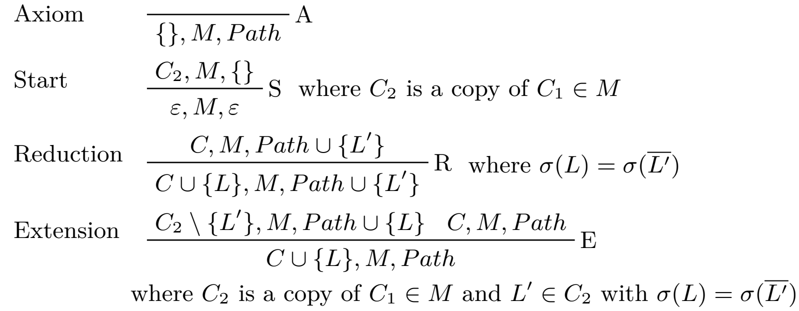


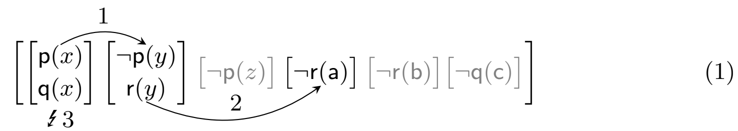
Fig. 1: Clausal connection calculus rules.

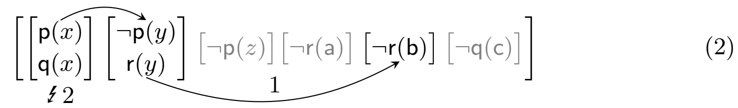
Bibel proved soundness and completeness of the calculus: for any formula F in CNF, we have that F is unsatisfiable iff there is a connection proof for the matrix corresponding to F [2].

Proof search proceeds by constructing derivations from bottom to top. We can understand a derivation for $\langle C, M, Path \rangle$ as an attempt to prove $(M \wedge Path) \implies C$, where we interpret $Path$ as conjunction of its literals. By interpreting ε as empty set, a derivation for $\langle \varepsilon, M, \varepsilon \rangle$ can be seen as a proof attempt of $M \implies \bot$.

We say that any reduction or extension step as in Figure 1 *connects* L to L'. We illustrate this by drawing an arrow from L to L' in the matrix. In this paper, we will only use extension steps in examples.

Let us walk through a *failed* proof search attempt for the matrix M from Example 1, and show its graphical representation as well as its resulting derivation in the calculus.





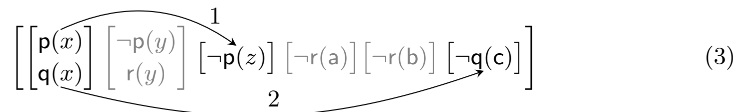


Fig. 2: Graphical representation of proof search.

Example 3. Consider matrix M from Example 1. Matrix (1) of Figure 2 illustrates a proof search attempt through M. We write the proof step n in matrix m as $(m.n)$. The proof search proceeds as follows: We first choose the first clause in M as start clause. This obliges us to connect both $\mathsf{p}(x)$ and $\mathsf{q}(x)$. We start with $\mathsf{p}(x)$, which we choose to connect in step (1.1) to $\neg\mathsf{p}(y)$, setting $\sigma(x) = y$. This in turn obliges us to connect $\mathsf{r}(y)$, which we choose to connect in step (1.2) to $\neg\mathsf{r}(\mathsf{a})$, setting $\sigma(x) = \sigma(y) = \mathsf{a}$. We are now left with the obligation to connect $\mathsf{q}(x)$. However, at this point (1.3), we cannot connect $\mathsf{q}(x)$ to any literal due to σ. As we are stuck at this point, we mark this with ↯. Figure 3 shows a derivation for M that corresponds to this proof search. The extension steps in the derivation are labelled like the corresponding proof steps in matrix (1). The derivation is *not* a connection proof for M, because the leaf $\langle \{\mathsf{q}(x)\}, M, \{\} \rangle$ is not an axiom.

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\{\}, M, \{\mathsf{p}(x), \mathsf{r}(y)\}}\,\text{A} \qquad \dfrac{}{\{\}, M, \{\mathsf{p}(x)\}}\,\text{A}}{\{\mathsf{r}(y)\}, M, \{\mathsf{p}(x)\}}\,\text{E (1.2)} \qquad \begin{matrix} ↯ \\ \vdots \\ \{\mathsf{q}(x)\}, M, \{\} \end{matrix}}{\{\mathsf{p}(x), \mathsf{q}(x)\}, M, \{\}}\,\text{E (1.1)}}{\varepsilon, M, \varepsilon}\,\text{S}
$$

Fig. 3: Derivation with $\sigma(x) = \sigma(y) = \mathsf{a}$.

This example illustrates that search in connection tableaux is not confluent, i.e. we can end up with unprovable leaves in derivations for a matrix M although the formula corresponding to M is unsatisfiable. This makes it necessary to backtrack to previous states of derivations to obtain a complete proof search method. We will study two backtracking strategies in the next section.

3 Backtracking

In the failed proof attempt shown in Example 3, we frequently talked about *obligations* and *choices* (also called *promises* and *alternatives*). Fulfilling obligations assures soundness, and trying choices exhaustively assures completeness.

Otten's *unrestricted backtracking* strategy [15] is sound and complete. It makes choices until an obligation cannot be fulfilled. At this point, the strategy changes the most recent choice for which there is an untried alternative. The strategy succeeds if it fulfills all obligations, and fails if it runs out of alternatives.

Example 4 (Unrestricted Backtracking). Consider the proof search attempt in Example 3. The last choice we made in that example was to connect $\mathsf{r}(y)$ to $\neg\mathsf{r}(\mathsf{a})$ as part of step (1.2). We can make a different choice here, namely connect $\mathsf{r}(y)$ to $\neg\mathsf{r}(\mathsf{b})$, which we perform in step (2.1). However, as it turns out, this will not help us when we have to deal with $\mathsf{q}(x)$ anew in step (2.2), for now

we have $\sigma(x) = \sigma(y) = \mathsf{b}$, which still does not permit a connection from $\mathsf{q}(x)$. So we backtrack again, leading to the proof search shown in matrix (3). This time, the last choice was to connect $\mathsf{r}(y)$ to $\neg\mathsf{r}(\mathsf{b})$, but now, we cannot find a different way to connect $\mathsf{r}(y)$. So we look at our second to last choice, namely to connect $\mathsf{p}(x)$ to $\neg\mathsf{p}(y)$. We can make an alternative choice here as step (3.1), namely to connect $\mathsf{p}(x)$ to $\neg\mathsf{p}(z)$. Now we are back once more to the dreaded $\mathsf{q}(x)$, but finally, due to $\sigma(x) = \sigma(z)$ not pointing to an actual term, we can connect $\mathsf{q}(x)$ to $\neg\mathsf{q}(\mathsf{c})$ as step (3.2). This concludes the proof, as we have no more obligations left at this point.

Otten's *restricted backtracking* strategy [16] is sound, but incomplete. However, it is often significantly more effective than the complete strategy. To define it, Otten introduces the property of *solvedness* on literals in a proof search.

Definition 5 (Principal literal, solved literal). When the reduction or extension rules are applied, the literal L (see Figure 1) is called the *principal literal* of the proof step. A reduction step *solves* a literal L iff L is its principal literal. An extension step S *solves* a literal L iff L is the principal literal of S and there is a proof for the left premise of S, i.e. there is a derivation for the left premise of S having only axioms as leaves.

The restricted backtracking strategy works like the unrestricted one, with one exception: Once a literal is solved, restricted backtracking discards all choices to solve the literal differently.

Example 6 (Restricted Backtracking). Consider matrix (1) of Figure 2. Proof step (1.1) does not solve any literal, so at this point, proof search behaves like in Example 4. Proof step (1.2) solves $\mathsf{r}(y)$, so at that point, alternative choices to solve $\mathsf{r}(y)$ are discarded. At the same time, step (1.2) also solves $\mathsf{p}(x)$, so alternative choices to solve $\mathsf{p}(x)$ are discarded as well. In proof step (1.3), we note that $\mathsf{q}(x)$ cannot be connected. However, unlike in Example 4, we have no alternative choices left to backtrack to because they were discarded as a result of step (1.2). That means that the restricted backtracking strategy cannot find a proof once we commit to connecting $\mathsf{p}(x)$ to $\neg\mathsf{p}(y)$ as first step.

4 Prover States

The connection prover leanCoP is compactly implemented in Prolog as a recursive predicate `prove` that takes C and *Path* as parameters, using a helper predicate `lit` that models M [15]. leanCoP implements restricted backtracking using Prolog's built-in cut operator [16].

Kaliszyk has reimplemented leanCoP using stacks for backtracking [8]. A prover state in Kaliszyk's implementation is as follows.

Definition 7 (Unrestricted Prover State). An *unrestricted prover state* is a tuple $(C, M, \mathit{Path}, \sigma, \mathit{Step}, \mathit{Alts}, \mathit{Prms})$. Here, *Path* is a stack containing the active path, σ is the substitution, *Step* is the last tried proof step, *Alts* is the stack

of alternatives $(C, Path, |\sigma|, Step, Prms)$, and $Prms$ is the stack of promises $(C, |Path|, |Alts|)$. The cardinality of the domain of σ is $|\sigma|$, and the number of elements of a stack S is $|S|$.

To recall, alternatives keep track of previously made choices, and promises keep track of proof obligations, i.e. which literals have to be connected. When the prover backtracks, it changes its state based on an alternative, and when the prover solves a literal, then it changes its state based on a promise. We say that we *apply* alternatives and promises to a state.

Definition 8 (Alternative and Promise Application). Suppose that we are in an unrestricted prover state $(C, M, Path, \sigma, Step, Alts, Prms)$. We *apply an alternative* $(C', Path', |\sigma'|, Step', Prms')$ by removing the latest bindings in σ until there are only $|\sigma'|$ left, and replacing C with C', $Path$ with $Path'$, $Step$ with $Step'$, and $Prms$ with $Prms'$. We *apply a promise* $(C', |Path'|, |Alts'|)$ by replacing C with C' and popping $Path$ and $Alts$ until they contain $|Path'|$ and $|Alts'|$ elements, respectively.

When performing restricted backtracking, Kaliszyk's prover uses states that store only the lengths of $Path$ and $Prms$ inside the alternatives. I call such states *restricted prover states*.

Definition 9 (Restricted Prover State). A *restricted prover state* is a tuple $(C, M, Path, \sigma, Step, Alts, Prms)$. The only difference to an unrestricted prover state is that here, $Alts$ is the stack of alternatives $(C, |Path|, |\sigma|, Step, |Prms|)$. We *apply an alternative* $(C', |Path'|, |\sigma'|, Step', |Prms'|)$ by replacing C with C' and $Step$ with $Step'$, removing the latest bindings in σ until there are only $|\sigma'|$ left, and popping $Path$ and $Prms$ until they contain $|Path'|$ and $|Prms'|$ elements, respectively.

Restricted states are considerably more efficient than unrestricted states. To show this, I evaluated unrestricted states where I implemented $Path$ and $Prms$ as stacks or singly linked lists. Both using stacks and lists, unrestricted states were significantly slower than restricted states; for example, the list version increased runtime for some problems by more than 30%.

However, restricted prover states are admissible only for strategies where applying an alternative *never* requires any stack in the state, such as $Path$, to grow. We can thus use restricted prover states for restricted backtracking, but not for unrestricted backtracking, where the lengths of $Path$ and $Prms$ may need to increase by backtracking, i.e. by applying an alternative.

Example 10. Consider the proof search using unrestricted backtracking in Example 4. There, we backtracked after step (1.3), where the path is $\{\}$ and the promises are $\{\mathsf{q}(x)\}$, to the outcome of step (1.1), where the path is $\{\mathsf{p}(x)\}$ and the promises are $\{\mathsf{q}(x), \mathsf{r}(y)\}$. The length of both path and promises increases by backtracking.

There are more complete strategies than Otten's restricted backtracking strategy that can be performed using restricted prover states. In the next section, I will show a new strategy that is the most complete strategy that can be performed with restricted prover states.

5 Less Restricted Backtracking

Restricted backtracking can be decomposed into two cuts: cuts on reduction and cuts on extension steps. Kaliszyk already implemented these two cuts separately, but did not describe it, as they are usually most useful in conjunction. In our case, the distinction will be more useful, as it allows us to more succinctly describe our new backtracking strategy.

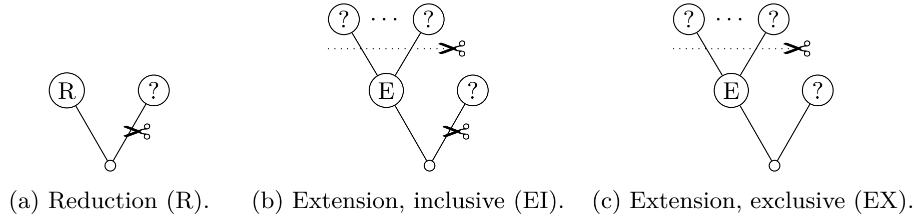
I will now distinguish *inclusive* and *exclusive* cuts. An inclusive cut discards all alternatives to solve a literal, whereas an exclusive cut discards all alternatives to solve a literal, *except for derivations starting with a different proof step.* Otten's restricted backtracking strategy shown in section 3 uses inclusive cuts on both reduction and extension steps. To the best of my knowledge, exclusive cuts have not been researched before.

Example 11 (Exclusive Cut). We will, for the last time, revisit the proof search in Figure 2. After proof step (1.2), exclusive cut discards alternative ways to solve $\mathsf{p}(x)$, except for derivations starting with different extension steps. As a result, after being stuck at step (1.3) with $\mathsf{q}(x)$, we *can* backtrack unlike in Example 6, namely to the proof search in matrix (3), because it solves $\mathsf{p}(x)$ in step (3.1) starting with a different extension step. From there, proof search behaves again like in Example 4, solving the problem in step (3.2) after connecting $\mathsf{q}(x)$.

To sum up the outcomes of different strategies on the proof search in Figure 2: The complete strategy (Example 4) solves the problem, going through all stages from (1) to (3). The exclusive cut (Example 11) also solves the problem, but takes one stage less, going through only (1) and (3). The inclusive cut (Example 6) fails after (1). For this example, exclusive cut therefore is the most efficient strategy.

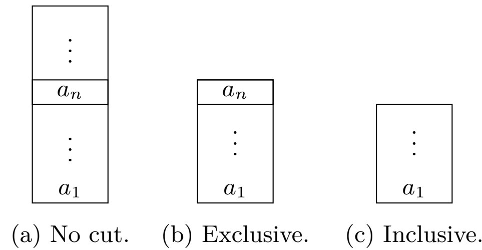
For reduction steps, an exclusive cut is equivalent to no cut, so I distinguish between inclusive and exclusive cut only for extension steps. I abbreviate (inclusive) cut on reduction steps as R and inclusive and exclusive cut on extension steps as EI and EX, respectively. Otten's restricted backtracking strategy can be described as a combination of R and EI, written as REI.

Figure 4 visualises the alternatives that are cut once a literal is solved. In each of the trees, the left child of the root is a proof step S that solves a literal, the children of the left child are alternatives to proof steps that are descendants of S, and the right child is the alternative to S. Reduction and extension steps are marked as R and E, respectively, and alternatives are marked as "?". Both the R and EI cut are inclusive cuts because the right child is cut, and the EX cut is exclusive because the right child is preserved. Both cuts on extension steps eliminate all alternatives below the proof step.



(a) Reduction (R). (b) Extension, inclusive (EI). (c) Extension, exclusive (EX).

Fig. 4: Effect of different cuts on the tree of alternatives.



(a) No cut. (b) Exclusive. (c) Inclusive.

Fig. 5: Effect of different cuts on the stack of alternatives.

Figure 5 shows the effect of inclusive and exclusive cut on the stack of alternatives (section 4). Figure 5a shows the initial situation of the stack after a literal was solved with a proof step whose alternative is a_n. Above a_n are alternatives to proof steps added after a_n, and below a_n are alternatives to proof steps added before a_n. Using no cut does not change the stack at this point. Both exclusive and inclusive cut eliminate all alternatives added after a_n, but the exclusive cut (Figure 5b) keeps one alternative more than the inclusive cut (Figure 5c), namely a_n.

Like inclusive cut, exclusive cut can be implemented with restricted prover states (section 4); changing an implementation to use inclusive instead of exclusive cut amounts to truncating the stack of alternatives to length n instead of length $n - 1$ once a literal was solved. Moreover, exclusive cut is the *most complete* strategy that can be performed with restricted states, because it discards *precisely* those alternatives that cannot be backtracked to without potentially making the path and the promises larger.

The seemingly small difference between inclusive and exclusive cut has a large impact on the effectiveness of the prover. We will see this in the evaluation (section 7).

6 Implementation

I implemented a connection prover called *meanCoP* based on leanCoP in Rust.[1] Like C++, Rust favours zero-overhead abstractions [22], making it a suitable candidate for the development of high-performance automated theorem provers. I use functional programming for preprocessing and imperative programming for the prover loop. I follow Kaliszyk's implementation for the prover loop [8], but diverge from it by using loops in place of tail recursive functions (as tail call optimisation is not guaranteed in Rust) and using dynamic instead of static arrays in order to allow for arbitrarily sized stacks, terms, etc. The prover loop does not use Rust's standard library and can be therefore compiled to targets such as WASM, which can be used to create websites with an embedded prover that is run locally in the browser. Furthermore, meanCoP contains a tiny proof checker that is run before outputting a proof. This has been very valuable to assure that exclusive cut is sound.

meanCoP supports most major features of leanCoP, such as conjecture-directed search, regularity, and lemmas [15]. Furthermore, meanCoP supports the R, EI, and EX cuts (section 5). By default, meanCoP uses (inclusive) cut on lemma steps, which does not hamper completeness, as lemma steps do not impact the substitution. meanCoP uses restricted prover states when either EI or EX is used, and (slower) unrestricted prover states otherwise (section 4).

7 Evaluation

I evaluate the performance of meanCoP (section 6) and other provers on several first-order problem datasets.[2] For every dataset and prover, I measure the number of problems solved by the prover in a given time. All evaluated connection prover strategies use conjecture-directed search and non-definitional CNF. I use the same hardware, the same timeout, and the same datasets as in my previous evaluation of connection provers together with Kaliszyk and Urban [5]. I will compare the results in this paper with those of the previous evaluation.

I use a 48-core server with AMD Opteron 6174 2.2GHz CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU. Each problem is always assigned one CPU. I run every prover with a timeout of 10 seconds per problem.

The first-order logic datasets used for evaluation are the non-clausal first-order problems in TPTP 6.3.0 (files matching `*+?.p`), the bushy and chainy datasets from MPTP2078 and the Miz40 dataset (all derived from Mizar), as well as the FS-top dataset (derived from the HOL Light part of Flyspeck). Table 1 gives the number of problems contained in each dataset.

[1] The name meanCoP abbreviates "more efficient, albeit non-lean connection prover". The source code of meanCoP is available at `https://github.com/01mf02/cop-rs`. I evaluated revision 699191d compiled with Rust 1.49.

[2] The evaluation results are available in more detail at `http://cl-informatik.uibk.ac.at/~mfaerber/cade-28.html`.

Table 1: Evaluation datasets and the number of contained first-order problems.

Dataset	TPTP	bushy	chainy	Miz40	FS-top
Problems	7492	2078	2078	32524	27111

7.1 Comparison of meanCoP strategies

The first part of the evaluation studies the impact of different combinations of cuts on the number of problems solved by meanCoP.

Table 2: Number of solved problems.

Cut	TPTP	bushy	chainy	Miz40	FS-top
None	1711	546	208	9236	4038
R	1835	641	250	12961	4446
EI	1972	724	322	13850	4248
EX	2028	814	264	15499	4758
REI	1982	730	**335**	13560	4267
REX	**2102**	**850**	294	**16134**	**4994**

Table 2 shows the number of problems solved by strategy. For all datasets, the complete strategy without any cut solves the least problems. Among the previously implemented cuts, namely R, EI, and REI, REI solves the most problems, except on the dataset FS-top, where R prevails. Adding cut on reduction (R) to any strategy increases the number of solved problems, except for the Miz40 dataset. The strategies with exclusive cut on extension steps (EX, REX) outperform those with inclusive cut (EI, REI) on all datasets except for the chainy one. Perhaps this is because the chainy dataset contains many problems with unusually many axioms, implying a larger explosion of the search space on which a more aggressive cut could be beneficial.

On most datasets, the strategies using exclusive cut bring an impressive improvement of the prover power. The REX strategy increases the number of solved problems compared to the REI strategy by 16.4% for bushy, 17.0% for FS-top (12.3% if we compare with the R cut), and 19.0% for Miz40. Remarkably, on TPTP, the improvement turns out much smaller with only 6.1%.

Table 3 shows the union of problems solved by a portfolio of strategies. The first line shows the problems solved by any of the four previously used cut strategies, including the unrestricted backtracking strategy without cut, but also combinations of cut on reduction and inclusive cut on extension, while excluding our new exclusive cut. Comparing the first line with the REX results from Table 2, we see that the new REX strategy solves single-handedly more

Table 3: Union of solved problems.

Cut	TPTP	bushy	chainy	Miz40	FS-top
Any but (R)EX	2255	833	384	15880	5185
REX and REI	2274	907	383	16801	5191
Any	2363	927	396	17434	5560

problems than a union of four strategies on the bushy and FS-top datasets, which is quite noteworthy. The second line shows the problems solved by any of the two most powerful strategies, including the REX strategy that uses exclusive cut. This combination is better than the combination of all previous cut strategies in the first line on all datasets except for chainy, where it is only one problem behind. Combining all strategies (line 3) clearly boosts the number of solved problems compared to the previously available strategies (line 1), namely 11.3% for bushy, 4.8% for TPTP, 9.8% for Miz40, and 7.2% for FS-top.

In conclusion, the new strategies do not only prove more problems, but the problems they solve are also sufficiently complementary from the problems solved by previously available strategies. This makes the new strategies attractive in portfolio modes.

7.2 Comparison with other provers

I compare meanCoP with several other provers that I previously evaluated in joint work with Kaliszyk and Urban [5, table 2].

Among the non-connection provers, I evaluate Vampire 4.0 [12] and E 2.0 [21], which performed best in the first-order category of CASC-J8 [23]. Vampire and E are written in C++ and C, respectively, implement the superposition calculus, and perform premise selection with SInE [6]. Furthermore, Vampire integrates several SAT solvers [3], and E automatically determines proof search settings for a given problem. I run E with `--auto-schedule` and Vampire with `--mode casc`. In addition, I evaluate the ATP Metis [7]: It implements the ordered paramodulation calculus (having inference rules for equality just like the superposition calculus), but is considerably smaller than Vampire and E and is implemented in Standard ML.

Furthermore, I evaluate two other connection provers apart from meanCoP, namely leanCoP 2.1 using the Prolog compiler ECLiPSe 5.10, and fleanCoP, which is a reimplementation of leanCoP in OCaml using streams. I evaluate both provers using restricted backtracking, i.e. the REI cut. Care is taken that leanCoP-REI, fleanCoP-REI, and meanCoP-REI perform the same inferences.

Table 4 shows the results of the comparison. The meanCoP prover is much faster (in terms of inferences per time) than both the original leanCoP as well as its OCaml reimplementation using streams, even if they perform the same inferences. The largest improvement can be seen on the chainy dataset, where leanCoP, fleanCoP and meanCoP (all using the REI strategy) prove 182, 289

Table 4: Number of solved problems by different provers.

Prover	TPTP	bushy	chainy	Miz40	FS-top
Vampire	4404	1253	656	30341	6358
E	3664	1167	287	26003	7382
Metis	1376	500	75	18519	3537
leanCoP-REI	1673	606	182	11243	3664
fleanCoP-REI	1859	670	289	12204	3980
meanCoP-REI	1982	730	335	13560	4267
meanCoP-REX	2102	850	294	16134	4994

(+58.8%), and 335 (+84.1% compared to leanCoP and +16.0% compared to fleanCoP) problems, respectively.

Vampire proves most problems on all datasets except for FS-top, where E prevails. On the chainy dataset, fleanCoP and meanCoP prove more problems than E, which is likely due to the conjecture-directed search. Metis proves the fewest problems, except on the Miz40 dataset, where it proves more problems than any connection prover, but less than Vampire and E.

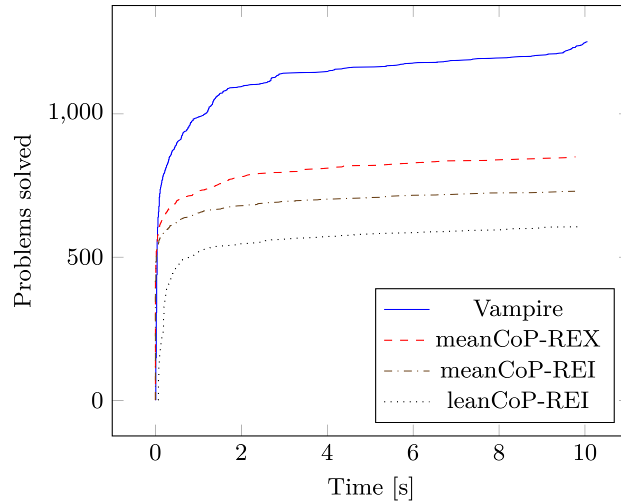


Fig. 6: Evolution of number of solved bushy problems over time.

Figure 6 shows for several provers the number of problems on the bushy dataset proved up to a certain time. meanCoP-REX proves considerably more problems than Vampire in the first 10 milliseconds, namely 432 versus a single one. However, Vampire catches up after about 50 milliseconds, leaving all other

provers behind. leanCoP solves its first problem about 50 milliseconds after any other prover, which is due to the relatively high start-up time caused by the compilation of the prover at each run. After 50 milliseconds, the order between the provers remains stable. The curves for the connection provers flatten with time, whereas the curve for Vampire shows several “bumps”, which are most likely due to strategy scheduling. The average time used to solve a problem is 0.45 seconds for meanCoP-REI, 0.56 seconds for meanCoP-REX, 0.76 seconds for leanCoP-REI, and 0.99 seconds for Vampire.

8 Related Work

The MaLeCoP prover by Urban et al. [25] and the FEMaLeCoP prover by Kaliszyk and Urban [9] were among the first to use machine learning to guide connection proof search. These provers order the applicable extension steps in prover states by Naive Bayesian probabilities that are inferred from previous proofs. Like leanCoP, they use depth-first search, iterative deepening, and backtracking, which makes such provers likely to benefit from advances in backtracking strategies as presented in this work.

Other connection provers have moved away more from leanCoP’s traditional backtracking-based search. I developed monteCoP in joint work with Kaliszyk and Urban [5], Kaliszyk et al. developed rlCoP [10], and Olšák et al. developed follow-up work to rlCoP [14]. All these provers use machine-learnt policies to explore the search space, with Monte Carlo Tree Search taking the role that backtracking plays in leanCoP. For that reason, such provers can probably not directly profit from this work.

We evaluate FEMaLeCoP and monteCoP on the bushy dataset, using 60 seconds timeout, definitional clausification and the REI strategy. Comparing the non-learning with the learning versions of the provers, the increase in number of solved problems is from 563 to 601 for monteCoP (+6.7%) and from 577 to 592 for FEMaLeCoP (+2.6%) [5, table 8], thus far below the increase of 16.4% gained in the current work.

Kaliszyk et al. evaluate rlCoP on the Miz40 dataset, where it proves 16108 problems after 10 iterations of training. Although I evaluate meanCoP on the same dataset, where meanCoP proves 16134 problems, it is unfortunately difficult to compare the results for two reasons: First, Kaliszyk et al. limit the number of inferences instead of the time allotted to the prover. Second, most inferences performed by rlCoP end up in prover states that are not actually explored, due to not being chosen by Monte Carlo Tree Search.

Another line of work extends connection provers with native support for equality. Rawson’s lazyCoP is a connection prover based on Paskevich’s connection tableaux calculus with lazy paramodulation [24, 20]. It supports first-order logic with equality. Given that lazyCoP does not use backtracking to control the search, it seems unlikely that exclusive cut could be integrated in this system.

Otten's ileanCoP for intuitionistic logic [15] and MleanCoP for modal logic [17], as well as nanoCoP for nonclausal proof search [18], could all integrate exclusive cut seamlessly.

9 Conclusion

I introduced a new kind of cut on extension steps called exclusive cut, which discards all alternatives to solve a literal, except for derivations starting with a different extension step. Exclusive cut can be implemented with the same fast restricted prover states as the previously used inclusive cut; furthermore, exclusive cut is the most complete strategy that can be performed using restricted prover states. I implemented the described techniques in a new prover called meanCoP based on restricted prover states. Evaluating meanCoP on several first-order problem datasets yielded that a combination of cut on reduction steps and exclusive cut on extension steps (REX) improves the number of solved problems compared to the previous best strategy by up to 19%.

Acknowledgements I am grateful to Jasmin Blanchette, Mathias Fleury, and Petar Vukmirović for their comments on drafts of this paper. This work has been supported by the Schrödinger grant (J 4386) of the Austrian Science Fund (FWF).

Bibliography

[1] Andrews, P.B.: Theorem proving via general matings. J. ACM 28(2), 193–214 (1981), https://doi.org/10.1145/322248.322249

[2] Bibel, W.: Automated theorem proving, 2nd Edition. Artificial intelligence, Vieweg (1987), https://www.worldcat.org/oclc/16641802

[3] Biere, A., Dragan, I., Kovács, L., Voronkov, A.: Experimenting with SAT solvers in Vampire. In: Gelbukh, A.F., Castro-Espinoza, F., Galicia-Haro, S.N. (eds.) Human-Inspired Computing and Its Applications - 13th Mexican International Conference on Artificial Intelligence, MICAI 2014, Tuxtla Gutiérrez, Mexico, November 16-22, 2014. Proceedings, Part I. Lecture Notes in Computer Science, vol. 8856, pp. 431–442. Springer (2014), https://doi.org/10.1007/978-3-319-13647-9_39

[4] Färber, M., Kaliszyk, C.: Certification of nonclausal connection tableaux proofs. In: Cerrito, S., Popescu, A. (eds.) Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11714, pp. 21–38. Springer (2019), https://doi.org/10.1007/978-3-030-29026-9_2

[5] Färber, M., Kaliszyk, C., Urban, J.: Machine learning guidance for connection tableaux. J. Autom. Reasoning 65, 287–320 (2021), https://doi.org/10.1007/s10817-020-09576-7

[6] Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6803, pp. 299–314. Springer (2011), https://doi.org/10.1007/978-3-642-22438-6_23

[7] Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Archer, M., Vito, B.D., Muñoz, C. (eds.) Design and Application of Strategies/Tactics in Higher Order Logics (STRATA). pp. 56–68. No. NASA/CP-2003-212448 in NASA Technical Reports (Sep 2003), http://www.gilith.com/research/papers

[8] Kaliszyk, C.: Efficient low-level connection tableaux. In: de Nivelle, H. (ed.) Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9323, pp. 102–111. Springer (2015), https://doi.org/10.1007/978-3-319-24312-2_8

[9] Kaliszyk, C., Urban, J.: FEMaLeCoP: Fairly efficient machine learning connection prover. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Pro-

ceedings. Lecture Notes in Computer Science, vol. 9450, pp. 88–96. Springer (2015), https://doi.org/10.1007/978-3-662-48899-7_7

[10] Kaliszyk, C., Urban, J., Michalewski, H., Olsák, M.: Reinforcement learning of theorem proving. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 8836–8847 (2018), http://papers.nips.cc/paper/8098-reinforcement-learning-of-theorem-proving

[11] Kaliszyk, C., Urban, J., Vyskocil, J.: Certified connection tableaux proofs for HOL light and TPTP. In: Leroy, X., Tiu, A. (eds.) Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India, January 15-17, 2015. pp. 59–66. ACM (2015), https://doi.org/10.1145/2676724.2693176

[12] Kovács, L., Voronkov, A.: First-order theorem proving and Vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 1–35. Springer (2013), https://doi.org/10.1007/978-3-642-39799-8_1

[13] Letz, R., Stenz, G.: Model elimination and connection tableau procedures. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 2015–2114. Elsevier and MIT Press (2001), https://doi.org/10.1016/b978-044450813-3/50030-8

[14] Olsák, M., Kaliszyk, C., Urban, J.: Property invariant embedding for automated reasoning. In: Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (eds.) ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 1395–1402. IOS Press (2020), https://doi.org/10.3233/FAIA200244

[15] Otten, J.: leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5195, pp. 283–291. Springer (2008), https://doi.org/10.1007/978-3-540-71070-7_23

[16] Otten, J.: Restricting backtracking in connection calculi. AI Commun. 23(2-3), 159–182 (2010), https://doi.org/10.3233/AIC-2010-0464

[17] Otten, J.: MleanCoP: A connection prover for first-order modal logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8562, pp. 269–276. Springer (2014), https://doi.org/10.1007/978-3-319-08587-6_20

[18] Otten, J.: nanocop: Natural non-clausal theorem proving. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 4924–4928. ijcai.org (2017), https://doi.org/10.24963/ijcai.2017/695

[19] Otten, J., Bibel, W.: leanCoP: lean connection-based theorem proving. J. Symb. Comput. 36(1-2), 139–161 (2003), https://doi.org/10.1016/S0747-7171(03)00037-3

[20] Paskevich, A.: Connection tableaux with lazy paramodulation. J. Autom. Reason. 40(2-3), 179–194 (2008), https://doi.org/10.1007/s10817-007-9089-7

[21] Schulz, S.: System description: E 1.8. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8312, pp. 735–743. Springer (2013), https://doi.org/10.1007/978-3-642-45221-5_49

[22] Stroustrup, B.: Foundations of C++. In: Seidl, H. (ed.) Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7211, pp. 1–25. Springer (2012), https://doi.org/10.1007/978-3-642-28869-2_1

[23] Sutcliffe, G.: The 8th IJCAR automated theorem proving system competition - CASC-J8. AI Commun. 29(5), 607–619 (2016), https://doi.org/10.3233/AIC-160709

[24] Sutcliffe, G.: Proceedings of the 10th IJCAR ATP system competition (CASC-J10) (2020), http://www.tptp.org/CASC/J10/Proceedings.pdf

[25] Urban, J., Vyskocil, J., Stepánek, P.: MaLeCoP machine learning connection prover. In: Brünnler, K., Metcalfe, G. (eds.) Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6793, pp. 263–277. Springer (2011), https://doi.org/10.1007/978-3-642-22119-4_21