randoCoP: Randomizing the Proof Search Order in the Connection Calculus

Thomas Raths Jens Otten

Institut für Informatik, University of Potsdam
August-Bebel-Str. 89, 14482 Potsdam-Babelsberg, Germany
{traths,jeotten}@cs.uni-potsdam.de

Abstract. We present randoCoP, a theorem prover for classical first-order logic, which integrates randomized search techniques into the connection prover leanCoP 2.0. By randomly reordering the axioms of the problem and the literals within its clausal form, the incomplete search variants of leanCoP 2.0 can be improved significantly. We introduce details of the implementation and present comprehensive practical results by comparing the performance of randoCoP with leanCoP and other theorem provers on the TPTP library and problems involving large theories.

1 Introduction

Connection calculi, such as the connection calculus [1, 2], the connection tableau calculus [8], or the model elimination calculus [9], are in contrast to standard tableau or saturation-based calculi not proof confluent. Therefore a large amount of backtracking is required during the proof search. By restricting this backtracking the performance of connection-based proof search procedures can be improved significantly [13]. leanCoP 2.0 [15, 14] is a theorem prover for classical first-order logic based on the connection calculus. A shell script consecutively runs different variants of the core prover with different options, which control the proof search. The most successful variants use restricted backtracking.

The downside of restricted backtracking is the loss of completeness. Whereas proofs for some formulae can be found very quickly, it might be impossible to find proofs for other formulae anymore. Since restricted backtracking cuts off alternative connections, the benefit of this approach strongly depends on the proof search order. The proof search order, in turn, usually depends on the order of clauses and literals in the given formula. Whereas the proof search procedure quickly finds a proof for one order, another order of the same clauses might result in an incomplete proof search order, i.e. no proof is found at all. By reordering the clauses, the downside of restricted backtracking can be minimized.

randoCoP extends the leanCoP 2.0 implementation by repeatedly reordering the axioms and literals of a given problem at random. This increases the chance to find a proof, in particular for the incomplete prover variants. In Section 2 we present experimental results of several reordering techniques and details of the implemented reordering strategy. In Section 3 the performance of randoCoP is compared with current state-of-the-art theorem provers.

2 Randomizing the Proof Search Order

We first describe the basic motivation behind the randomized reordering technique. Afterwards we present a detailed practical analysis, which determines the specific reordering strategy that is used within randoCoP.

2.1 Motivation

leanCoP searches for connections in the order of the given input clauses. Connections to clauses at the beginning of the input clauses are considered first, before (on backtracking) clauses at the end are examined. Therefore reordering clauses is a simple way of modifying the proof search order. But the effect of reordering clauses is limited for complete search strategies (see Section 2.2), since every clause is considered sooner or later anyway.

leanCoP 2.0 has the option to restrict backtracking by cutting off alternative connections in branches that have already been closed before [13]. With this incomplete search technique some clauses might not be considered anymore and the reordering of clauses has a significant effect. It makes it possible to find proofs for problems that could not be solved before. Figure 1 illustrates this fact. The triangle represents the search space, the crosses mark the solutions, and the grey shaded area is the search space that can be traversed within a certain time limit and is roughly the same for all three triangles.

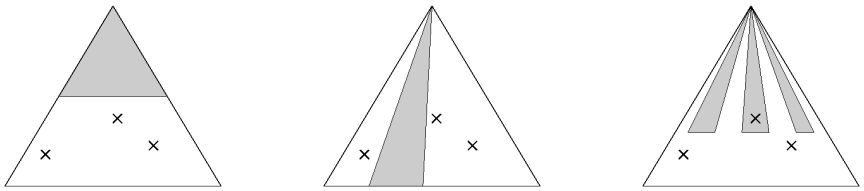

Fig. 1. Search strategies: complete, restricted backtracking without/with reordering

The complete search strategy (left hand side), does not reach the proof depth required for a solution. The search with restricted backtracking (in the middle), reaches the depth of the solutions but not the required breadth. Only the search strategy with restricted backtracking and repeatedly reordering the clauses (right hand side) is able to find a proof.

Clauses can be reordered in several ways, e.g. by rotating or shifting clauses. leanCoP 2.0 already contains an option for reordering clauses that uses a simple perfect shuffle algorithm. But the effect on the proof search is small, since the generated clause orders are not sufficiently diverse. A random reordering mixes the order of clauses more thoroughly. Therefore instead of the built-in reordering technique a randomized reordering should be used. Since the outcome of the proof search needs to be reproducible, a deterministic pseudo-random reordering needs to be implemented.

2.2 Evaluation and Analysis

Our practical experiments have shown that a reordering of the axioms of a given problem is more effective than the reordering of clauses after the clausal form translation has been applied. These experiments also showed that reordering the literals within the clauses has a positive effect on the proof search as well.

The proof search order is randomly modified in the following way. Let $A_1 \wedge A_2 \wedge \ldots A_n \wedge \Rightarrow C$ be the given formula where $A_1, A_2, \ldots, A_n$ are the axioms and C is the conjecture of the problem. Let $\pi^m : \{1, \ldots, m\} \rightarrow \{1, \ldots, m\}$ be a (pseudo-)random permutation.

1. *Reordering of axioms*: The following formula is generated: $A_{\pi^n(1)} \wedge A_{\pi^n(2)} \wedge \ldots \wedge A_{\pi^n(n)} \Rightarrow C$. Let $C_1, C_2, \ldots, C_m$ be this formula in clausal form.
2. *Reordering of literals*: For each clause C_i=$\{L_1, \ldots, L_k\}$ the reordered clause C'_i=$\{L_{\pi^k(1)}, \ldots, L_{\pi^k(k)}\}$ is generated. $C'_1, \ldots, C'_k$ is the final set of clauses.

We first want to find the options of the leanCoP 2.0 core prover that are most suited for our randomized reordering approach. A *variant* of the leanCoP 2.0 Prolog core prover is determined by a set of options that control the proof search [14]. The following options can be used:

1. `nodef`/`def`: The standard/definitional translation into clausal form is done. If none of these options is given the standard translation is used for the axioms whereas the definitional translation is used for the conjecture [13].
2. `conj`: The conjecture clauses are used as start clauses. If this option is not given the positive clauses are used as start clauses.
3. `scut`: Backtracking is restricted for alternative start clauses [13].
4. `cut`: Backtracking is restricted for alternative reduction/extension steps [13].
5. `comp(I)`: Restricted backtracking is switched off when iterative deepening exceeds the active path length `I`.

The option `conj` is complete only for formulae with a provable conjecture, and the options `scut` and `cut` are only complete if used in combination with the option `comp(I)`.

The following tests were performed on all 3644 non-clausal problems (FOF division) of the TPTP library [18] version 3.3.0. The left side of Table 1 shows the number of proved problems without reordering using a time limit of 180 seconds. The right side displays the number of proved problems with repeated reordering. The allotted time limit for each order is 3 seconds with an overall time limit of 180 seconds allowing around 60 reorderings for each problem.

The prover variants using restricted backtracking (options `cut` and `scut`) benefit most from the randomized reordering technique. The greatest advance is made for the variant using the default clausal form translation and both of the options `cut` and `scut`. The variants using a full definitional (`def`) or the standard (`nodef`) clausal form translation and no restricted backtracking show a lower performance. The most successful variants using reordering use the options {`cut`}, {`cut`,`scut`}, {`conj`,`cut`}, and {`conj`,`cut`,`scut`}.

Table 1. Results without and with repeated reordering for different prover variants

options	-	cut	cut,scut	options	-	cut	cut,scut
-	1217	1346	1216	-	1342	1691	1684
def	1111	1321	1204	def	1044	1402	1428
nodef	1166	1257	1152	nodef	1077	1432	1436
conj	1208	1398	1336	conj	1422	1658	1597
conj,def	1208	1407	1349	conj,def	1095	1452	1370
conj,nodef	1227	1319	1058	conj,nodef	1152	1365	1239

We have tested the most successful variants again using different time limits for each order of the axioms/literals. Table 2 shows the results using an overall time limit of 180 seconds and a time limit of 2 seconds, 3 seconds and 4 seconds for proving each order allowing around 90, 60 and 45 reorderings, respectively. The difference in the number of proved problems is rather small with a slight advantage for a time limit of 3 seconds.

Table 2. Results of different time limits per axiom/literal order

options \ time	$2s$	$3s$	$4s$
cut	1678	1691	1688
cut,scut	1670	1684	1650
conj,cut	1636	1658	1665
conj,cut,scut	1569	1597	1607

Further evaluations have shown that the most number of problems are proved by repeatedly running the two variants {cut,scut} and {def,conj,cut} each for a time limit of 3 seconds and 2 seconds on each axiom/literal order, respectively.

In order to prove and refute a large number of problems within the first few seconds, the proof process is started by running the most successful complete prover variant using the options cut and comp(7) once for 5 seconds. A complete prover variant using the option def only is invoked at the end of the proof process for at least 10 seconds.

2.3 The Implementation

randoCoP uses the random library of ECLiPSe Prolog, which contains the predicate rand_perm/2 that randomly permutes a list, and randomise/1 to seed and initialize the random generator, which allows the same sequence of permutations to be generated, and thus makes the result reproducible.

randoCoP consists of a shell script that calls the leanCoP 2.0 core prover extended by a few predicates realizing the reordering of axioms and literals. This shell script is called with an argument that determines the overall time limit and invokes the prover variants according to Section 2.2.

Let $TotalTime$ be this given time limit in seconds. Then the shell script invokes the following variants of the core prover in the following order:

1. Prover variant {`cut`,`comp(7)`} for 5 seconds
2. Repeated reordering step, $(TotalTime - 15)/5$ times:
   (a) Reordering of axioms and literals
   (b) Prover variant {`cut`,`scut`} for 3 seconds
   (c) Prover variant {`def`,`conj`,`cut`} for 2 seconds
3. Prover variant {`def`} for 10 seconds

If, for example, $TotalTime$ is set to 600 seconds, then (600-15)/5=117 reordering steps are done and after each reordering the two prover variants are run for 3 seconds and 2 seconds, respectively.

3 Performance

To evaluate the performance we tested randoCoP on all non-clausal problems in the TPTP library, and the problems of the MPTP challenge. All tests were done on a system with a 3 GHz Xeon processor and 4 Gbyte of memory running Linux and ECLiPSe Prolog 5.8. We compare the performance of randoCoP with the current state-of-the-art provers.

3.1 The TPTP Library

For the tests all 3644 problems of version 3.3.0 of the TPTP library [18] that are in non-clausal form (FOF division) are considered. In order to solve satisfiable or unsatisfiable problems — i.e., those problems without a conjecture — these problems are negated. Equality is dealt with by adding the equality axioms. The time limit for all problems is 600 seconds.

In Table 3 the performance of randoCoP is compared with the following theorem provers: Otter 3.3 [10], version "20070805r009" of SNARK [19], leanCoP 2.0 [14], version "Dec-2007" of Prover9[1] [11], iProver 0.2 [5], Equinox 1.2 [3], SPASS 3.0 [23], E 0.999 [17], and Vampire 9.0 [16]. The rating and the percentage of proved problems for some rating intervals are given. FNE, FEQ and PEQ are problems without, with and containing only equality, respectively. Furthermore, the number of proved problems for each domain (see [18]) that contains at least 10 problems is shown.

randoCoP proves more problems than Equinox, iProver, Prover9, leanCoP 2.0, SNARK and Otter. It solves more problems in the SEU domain than any other prover. The SEU domain contains problems from set theory taken from the MPTP [20] (see also Section 3.2). Including the equality axioms, these problems contain up to 1100 axioms, of which not all are required to prove the conjecture. For the SEU domain randoCoP also shows the biggest improvement compared to leanCoP 2.0, with 491 proved problems compared to 296 problems proved by leanCoP 2.0. A notable improvement is also made for the domains NUM and SWC. 96 of all proved problems have the highest rating of 1.0.

[1] The most recent version "2008-04A" of Prover9 has a significant lower performance.

Table 3. Benchmark results on the TPTP v3.3.0 library

	Otter 3.3	SNARK 08/07	leanCoP 2.0	Prover9 12/07	iProver 0.2	Equinox 1.2	randoCoP 1.0	SPASS 3.0	E 0.999	Vampire 9.0
proved	1310	1565	1638	1677	1858	1876	**1879**	2127	2250	2377
[%]	36%	43%	45%	46%	51%	52%	**52%**	58%	62%	66%
0s to 1s	987	1259	1124	1281	1224	1376	**1161**	1627	1760	1530
1s to 10s	183	130	123	197	370	239	**229**	248	229	283
10s to 100s	106	117	193	141	179	151	**375**	170	192	394
100s to 600s	34	59	198	58	85	110	**114**	82	69	170
rating 0.0	455	435	450	464	473	500	**452**	500	503	488
rating >0.0	855	1130	1188	1213	1385	1376	**1427**	1627	1747	1889
rating 1.0	7	8	13	8	9	18	**96**	19	12	30
0.00...0.24	72%	72%	72%	73%	76%	76%	**73%**	78%	79%	78%
0.25...0.49	40%	62%	48%	70%	73%	72%	**52%**	84%	85%	86%
0.50...0.74	3%	20%	36%	28%	43%	40%	**43%**	58%	71%	80%
0.75...1.00	1%	2%	11%	3%	7%	9%	**27%**	15%	19%	27%
FNE	476	437	492	526	562	541	**496**	555	561	566
FEQ	834	1128	1146	1151	1296	1335	**1383**	1572	1689	1811
PEQ	47	83	30	71	76	196	**31**	191	165	133
AGT	16	17	29	17	19	10	**32**	18	19	52
ALG	60	84	32	83	80	186	**38**	196	162	130
CSR	3	16	2	27	10	15	**3**	2	25	27
GEO	160	121	171	171	172	167	**169**	168	174	177
GRA	5	9	6	1	8	9	**6**	15	15	19
KRS	106	110	105	103	112	111	**105**	107	112	112
LCL	18	56	24	48	33	10	**25**	51	78	101
MED	5	2	7	1	9	5	**5**	9	8	9
MGT	54	58	45	62	65	67	**50**	56	67	67
NLP	6	11	13	11	22	22	**15**	22	22	22
NUM	31	45	70	49	43	45	**87**	48	62	105
PUZ	6	6	6	6	6	6	**6**	6	6	6
SET	229	262	339	276	316	245	**335**	290	339	369
SEU	149	229	296	259	291	305	**491**	353	316	347
SWC	84	131	87	101	170	175	**99**	256	326	297
SWV	157	181	177	183	210	221	**181**	235	225	236
SYN	210	214	217	267	277	265	**218**	279	276	281
refuted	0	113	33	0	329	0	**33**	362	366	0
time out	700	1596	1949	668	1455	1395	**1732**	1125	1023	985
gave up	1181	213	0	320	0	373	**0**	0	0	280
errors	453	157	24	979	2	0	**0**	30	5	2

3.2 The MPTP Challenge

The MPTP challenge is a set of problems from the Mizar library translated into first-order logic [20]. There are two divisions, bushy and chainy, each containing 252 problems. Whereas the bushy division contains only the relevant axioms and lemmata required to prove the main theorem, the chainy division contains all axioms and lemmata that were available at the time of proving the main theorem of the challenge. The time limit for each problem is 300 seconds.

The result of randoCoP on the bushy and chainy division is shown in Table 4 and Table 5, respectively. Again, equality axioms are added, which results in formulae with a total of up to 1700 axioms. The performance is compared with the theorem provers mentioned in Section 3.1.

Table 4. Benchmark results for the bushy division of the MPTP challenge

	Otter 3.3	Prover9 12/07	SNARK 08/07	leanCoP 2.0	iProver 0.2	Equinox 1.2	E 0.999	SPASS 3.0	Vampire 9.0	randoCoP 1.0
proved	68	119	122	128	128	131	141	160	166	**189**
[%]	27%	47%	48%	51%	51%	52%	56%	64%	66%	**75%**
0s to 1s	56	86	91	93	71	86	120	121	104	**93**
1s to 10s	4	19	17	7	33	23	12	16	22	**23**
10s to 100s	7	9	12	20	20	17	8	17	31	**60**
100s to 300s	1	5	2	8	4	5	1	6	9	**13**
time out	46	82	130	124	124	121	111	90	86	**63**
gave up	137	3	0	0	0	0	0	0	0	**0**
errors	1	48	0	0	0	0	0	2	0	**0**

Table 5. Benchmark results for the chainy division of the MPTP challenge

	Otter 3.3	Prover9 12/07	SNARK 08/07	Equinox 1.2	iProver 0.2	Vampire 9.0	SPASS 3.0	leanCoP 2.0	E 0.999	randoCoP 1.0
proved	29	52	58	79	79	81	82	88	91	**128**
[%]	12%	21%	23%	31%	31%	32%	33%	35%	36%	**51%**
0s to 1s	17	33	22	41	29	29	45	38	47	**38**
1s to 10s	7	8	14	14	37	31	18	21	18	**29**
10s to 100s	5	6	14	17	10	14	15	23	20	**34**
100s to 300s	0	5	8	7	3	7	4	6	6	**27**
time out	150	101	194	173	173	171	170	164	161	**124**
gave up	73	0	0	0	0	0	0	0	0	**0**
errors	0	99	0	0	0	0	0	0	0	**0**

randoCoP shows a decent performance in both division. The time complexity is better compared to the other provers, as many problems are (still) proved after 10 seconds. We have not tested the provers MaLARea 0.1, SRASS 0.1, and Fampire 1.3, which prove 187/142, 171/127, and 191/126 of the problems in the bushy/chainy division, respectively (according to the MPTP challenge web site).

4 Conclusion and Related Work

We have presented randoCoP, a theorem prover for classical first-order logic, which integrates a random proof search strategy into the connection prover leanCoP 2.0. Repeatedly reordering the axioms of the problem and the literals within its clausal form improves performance of leanCoP 2.0 significantly.

Some incomplete strategies of leanCoP 2.0 effectively restrict backtracking and increase the depth of the search space that can be investigated within a certain amount of time. But they might cut off specific proof search orders required to find a proof. randoCoP partly compensates for this disadvantage and the loss of completeness by increasing again the breadth of the explored search space. The combination of restricted backtracking and randomized reordering is highly effective, in particular for hard problems containing many axioms.

The core prover of randoCoP and leanCoP 2.0 consists only of a few lines of Prolog code. This indicates that tens to hundreds of thousands of lines in, e.g., C and low-level optimizations are not needed to succeed in automated theorem proving. Instead it shows that good heuristics for traversing the vast search are important in automated reasoning research.

The RCTHEO system [4] randomly reorders clause instances. It is a an OR-parallel version of SETHEO [7], where each node executes one instance of the sequential prover SETHEO. The performance is similar to PARTHEO, a parallel version of SETHEO.

The SETHEO system offers a dynamic subgoal reordering option. The reordering is not at random but prefers subgoals with the highest probability to fail, in order to reduce the search space. Syntactic criteria, such as the number of variables, are used to determine the specific order. The subgoal order is then determined dynamically whenever the next subgoal is selected.

We have tested SETHEO without and with subgoal reordering on all non-clausal problems of the TPTP library (see Section 3.1). *Without* subgoal reordering SETHEO proves 1192 out of the 3644 problems. *With* subgoal reordering (using the option -dynsgreord 2) it only solves 1185 problems, i.e. the performance does not really improve. This confirms our own testing with reordering on several variants of the leanCoP 2.0 core prover (see Section 2.2). The effect of reordering axioms (or clauses) and literals is limited when a complete search in connection calculi is done. In this case the performance can even get worse. On the other hand the incomplete variants of the leanCoP 2.0 core prover benefit significantly from the randomized reordering technique.

Further research includes the adaption of randoCoP to other (non-classical) logics — such as intuitionistic logic [12, 14] or modal logic [6] — for which matrix characterisations exist (see also [21, 22]). It is also worth investigating approaches that randomize, e.g., the order of the conjecture clauses, or dynamically reorders clauses and/or literals *during* the proof search. And finally we plan to (slightly) extend the leanCoP 2.0 core prover so that a compact connection proof is returned. A readable proof is then output by a separate prover component.

The source code of randoCoP can be obtained at the leanCoP website at `http://www.leancop.de`.

Acknowledgements. The authors would like to thank the referees for their useful comments, which have helped to improve this paper.

References

1. W. Bibel. Matings in matrices. *Communications of the ACM*, 26:844–852, 1983.
2. W. Bibel. *Automated Theorem Proving*. Vieweg, second edition, 1987.
3. K. Classen. Equinox, a new theorem prover for full first-order logic with equality. In *Dagstuhl Seminar 05431 on Deduction and Applications*, 2005.
4. W. Ertel. OR-parallel theorem proving with random competition. In A. Voronkov, Ed., *LPAR'92*, LNAI 624, pp. 226–237. Springer, 1992.
5. K. Korovin. Implementing an instantiation-based theorem prover for first-order logic. In C. Benzmueller, B. Fischer, G. Sutcliffe *IWIL-6*, 2006.
6. C. Kreitz, J. Otten. Connection-based theorem proving in classical and non-classical logics. *Journal of Universal Computer Science*, 5:88–112, Springer, 1999.
7. R. Letz, J. Schumann, S. Bayerl, W. Bibel. SETHEO: a high-performance theorem prover. *Journal of Automated Reasoning*, 8:183–212, 1992.
8. R. Letz, G. Stenz Model elimination and connection tableau procedures. *Handbook of Automated Reasoning*, pp. 2015–2114, Elsevier, 2001.
9. D. Loveland. Mechanical theorem proving by model elimination. *JACM*, 15:236–251, 1968.
10. W. McCune. Otter 3.0 reference manual and guide. Technical Report ANL-94/6, Argonne National Laboratory, 1994.
11. W. McCune. Release of Prover9. *Mile High Conference on Quasigroups, Loops and Nonassociative Systems*, Denver, Colorado, 2005.
12. J. Otten. Clausal connection-based theorem proving in intuitionistic first-order logic. *TABLEAUX 2005*, LNAI 3702, pages 245–261, Springer, 2005.
13. J. Otten. Restricting backtracking in connection calculi. Technical report, Institut für Informatik, University of Potsdam, 2008.
14. J. Otten. leanCoP 2.0 and ileanCoP 1.2: high performance lean theorem proving in classical and intuitionistic logic. *IJCAR 2008*. LNCS, Springer, 2008.
15. J. Otten, W. Bibel. leanCoP: lean connection-based theorem proving. *Journal of Symbolic Computation*, 36:139–161, 2003.
16. A. Riazanov, A. Voronkov. The design and implementation of Vampire. *AI Communications* 15(2-3): 91–110, 2002.
17. S. Schulz. E - a brainiac theorem prover. *AI Communications*, 15(2):111–126, 2002.
18. G. Sutcliffe, C. Suttner. The TPTP problem library - CNF release v1.2.1. *Journal of Automated Reasoning*, 21: 177–203, 1998.
19. M. Stickel, R. Waldinger, M. Lowry, T. Pressburger, I. Underwood. Deductive composition of astronomical software from subroutine libraries. In A. Bundy, Ed., *CADE-12*, LNCS 814, pp. 341–355, Springer, 1994.
20. J. Urban. MPTP 0.2: design, implementation, and initial experiments. *Journal of Automated Reasoning*, 37:21–43, 2006.
21. A. Waaler. Connections in nonclassical logics. *Handbook of Automated Reasoning*, pp. 1487–1578, Elsevier, 2001.
22. L. Wallen. *Automated deduction in nonclassical logic*. MIT Press, 1990.
23. C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, D. Topic. System description: SPASS version 3.0. In F. Pfenning, Ed., *CADE-21*, LNCS 4603, pp. 514–520. Springer, 2007.