A Constraint Programming Approach to Log-based Reconciliation Problems for Nomadic Applications

François Fages

Projet Contraintes, INRIA-Rocquencourt,
BP105, 78153 Le Chesnay Cedex, France,
francois.fages@inria.fr

Abstract. Nomadic applications create replicas of shared objects that evolve independently while they are disconnected. When reconnecting, the system has to reconcile the divergent replicas. Log-based reconciliation is a novel approach in which the input is a common initial state and logs of actions that were performed on each replica. The output is a consistent global schedule that maximises the number of accepted actions. The reconciler merges the logs according to the schedule, and replays the operations in the merged log against the initial state, yielding to a reconciled common final state.

In this paper, we show the NP-hardness of the log-based reconciliation problem, and study a constraint logic program that uses integer constraints for expressing precedences between actions, and boolean constraints for expressing dependencies between actions. Interestingly, we demonstrate the existence of a single computational complexity peak on randomly generated problems, around densities 7 for precedence constraints and 0 for dependency constraints between actions. Around this peak we observe phase transitions in the two dimensions of the density, where the mean running time of the program shifts from polynomial in the order to exponential. On realistic benchmarks, first evaluation results show that the program finds nearly optimal solutions up to a thousands of actions and proves optimality up to a hundred of actions.

1 Introduction

Data replication is a standard technique in distributed systems to make data available in different sites. The different sites may be disconnected (mobile computing) or connected (groupware) in which case shared data are replicated in order to enable independent work. Obviously the replication of mutable shared data may cause conflicts, the replicas may diverge into inconsistent states that have thus to be reconciled. Nomadic applications create replicas of shared objects that evolve independently while they are disconnected, when reconnecting the system has to reconcile the divergent replicas. The purpose of a reconciler is to detect and resolve conflicts in order to restore a consistent state.

What constitutes a conflict depends on the semantics of the application and on the user's intent. For example, in the version management system CVS [10], write actions are said to conflict if and only if they occur in the same line of the same text file. Accordingly, many existing reconcilers are restricted to specific data types, such as source files, or file systems [3] or calendars. In contrast, in the log-based approach to reconciliation [16, 9, 14], the input is a common initial state and logs of actions that were performed on each replica. In this novel setting, an action is composed of a precondition, an operation and a postcondition [18]. The output is a consistent global schedule that maximises the number of accepted actions [14]. The reconciler merges the logs according to the schedule, and replays the operations in the merged log against the initial state, yielding to a reconciled common final state [14].

In this paper, we show the NP-hardness of the log-based reconciliation problem, by encoding SAT as a reconciliation problem, and propose a constraint logic program for solving this problem. In section 3 we present a CLP program based on a simple modeling using boolean constraints for expressing dependencies between actions, constraints over integers for expressing precedence constraints, and some heuristics for guiding the search during branch-and-bound optimization. We evaluate the performance of this program on a set of randomly generated benchmarks, with various *densities*, defined as the ratio between the number of constraints and the number of variables, called the *order* of the instance. Interestingly, we demonstrate the existence of a single computational complexity peak around a point of density 7 for precedence constraints and density 0 for dependency constraints between actions, for all orders. The dependency constraints have for effect to group variables into equivalence classes which in effect decrease the number of different variables. This makes the mean time complexity quickly decreases from exponential to polynomial, with a density threshold value less than 2. In the other dimension, along the density axis for precedence constraints, at the hardness peak of density 0 for dependency constraints, the average-case complexity jumps from polynomial to exponential at a density threshold value less than 2. It reaches its maximum at a density threshold value around 7, and then decreases but remains exponential until the density approaches the value of the order. Similarly, we measure the mean-time complexity of the program for finding a first solution and show that it is polynomial in the order and does not contain computational complexity peak. Finally we develop a set of benchmarks intended to model more realistic log-based reconciliation problems, using densities (1.5, 1.5). For these problems, the program finds quasi-optimal solutions up to a thousands of actions, and proves optimality up to a hundred of actions.

Related work

The complexity of randomly generated combinatorial problems has been the subject of a large body of theoretical and experimental work, since the discovery of an intriguing connection between the density of combinatorial problems, the rapid transition in their satisfiability, and their computational complexity [4]. In random 3-SAT, the phase transition has been shown to occur at density 4.26.

Below that density threshold value the probability of satisfiability goes to 1 as the order increases, while it goes to 0 above 4.26. At the crossover point the probability is 0.5. The value of the crossover point, first established experimentally, is the subject of a continuing research aiming at providing accurate formal lower and upper bounds borrowing techniques from various disciplines [8, 2]. The crossover point corresponds also to a peak in running time for 3-SAT solvers and is the center of an "easy-hard-easy" pattern [17, 12]. In [5], an interesting systematic study of three complete SAT solvers shows that, while the hardest problems are indeed around the crossover point, the shape of the mean-time[1] complexity as a function of the density and the order, is solver dependent. Different phase transitions where the complexity jumps from polynomial to exponential depend on the search methods.

In this perspective on randomly generated combinatorial problems, the log-based reconciliation problem appears as a two-dimensional problem: as we have two kinds of constraints over boolean and integer variables, it is interesting not to confuse these constraints and consider the coupling of both densities for each instance. We show that the average-case complexity of the CLP program, as a function of both densities, has nevertheless a simple shape with a single peak around the point of densities (0,7) (see figure 1 and 2), and that this shape is basically preserved when the order increases (see figure 3).

As log-based reconciliation is an optimization problem, it currently seems difficult to relate the densities of the hardness peak to crossover points in the associated satisfiability problems. Instead, we study other phase transitions, where the complexity of the CLP solver jumps from polynomial to exponential, and we provide experimental evidence for the polynomial average-case complexity of finding the first solution, for all densities (see figure 7).

On the application side, log-based reconciliation is a new topic for which few algorithms have been developed [14]. It is worth noting that the objective function of maximizing the number of accepted actions, is different from maximizing the number of satisfied constraints. For that reason, the modeling of log-based reconciliation as a max-CSP problem is inadequate.

2 The log-based reconciliation problem

2.1 Statement of the optimization problem

We have to reconcile a set of logs of actions that have been realized independently, by trying to accept the greatest number of actions as possible.

Input: A finite set of L initial logs of actions $\{[T_i^1, ..., T_i^{n_i}] \mid 1 \leq i \leq L\}$, some dependencies between actions $T_i^j \Rightarrow T_k^l$, meaning that if T_i^j is accepted then T_k^l must be accepted, and some precedence constraints $T_i^j < T_k^l$, meaning that if

[1] The authors use in fact the median running time as an experimental surrogate of the mean running time. We do not use this technique in this paper although it could provide estimations for some larger instances.

the actions are accepted they must be executed in that order. The precedence constraints are supposed to be satisfied inside the initial logs.

Output: A subset of accepted actions, of maximal cardinality, satisfying the dependency constraints, given with a global schedule $T_i^j < ... < T_k^l$ satisfying the precedence constraints.

Note that the output depends solely of the precedence constraints between actions given in the input. In particular it is independent of the precise structure of the initial logs. The initial consistent logs can thus be used as starting solutions in some algorithms but can be forgotten as well without affecting the output.

2.2 Complexity

Proposition 1. *The decision problem, i.e. finding a schedule of a given length, is NP-complete, even without dependency constraints.*

Proof. The decision problem is obviously in NP. Indeed, for any guessed schedule, one can check in polynomial time whether the schedule is consistent.

NP-completeness is shown by encoding SAT into a reconciliation problem with singleton initial logs and precedence constraints only.

Let us assume a SAT problem over N boolean variables with C clauses. For each boolean variable p, we associate $2 * C$ actions p_0^1, p_1^1, ..., p_0^C, p_1^C, with precedence constraints $p_0^i < p_1^j$ and $p_1^j < p_0^i$ for all clause indices i, j in $[1, C]$. The actions p_0^i and p_1^j are thus mutually exclusive for all clause indices i, j. We represent the valuation false for p by accepting the actions p_0^i for all $1 \leq i \leq C$, and the valuation true by accepting p_1^j for all $1 \leq i \leq C$. This defines a one-to-one mapping σ between valuations over N boolean variables, and the accepted actions in schedules of length $N * C$ satisfying the mutual exclusion constraints.

For each clause, such as $p \vee q \vee \neg r$, we associate the precedence constraints $p_0^i < q_0^i < r_1^i < p_0^i$ where i is the index of the clause. Being cyclic, these precedence constraints forbid to take simultaneously the actions p_0^i, q_0^i, r_1^i and p_0^i, that is, they encode the equivalent formula $\neg(\neg p \wedge \neg q \wedge r)$. Hence a valuation η satisfies a clause if and only if the actions in $\sigma(\eta)$ satisfy all the precedence constraints associated to the clause. Note that, unlike the mutual exclusion constraints, the precedence constraints are posted between action variables with the same clause index only.

Now we prove that a set of C clauses over N variables is satisfiable if and only if there exists a schedule accepting $N * C$ actions and satisfying the mutual exclusion constraints and the precedence constraints associated to the clauses.

The implication is clear: if η is a valuation which satisfies all the clauses, then $\sigma(\eta)$ is a set of $N * C$ actions which satisfies the mutual exclusion constraints, and which can be ordered with increasing clause indices and according to the precedence constraints for variables with the same clause index.

For the converse, let us suppose that we have a consistent schedule of $N * C$ actions. Because of the mutual exclusion constraints, the schedule defines a valuation of the SAT problem: indeed for each propositional variable p, either

p_O^i is accepted for all i, and p is false, either p_1^i is accepted for all i, and p is true. Furthermore the precedence constraints between actions of index i establish that that valuation satisfies the ith clause. Therefore the valuation associated to the schedule satisfies all the clauses. QED.

Below we present a simple constraint logic program for solving this problem and use this program to demonstrate the existence of a peak of computational complexity on randomly generated problems for some densities of dependency and precedence constraints.

3 A CLP(FD,B) approach

3.1 Modeling with mixed boolean and integer constraints

In this modeling of the problem, we do not use the structure of the initial logs of actions. We thus simply consider the set of all actions given with their precedence constraints. We have n elementary actions to which we associate:

- n boolean variables $a_1, ..., a_n$ which say whether the action is accepted or not,
- n integer variables $p_1, ..., p_n$ which give the position of the action in the global schedule if the action is accepted, and which are undefined otherwise.

We have some dependency constraints

$$a_i \Rightarrow a_j$$

and some precedence constraints

$$a_i \wedge a_j \Rightarrow (p_i < p_j)$$

or equivalently, assuming false is 0 and true is 1,

$$a_i * a_j * p_i < p_j$$

We want to maximize $a_1 + ... + a_n$.

The search for solutions goes through an enumeration of the boolean variables a_i's, with the heuristics of instantiating first the variable a_i which has the greatest number of constraints on it (i.e. first-fail principle w.r.t. the number of posted constraints) and trying first the value 1 (i.e. best-first search for the maximization problem).

This leads to the following simple CLP(FD,B) program, given in GNU-Prolog syntax:

```
solve(Transactions,Dependencies,Precedences,Schedule) :-
        length(Transactions,N),
        length(La,N), fd_domain_bool(La),
        length(Lp,N), fd_domain(Lp,1,N),
```

```
        dependencies(Dependencies, Transactions, La),
        precedences(Precedences, Transactions, La, Lp),
        sum(La,S),
        fd_maximize(
           fd_labeling(La,[variable_method(most_constrained),
                           reorder(true),
                           value_method(max)]),
           S),
        fd_labeling(Lp,[value_method(min)]),
        schedule(La, Lp, Transactions, Keysort),
        sort(Keysort,Schedule).

dependencies([],_,_).
dependencies([(X#==>Y)|L],T,La):-
        nth(I,T,X), nth(I,La,A),
        nth(J,T,Y), nth(J,La,B),
        A#==>B,
        dependencies(L,T,La).

precedences([],_,_,_).
precedences([(X#<Y)|L],T,La,Lp):-
        nth(I,T,X), nth(I,La,A), nth(I,Lp,P),
        nth(J,T,Y), nth(J,La,B), nth(J,Lp,Q),
        A*B*P#<Q,
        precedences(L,T,La,Lp).

sum([],0).
sum([B|L],S):- S#=B+R, sum(L,R).

schedule([],[],[],[]).
schedule([B|La],[P|Lp],[T|Tr],S):-
        ((B=0)
        -> schedule(La,Lp,Tr,S)
        ;  S=[(P-T)|R], schedule(La,Lp,Tr,R)).
```

Note that the labeling done in the optimization predicate proceeds through the boolean variables only. It is well known indeed that interval propagation algorithms provide indeed a complete procedure for checking the satisfiability of precedence constraints. It is thus not necessary to enumerate the possible values of the position variables in the schedule, as we know that the earliest dates are consistent. The labeling of the positions is done outside the optimization predicate, just to compute a ground schedule by taking the earliest dates for each action, without backtracking.

3.2 Hardness peak for the optimization problem

The CLP program has been evaluated on randomly generated problems. The computation times reported in the figures below are the cumulated computation

times for ten runs obtained with randomly generated problems of each characteristics. They thus provide an approximation of the mean running time. They have been measured in SICStus-Prolog on a Laptop computer Compaq Armada 700 MHz under Windows 98 and are given in milliseconds.

Figure 1 depicts a peak in running times for finding the optimal solution and proving optimality, around density 7 for precedence constraints and density 0 for dependency constraints. Figure 2 shows these variations more accurately using a logarithmic scale for the running time. This figure shows that the dependency constraints quickly make the problems easy with a phase transition "hard-easy" between densities 1 and 2 for dependency constraints. In the other dimension, along the density 0 for precedence constraints, a phase transition "easy-hard" occurs at a density less than 2 for precedence constraints. The peak of complexity is at density 7, then the computational complexity slowly decreases for densities of precedence constraints greater than 7. The figure shows however that the exponential time complexity remains far beyond the peak for densities greater than 7. This indicates that there does not exist another phase transition "hard-easy" before the very end when the density of precedence constraints approaches the value of the order.

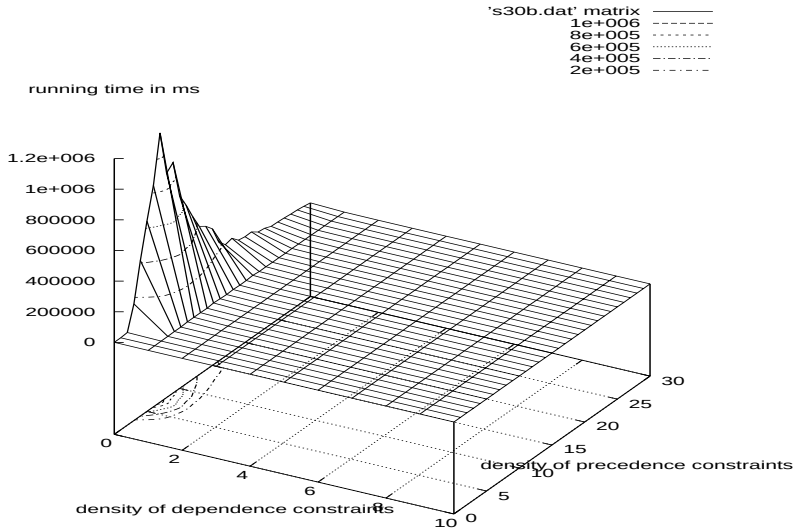

Fig. 1. Hardness peak at densities (0,7) at order 30.

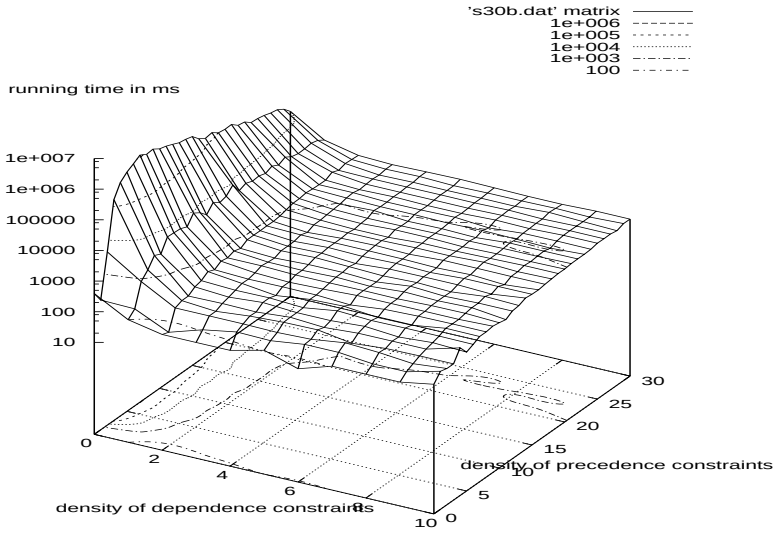

Fig. 2. Hardness peak on a logarithmic scale at densities (0,7) at order 30.

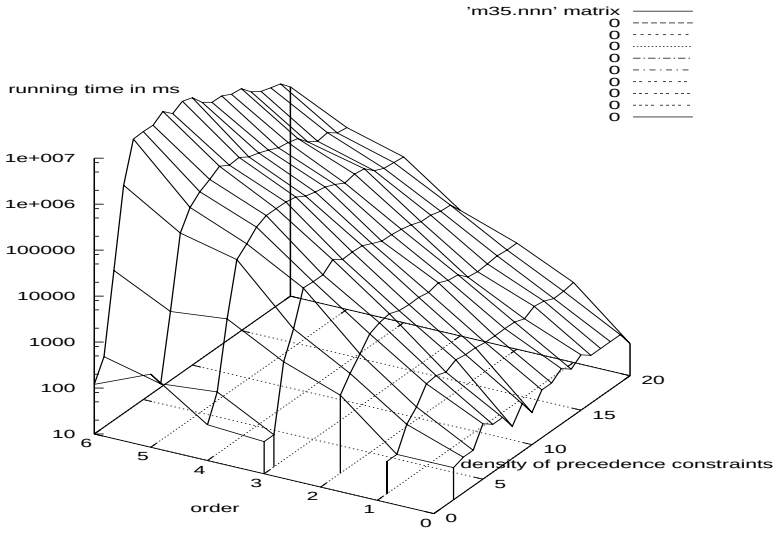

Fig. 3. Mean running time on a logarithmic scale, as a function of the order ranging from 5 to 35 (scaled 0 to 6), around the hardness peak: the density of dependency constraints is 0 and the density of precedence constraints varies from 0 to 20.

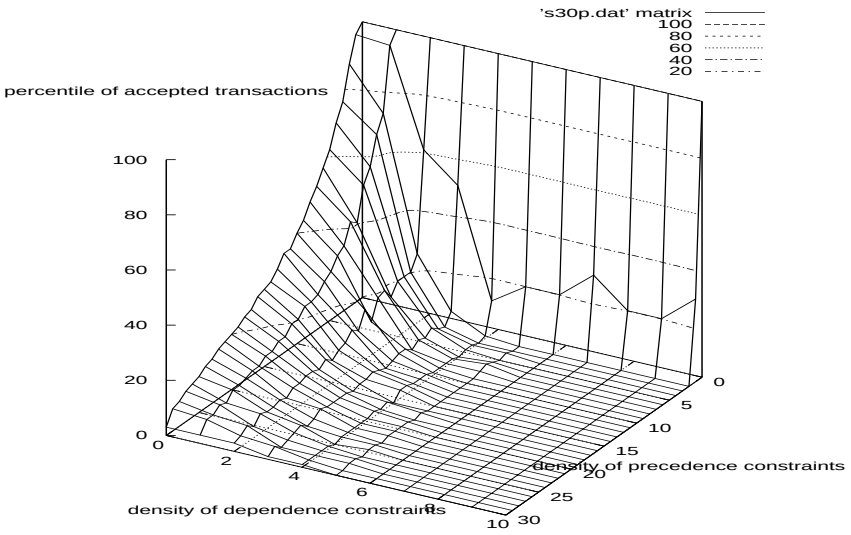

Fig. 4. Percentiles of accepted actions in optimal solutions at order 30.

It is worth noting that these figures computed at order 30 seem quite general for all orders. Figure 3 depicts the mean running time on a logarithmic scale, as a function of the order, around the hardness peak. The figure indicates the exponential behavior of the CLP program as a function of the order, with a constant hardness peak centered around density 7 for precedence constraints and density 0 for dependency constraints.

Figure 4 shows the percentiles of accepted actions in the optimal solutions at order 30. Note that the y axis which indicates the density of precedence constraints, is reversed for a better viewing of the surface. Obviously, the percentile of accepted actions decreases from 100% to 0% as the density of precedence constraints varies from 0 to the order, and similarly for the dependency constraints as soon as the density of precedence constraints is non zero.

3.3 Polynomial complexity of the first solution found

The computational complexity of optimality proofs prevents experimentation on large orders. In this section we measure the running time of the CLP

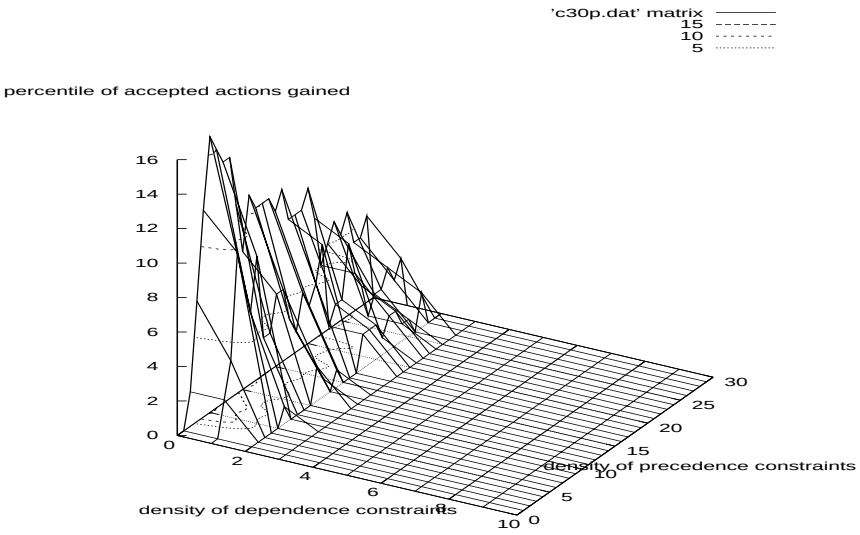

Fig. 5. Percentiles of actions added in the optimal schedule w.r.t. the first schedule found, at order 30, for all densities of precedence constraints ranging from 0 to 30, and densities of dependency constraints ranging from 0 to 10.

program for finding the first solution with the labeling heuristics but without optimization.

Interestingly, the first solution found with the labeling heuristics is always of good quality. Figure 5 depicts the percentiles of actions added in the optimal schedule w.r.t. the first schedule found. Not surprisingly, the highest percentiles are in the zone of the computational complexity peak. At order 30, the maximum error in the first solution found is 16%, and the error stays below 10% for all densities outside a limited peak defined by the intervals $[0, 1]$ for dependencies and $[4, 9]$ for precedences. This good behavior of the program is of course due to the labeling heuristics which uses best-first search.

Figure 6 depicts the running times for finding the first solution at order 50 for all densities. This figure shows the absence of computational complexity peak. Figure 7 depicts the running times for finding the first solution as a function of the order, ranging from 100 to 700, and of the density for precedence constraints, ranging from 0 to 20, in the hardest part i.e. without dependency constraints. The figure indicates a polynomial complexity of the CLP program for computing the first solution.

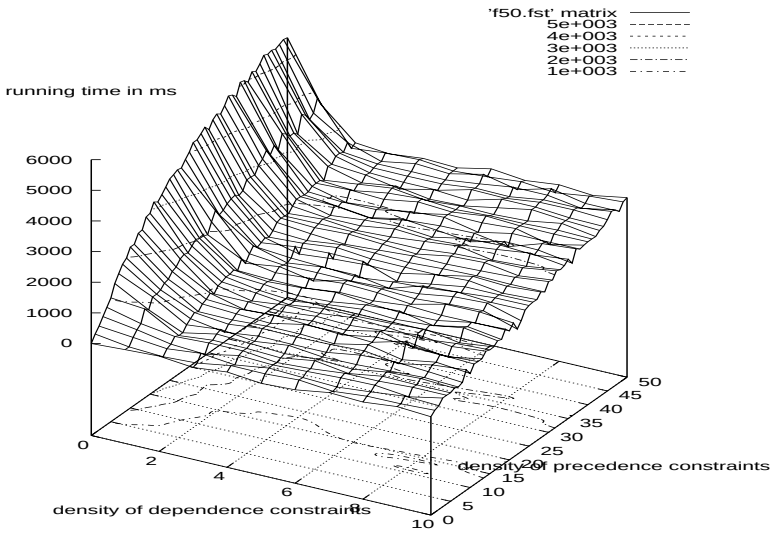

Fig. 6. Cumulated computation time for finding the first solution on ten runs at order 50, as a function of the density of precedence constraints (ranging from 0 to 50), and of the density of dependency constraints (ranging from 0 to 10).

4 Benchmarks

At the present time, we do not have benchmarks of real-life log-based reconciliation problems. Nevertheless we expect that in real-life reconciliation problems, the optimal solutions accept more than 80% of the actions typically. Figure 4 contains an analysis of the percentiles of accepted actions in optimal solutions as a function of the densities, at order 30. These considerations, plus some preliminary inspections at some calendar applications or the jigsaw problem presented in [14], lead us to create a benchmark of randomly generated problems with density 1.5 for both precedence constraints and dependency constraints. We added a second series of more difficult benchmarks generated with the same density 1.5 for precedence constraints but without dependency constraints. These benchmarks are available at URL `http://pauillac.inria.fr/~fages/Reconcile/Benchs`.

Table 1 depicts the experimental results on both series of benchmarks. The size given in the second column is the total number of actions. The numbers of dependency constraints and precedence constraints are given in the following columns. These constraints are generated for each pair of actions randomly, with probability $1.5/size$ which gives $1.5 * size$ constraints of each type in average. The second series of benchmarks contains no dependency constraints.

The running times of this section have been measured in GNU-Prolog on a 866MHz Pentium III PC under Linux. We indicate, in order, the number of

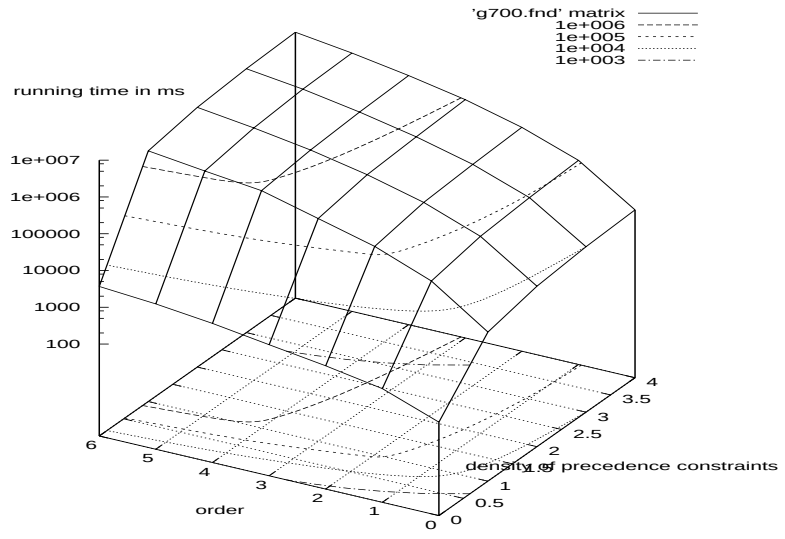

Fig. 7. Cumulated computation time on a logarithmic scale on ten runs, as a function of the order, ranging from 100 to 700 (scaled 0 to 6), and of the density of precedence constraints, ranging from 0 to 20 (scaled 0 to 4). The density of dependency constraints is 0.

Benchmark				CLP				
Bench	Size	#dep	#prec	First sol.	Time (ms)	Opt. sol	Time (ms)	Proof (ms)
r100v1	100	120	151	98	60	98	110	110
r100v2	100	163	141	75	80	77	90	190
r100v3	100	139	145	94	70	95	130	240
r100v4	100	138	152	100	60	100	60	70
r100v5	100	134	162	52	70	52	70	80
r200v1	200	315	284	64	240	65	260	300
r200v2	200	260	301	189	270	191	28330	110600
r500v1	500	740	784	198	1400	198	1400	1540
r500v2	500	761	736	490	1650	$\geq$491	65090	??
r800v1	800	1181	1165	770	4990	??	??	??
r800v2	800	1237	1134	318	3410	318	3410	5580
r1000v1	1000	1493	1562	389	5500	389	5500	6080
r1000v2	1000	1462	1586	938	42880	??	??	??
t40v1	40	0	63	36	10	36	10	340
t40v2	40	0	64	36	10	37	180	440
t40v3	40	0	49	38	10	38	10	20
t40v4	40	0	64	35	10	37	110	240
t50v1	50	0	82	44	20	45	1080	6220
t50v2	50	0	78	46	10	47	240	630
t50v3	50	0	84	43	10	44	970	8120
t50v4	50	0	67	45	10	46	90	200
t70v1	70	0	91	67	10	68	530	1120
t70v2	70	0	104	67	20	67	20	190
t80v1	80	0	120	75	30	??	??	??
t100v1	100	0	146	93	50	??	??	??
t200v1	200	0	323	189	200	??	??	??
t500v1	500	0	740	482	1980	??	??	??
t800v1	800	0	1181	772	7230	??	??	??
t1000v1	1000	0	1459	954	29020	??	??	??

Table 1. Experimental results.

accepted actions in the first schedule found, the running time for finding this solution, the number of accepted actions in the optimal schedule, the running time for finding the optimal schedule (from the start), and finally the total running time including the proof of optimality.

As expected from the previous section (see figure 4), the first solution found in these benchmarks is always very near or equal to the optimal solution. This indicates that the most-constrained variable choice heuristics combined with the best-first search heuristics perform very well in this modeling of the problem. Optimal solutions with their optimality proof are always computed in less than a second for problems with up to a hundred of actions. On larger problems optimality proofs become difficult to obtain but the first solution found is always satisfying and fast to compute. The problems without dependency constraints are harder to solve. Optimality proofs are nevertheless always obtained on problems of size up to 70.

5 Discussion

Reconciliation problems in nomadic applications present interesting combinatorial optimization problems, with both static versions and on-line versions. We have studied an NP-hard log-based reconciliation problem. Its modeling with boolean and precedence constraints lead to a simple CLP(FD,B) program that finds surprisingly good solution up to a thousand of actions, and proves optimality up to a hundred of actions on realistic benchmarks. Note that this program could still benefit from more efficient algorithms for detecting cycles in precedence constraints, with a complexity independent of the size of the variables' domain [7].

Besides the intrinsic interest of the proposed CLP program for solving real-life log-based reconciliation problems, we believe that the experimental investigation of the average-case complexity of this program is of quite general interest for the design of log-based reconciliation algorithms. In particular, our analysis locates where the hard problems are, and clearly shows that it is crucial to use dependency constraints in an active way when searching for a schedule as these constraints greatly reduce the complexity of the problem.

The phase transitions observed in this optimization problem, along the two axes of densities for precedence and dependency constraints are also interesting to study in their own right. Nevertheless it should be clear that providing rigorous proofs for our experimental observations may be a very difficult task.

On-going work concerns the development of other algorithms based on local search [1], which is one fundamental heuristics that has been shown particularly effective for many classes of applications, including some classical benchmarks of constraint programming [15, 6, 11]. One potential advantage of local search methods for reconciliation problems is that they can benefit from the initial logs as starting solution. Another one is that they can provide solutions incrementally for on-line reconciliation problems. Our first experiments with a randomized local search algorithm with Tabu heuristic [13] showed however poor performance

compared to the CLP program. One difficulty lies in the handling of both boolean variables and integer variables, and in the handling of the objective function which differs from a max-CSP problem.

Acknowledgment. I would like to thank Silvano Dal Zilio, Peter Dreuschel, Cédric Fournet, Anne-Marie Kermarrec, Marc Shapiro and Antony Rowstron for fruitful discussions on this application.

References

1. E. Aarts and J. Lenstra (Eds). *Local Search in Combinatorial Optimization*. Wiley, 1997.
2. D. Achlioptas. Setting two variables at a time yields a new lower bound for random 3-sat. In *Proc. of 32th ACM Symp. on Theory of Computing*, pages 28–37, 2000.
3. S. Balasubramanian and B. Pierce. What is a file synchronizer? In *Proc ACM/IEEE Int. Conf on Mobile Computing and Networking*, 1998.
4. P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proc. 12th International Joint Conference on Artificial Intelligence IJCAI'91*, pages 331–337, 1991.
5. C. Coarfa, D.D. Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M.Y. Vardi. Random 3-sat: the plot thickens. In *Proc. of sixth Conference on Principles and Practice of Constraint Programming CP*, volume 1894 of *Lecture Notes in Computer Science*, Singapore, September 2000. Springer-Verlag.
6. P. Codognet. Adaptive search: preliminary results. In *Proc. of fourth ERCIM/CompulogNet Workshop on Constraints*, Venice, Italy, june 2000.
7. R. Dechter, I. Mieri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, january 1991.
8. O. Dubois, Y. Boufkhad, and J. Mandler. Typical random 3-sat formulae and the satisfiability threshold. In *Proc. of 11th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 126–127, 2000.
9. W.K. Edwards, E.D. Mynatt, K. Petersen, M.J. Spreitzer, D.B. Terry, and M.M. Theimer. Designing and implementing asynchronous applications with bayou. In *Proc. Int. Symp. on User Int. Software and Tech.*, Alberta, Canada, october 1997.
10. P. Cederqvist et al. Version management with cvs, 1992.
11. P. Galinier and J.K. Hao. A general approach for constraint solving by local search. *Submitted to Annals of Operations Research*, january 2001.
12. I. Gent and T. Walsh. The SAT phase transition. In *Proc European Conf. on Artificial Intelligence*, pages 105–109, 1998.
13. F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1998.
14. A.M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The IceCube approach to the reconciliation of divergent replicas. In *Proc. of Twentieth ACM Symposium on Principles of Distributed Computing PODC*, Newport, RI USA, August 2001.
15. L. Michel and P. Van Hentenryck. Localizer: A modeling language for local search. *INFORMS Journal of Computing*, 11(1):1–14, 1999.
16. K. Petersen, M.J. Spreitzer, D.B. Terry, M.M. Theimer, and D.J. Demers. Flexible update propagation for weakly consistent replication. In *Proc. ACM SIGOPS Symp. on Operating Systems Principles SOSP-16*, pages 288–301, Saint-Malo, France, 1997.

17. B. Selman and S. Kirkpatrick. Critical behavior in the computational cost of satisfiability testing. *Artificial Intelligence*, 81((1-2)):273–295, 1996.
18. M. Shapiro, A. Rowstron, and A.M. Kermarrec. Application-independent reconciliation for nomadic application. In *Proc. of SIGOPS European workshop Beyond the PC: new challenges for the operating system*, Kolding, Danemark, september 2000.