Comparing ASP, CP, ILP on Wire Routing Problems

Elvin Çoban and Esra Erdem

Faculty of Engineering and Natural Sciences
Sabancı University, Istanbul 34956, Turkey

Abstract. We study three declarative programming paradigms, Answer Set Programming (ASP), Constraint Programming (CP), and Integer Linear Programming (ILP) on a challenging application: wire routing. We represent wire routing in each formalism in a systematic way, compare the formulations both from the point of view of knowledge representation (e.g., how tolerant they are to elaborations) and from the point of view of computational efficiency (in terms of computation time and program size). Also we discuss various reformulations of the problems based on different mathematical models. Some formulations suggest new approaches to solving wire routing.

1 Introduction

Wire routing is the problem of determining the physical locations of all the wires interconnecting the circuit components (e.g., transistors, gates, functional units) on a chip, possibly in the presence of obstacles (e.g., parts of the chip occupied by existing devices such as memory and registers). We consider in particular one sort of wire routing that asks for a configuration of wires connecting a given set of pins such that the wires do not go through obstacles, and that the total wire length be minimum (motivated by delay minimization). The decision version of this problem is NP-hard [1].

We consider two decision problems corresponding to the problem above, and represent each decision problem in three declarative programming paradigms, ASP, CP, and ILP in a systematic way: First we describe the constraints in a metalanguage, as precise and straightforward as possible, and then formulate each constraint in each formalism. We compare these formulations both from the point of view of knowledge representation (e.g., how easy it is to express some concepts and constraints, like reachability and cardinality constraints, and how tolerant the representations are to elaborations) and from the point of view of computational efficiency (i.e., in terms of CPU time and program size). After that we discuss possible ways of improving the computational efficiency (e.g., by introducing auxiliary variables, and adding redundant constraints) taking into account elaboration tolerance, and other reformulations of the problems based on different mathematical models. Also we consider alternative mathematical models of wire routing and examine the corresponding reformulations in other formalisms. In our experiments, we use the ASP solver CMODELS;[1] and the commercial CP and ILP solver ILOG OPL.[2]

[1] http://www.cs.utexas.edu/users/tag/cmodels.html
[2] http://www.ilog.com/products/solver

One concept that we use when comparing formalizations is elaboration tolerance, introduced by John McCarthy in [2]: A formalism is elaboration tolerant if it can be easily modified to adapt to new phenomena. The simplest modification to a formulation is adding new formulas and predicates, whereas adding arguments to formulas and predicates is considered to change a formulation the most.

A similar study that compares ASP and CP/ILP is [3]: the authors compare the computation times of some solvers on some problems, relative to various reformulations (e.g., by adding auxiliary variables). Our work can be seen as a continuation and/or a complement of this study. On the other hand, our paper not only reports a comparison of ASP, CP, ILP based on our experiences, but also introduces a new approach and formulations to solving wire routing.

We refer the readers to relevant books for more information about ASP, ILP, and CP [5–7]. All formulations of the problems mentioned in the paper, as well as results of our experiments, are available at the web page [4].

2 Problem description

We study wire routing by means of a graph problem, Rectilinear Steiner Tree (RST) construction, defined as follows. Consider an undirected graph (V, E) with a positive number, called the *length*, assigned to every edge. A *Steiner tree* for a set S of vertices is a tree (V', E') that is a subgraph of (V, E) where $S \subseteq V'$ and the total length of edges in E' is minimum. In the **RST**) problem the goal is to find a Steiner tree in the case when the edges of the given graph (V, E) are horizontal and vertical line segments in a plane. We assume that the graph is specified as a subset of the grid of unit squares; the goal is to find a total number of unit segments covered by the tree is minimum. We consider a decision problem that corresponds to it:

RST-DEC Given a subgraph G of a rectangular grid, a source point s, a set S of sink points, and an integer k, decide that there is a subgraph of G such that

- R1 every point in the subgraph is included in a path connecting a sink to the source,
- R2 each sink i and the source are included in the same path (Path i) in the subgraph,
- R3 in Path i, each end point is not connected to more than one point,
- R4 in Path i, each internal point is connected to exactly two points, and
- R5 the total length of the subgraph is less than or equal to k.

Conditions R1–R4 ensure that the subgraph connects every sink point to the source. For a sufficiently small lower bound k, a solution to **RST-DEC** is a solution to **RST**.

Another mathematical model of **RST-DEC**, introduced in particular for an ILP formulation, is flow-based. The idea is to introduce a dummy node, say D, and assign some flow (of value 0 or 1) to every edge on a directed Path i that connects D to an end point i; the edges that have positive flow are included in the subgraph.

3 Formulations of wire routing

The former mathematical model for **RST-DEC** described in the previous section has already been formulated in ASP [11]. The latter model has been formulated in ILP [12]. We have represented the former model in ILP and CP; and the latter model in ASP and CP. All formulations are available at [4].

We want to emphasize that our goal in this work is not only to compare efficiency of existing ASP/ILP/CP solvers but also to compare these three paradigms from the point of view of knowledge representation. This is why we formulated each mathematical model in all three formalisms, in a systematic way, as described in the introduction.

4 A comparison of the formulations

We compare the formulations of **RST-DEC** both from the point of view of representation (in terms of expressivity, and elaboration tolerance), and from the point of view of computational efficiency (in terms of CPU time, and program size).

Elaboration tolerance Consider a variation of wire routing that requires some restrictions on lengths of the wires connecting the sink pins to the source pin, that is to say, on signal delays. We can express in ASP that any wire cannot be longer than a specific value, say `l`, by adding to the problem description the constraint

```
:- l+2 {at(N,X,Y):point(X,Y)}, path(N).
```

(no path connecting the source pin to a sink pin is allowed to cover `l+1` unit segments on the grid). This elaboration can be easily tolerated by CP/ILP representation as well, by adding to the problem description the constraint

```
forall(<k1,k2> in sink) sum(<X,Y> in V) at[<k1,k2>,<X,Y>] <= l+1;
```

The formulations of **RST-DEC** with respect to R1-R5 do not include an explicit definition of a path; connectedness of a tree is guaranteed when we find the shortest path. However, if the upper bound k on the total number of the segments covered by the paths is not small enough then the graph generated by any of these programs may not be a tree, i.e., there may be cycles in it. So let us consider a variation of **RST-DEC** where the generated subgraph is a tree. Such a modification can be tolerated by ASP by adding to the program the constraint

```
:- at(N,X,Y), at(N1,X,Y), not reachable(N,N1,X,Y),
   point(X,Y), path(N;N1), N<=N1.
```

where `reachable(N,N1,X,Y)` expresses that point `(X,Y)` is reachable from the source point via the common segments of the paths `N` and `N1` (`N`≤`N1`). This predicate has a straightforward recursive definition. The base case is defined by the rule

```
reachable(N,N1,X,Y) :- source(X,Y), path(N;N1), N <= N1.
```

The inductive step is defined by the rule

```
reachable(N,N1,X+1,Y) :- reachable(N,N1,X,Y), path(N;N1), N <= N1,
  in(h,N,X,Y), in(h,N1,X,Y), point(X,Y), point(X+1,Y).
```

and a similar rule for vertical segments. With this formulation, one can generate a rectilinear tree of length at most k, using an answer set solver. On the other hand, this elaboration cannot be tolerated in the CP and ILP formulations easily (without enumerating all possible paths, and/or introducing too many auxiliary variables). In this sense, ASP is more elaboration tolerant.

Computational efficiency With each formulation of **RST-DEC**, we tried to solve the four problem instances, of [11], using CMODELS 3.74 with LPARSE 1.0.17 and MINISAT 2.0, and ILOG OPL 5.5 on a machine with Xeon 1.5GHz CPU with 4x512MB

Table 1. Experimental results for **RST-DEC**: computation times

Problem	# of sink pins	k	ILP – OPL		ASP – CMODELS	
			k	$k-1$	k	$k-1$
A	15	58	4.64	-	1.38	10.38
B	20	68	10.35	-	2.30	18.54
C	25	74	149.94	-	2.75	56.24
D	30	76	265.73	-	2.76	10.91

Table 2. Experimental results for **RST-DEC**: program size

Problem	# of sinks	k	ILP – OPL		ASP – CMODELS	
			# of variables	# of constraints	# of atoms	# of clauses
A	15	58	11776	72243	36643	110381
B	20	68	15616	115429	41989	126943
C	25	74	19456	168277	46200	140060
D	30	76	23296	230699	50248	152784

RAM running RedHat Linux 4.3. For each problem instance of **RST**, we considered two instances of **RST-DEC**: one with the optimal value of k (to generate a solution), and the other with 1 less than the optimal (to verify the minimality of the solution). As in [11], when generating a solution we added to the programs some rules/constraints describing the heuristics above; verification of minimality does not involve those heuristics.

Table 1 compares the computation time (CPU time in sec.s), and Table 2 compares the program size (number of variables and constraints in the case of OPL, and number of atoms and clauses in the case of CMODELS) of formulations of **RST-DEC** described above. In Table 1, a dash - indicates that the problem could not be solved in 900 sec.s. For instance, consider Problem A. An RST is computed using OPL, when $k = 58$, in 4.64 sec.s with the ILP formulation of **RST-DEC** and the heuristics. With the ILP formulation of **RST-DEC** only (without any heuristics), OPL could not verify in less than 900 sec.s that there is no tree of size $k = 57$ connecting the given sink pins to the source. In the former computation, the program contains 11776 variables and 72213 constraints. With the CP formulation, OPL could not generate a solution for any of the problems in 900 sec.s.; so results for CP are not shown in the tables.

The flow-based ILP formulation significantly improves the computational efficiency: for Problem C with $k = 74$, the CPU time decreases to 3.17 sec.s, and the number of constraints to 10241. The ASP formulation does not improve the computational efficiency: for Problem C, the CPU time increases to 62.70 sec.s, and the number of clauses to 560520. No problems can be solved in 900 sec.s with the CP formulation.

5 Conclusion

We studied wire routing in the context of three declarative programming paradigms, ASP, CP, ILP, usually used by different research communities in AI/CP/OR to solve such hard problems. We observed some similarities and differences of these three formalisms and the state-of-the-art solvers developed for reasoning in these formalisms, not only from the point of view of computational efficiency but also from the point of view of knowledge representation.

Due to the declarative nature of the formalisms, and the systematic representation of the problems in each formalism, we observed some refreshing similarities. For instance, in all formalisms one can express constraints on the cardinality of a set, as seen in the formulation of **RST-DEC** with length restrictions: in CP and ILP one can use summation, and in ASP one can use a cardinality expression. On the other hand, we observed some differences due to the expressivity of the formalisms. In the ASP representations, one can include recursive definitions, like the definition of reachability in the formulation of **RST-DEC**; in the CP/ILP formulations, we have to enumerate all possibilities (cf. formulations of Steiner tree problem in [12]), or introduce too many auxiliary variables. Due to the expressivity of its representation language, ASP is more tolerant to some elaborations (e.g., computation of a tree). Besides comparing ASP, CP, ILP on wire routing, we introduced new formulations (e.g., CP/ILP formulations of **RST-DEC**, and ASP formulations of the flow-based approach). CMODELS performs better with the straightforward representation of **RST-DEC**; whereas OPL performs better with the flow-based ILP formulation.

This work is about comparison of the paradigms ILP, CP, and ASP with respect to knowledge representation and with respect to computational efficiency of their solvers. Therefore, we have not considered in our comparisons the existing algorithmic approaches, relaxations of representations, embedded uses of these paradigms, etc. (e.g., [13]), where the concern is more about efficient computation of (sometimes not exact) solutions rather than representational issues like elaboration tolerance.

References

1. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP complete. SIAM Journal of Applied Mathematics **32** (1977) 826–834
2. McCarthy, J.: Elaboration tolerance. In: Proc. of CommonSense. (1998)
3. Cadoli, M., Mancini, T., Micaletto, D., Patrizi, F.: Evaluating ASP and commercial solvers on the CSPLIB. In: Proc. of ECAI. (2006) 68–72
4. `http://people.sabanciuniv.edu/~esraerdem/benchmarks/rst.html`
5. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
6. Hoffman, K.L., Padberg, M.: Encyclopedia of Operations Research and Management Science. Kluwer, Massachusetts (2001)
7. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming. Elsevier (2006)
8. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Logic Programming: Proc. of the 5th International Conference and Symposium. (1988) 1070–1080
9. Ferraris, P., Lifschitz, V.: Mathematical foundations of answer set programming. In: We Will Show Them! Essays in Honour of Dov Gabbay. Volume 1. (2005) 615–664
10. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234
11. Erdem, E., Wong, M.D.F.: Rectilinear Steiner Tree construction using answer set programming. In: Proc. of ICLP. (2004) 386–399
12. Maculan, N., Plateau, G., Lisser, A.: Integer linear models with a polynomial number of variables and constraints for some classical combinatorial problems. Pesquisa Operacional **23**(1) (2003) 161–168
13. Polzin, T.: Algorithms for the Steiner Problem in Networks. PhD Thesis (2003)