Experimental Evaluation of Subgraph Isomorphism Solvers

Christine Solnon

Experimental Evaluation of Subgraph Isomorphism Solvers

Christine Solnon*

INSA-Lyon, LIRIS, UMR5205, F-69621, France

Abstract. Subgraph Isomorphism (SI) is an NP-complete problem which is at the heart of many structural pattern recognition tasks as it involves finding a copy of a pattern graph into a target graph. In the pattern recognition community, the most well-known SI solvers are VF2, VF3, and RI. SI is also widely studied in the constraint programming community, and many constraint-based SI solvers have been proposed since Ullman, such as LAD and Glasgow, for example. All these SI solvers can solve very quickly some large SI instances, that involve graphs with thousands of nodes. However, McCreesh *et al.* have recently shown how to randomly generate SI instances the hardness of which can be controlled and predicted, and they have built small instances which are computationally challenging for all solvers. They have also shown that some small instances, which are predicted to be easy and are easily solved by constraint-based solvers, appear to be challenging for VF2 and VF3. In this paper, we widen this study by considering a large test suite coming from eight benchmarks. We show that, as expected for an NP-complete problem, the solving time of an instance does not depend on its size, and that some small instances coming from real applications are not solved by any of the considered solvers. We also show that, if RI and VF3 can solve very quickly a large number of easy instances, for which Glasgow or LAD need more time, they fail at solving some other instances that are quickly solved by Glasgow or LAD, and they are clearly outperformed by Glasgow on hard instances. Finally, we show that we can easily combine solvers to take benefit of their complementarity.

1 Introduction

Subgraph Isomorphism (SI) is an NP-complete problem which involves finding a copy of a pattern graph into a target graph, *i.e.*, finding a mapping that associates a different target node to each pattern node in such a way that edges are preserved. There are two main variants of SI: in the non-induced case, only pattern edges must be preserved (*i.e.*, pattern nodes connected by an edge must be mapped to target nodes connected by an edge); in the induced case, target edges must also be preserved (*i.e.*, target nodes connected by an edge cannot be mapped to pattern nodes not connected by an edge).

SI is at the heart of many structural pattern recognition tasks in different application fields such as image or biology, for example [7]. In the pattern recognition community,

* This work has been done in collaboration with Ciaran McCreesh, Patrick Prosser, and James Trimble. In particular, all experiments have been run by Ciaran McCreesh and used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1).

the most well-known algorithms used to solve SI are VF2 [8], VF3 [5], and RI [4]. These solvers will be referred to as *PR solvers*. PR solvers perform a depth-first search in a space of states: each state corresponds to a partial mapping where some pattern nodes have been mapped, and each state is recursively extended by adding to its partial mapping a new couple of mapped pattern/target nodes.

SI is also widely studied in the constraint programming community as it may be modelled as a constraint satisfaction problem in a straightforward way. Many constraint-based solvers have been proposed for solving SI since Ullman [20] such as, for example, nRF+ [15], ILF [21], LAD [18], SND [2], and Glasgow [1, 16]. These solvers will be referred to as *CP solvers*. Like VF2, VF3, and RI, CP solvers recursively extend partial mappings. However, a fundamental difference is that CP solvers maintain, for each non-mapped pattern node, the list of candidate target nodes that may be mapped to it, and they propagate constraints to reduce these lists. This constraint propagation mechanism is expensive, both in memory and time, but it reduces the number of states to explore.

Recent PR and CP solvers can solve very quickly rather large SI instances, that involve graphs with thousands of nodes. Indeed, being NP-complete does not mean that all instances are hard to solve, and some instances of NP-complete problems can be very easy to solve. In particular, in [6], Cheeseman *et al.* show that NP-complete problems can be summarised by at least one "order parameter", and that hard instances occur at a critical value of such a parameter. In [17], McCreesh *et al.* use this approach to generate "really hard" random SI instances according to three random graph models. For example, for *Erdős-Rényi* random graphs (where edges are generated according to an independent probability [11]), instances of non-induced SI may be generated by fixing pattern and target numbers of nodes, and varying pattern and target edge probabilities from 0 to 1. In this case, a phase transition occurs between entirely satisfiable instances (when patterns are sparse and targets are dense) and entirely unsatisfiable instances (when patterns are dense and targets are sparse), and the location of this phase transition can be predicted by computing the expected number of solutions. Instances located within this phase transition are computationally challenging for all solvers even when graphs are small (*e.g.*, thirty pattern nodes and 150 target nodes). However, the experimental study reported in [17] also shows that some small instances which are predicted as easy, and which are easily solved by CP solvers, appear to be challenging for PR solvers.

In this paper, we widen this experimental study and we experimentally evaluate and compare RI, VF2, VF3, Glasgow, and LAD on a large test suite of $14,621$ instances coming from eight benchmarks. In Section 2, we describe our test suite. In Section 3, we show that, as expected for an NP-complete problem, the solving time of an instance does not depend on its size, and that some small instances (including instances coming from real applications) are not solved by any of the considered solvers. In Section 4, we identify easy and hard instances and we show that, if PR solvers are able to solve very quickly easy instances (for which CP solvers often need more time), they fail at solving some other instances that are rather quickly solved by CP solvers, and they are clearly outperformed by Glasgow on hard instances. Finally, in Section 5, we show that we can easily combine PR and CP solvers to take benefit of their complementarity.

Class	#inst	Pattern graphs						Target graphs					
		#nodes		#edges		density		#nodes		#edges		density	
		min	max	min	max	min	max	min	max	min	max	min	max
images	6,302	4	170	4	241	.02	.67	1,072	5,972	1,539	8,888	.00	.00
meshes	3,018	40	199	114	539	.02	.15	201	5,873	252	15,292	.00	.02
LV	3,831	10	128	10	4,950	.02	1.00	10	6,671	10	209,000	.00	1.00
randERP	200	30	30	128	387	.29	.89	150	150	4,132	8,740	.37	.78
randER	270	40	360	41	12,410	.02	.21	200	600	436	34,210	.02	.19
randBVG	540	40	480	43	2,137	.01	.20	200	800	299	3,600	.00	.05
randM	360	51	777	76	2,075	.01	.08	256	1,296	672	4,377	.00	.03
randSF	100	180	900	478	5,978	.01	.17	200	1000	592	7,148	.01	.16

Table 1. For each class, we give the number of instances (#inst) and then describe pattern and target graph features: minimum and maximum number of nodes, number of edges, and density.

2 Experimental set-up

Test suite. We consider $14,621$ instances coming from eight benchmarks described in Table 1, and available at `liris.cnrs.fr/christine.solnon/SIP.html`. *images* and *meshes* are coming from real applications where both pattern and target graphs correspond to graphs extracted from segmented images and 3D meshes [9, 19].

LV is a benchmark described in [15]. It uses 113 graphs with various properties coming from the Stanford GraphBase described by Knuth in [13]. The benchmark is built by splitting the set of graphs in two parts: the first part contains the 50 smallest graphs; the second part contains the 63 remaining graphs. We consider all pairs of graphs such that the pattern graph belongs to the first part, the target graph belongs to the first or the second part, and the target graph has at least as many nodes as the pattern graph.

*rand** (with $* \in \{ERP, ER, BVG, M, SF\}$) are randomly generated instances. *randERP* are instances close to the phase transition (expected to be hard as explained in [17]), and all graphs are *Erdős-Rényi* graphs. *randER*, *randBVG*, and *randM* are coming from the database described in [10], and graphs are *Erdős-Rényi* graphs, (modified) bounded valence graphs and 4D meshes, respectively. *randSF* is described in [22] and it contains scale-free graphs. All instances in *randER*, *randBVG*, *randM*, and *randSF* (except 20 instances in *randSF*) are feasible by construction because the pattern has been extracted from the target.

All graphs have at least as many edges as nodes. Hence, the size of a graph is dominated by its number of edges.

Performance measures. The experiments were performed on the EPCC Cirrus HPC facility, on systems with dual Intel Xeon E5-2695 v4 CPUs and 256GBytes RAM, running Centos 7.3.1611, and GCC 7.2.0 as the compiler. Each run has been limited to $1,000$ seconds of CPU time. Some instances are not solved within this time limit (note that even when increasing the time limit to $100,000$ seconds some instances are still unsolved). We consider two different performance measures: when all solvers have been able to solve all instances of a benchmark, we report the average solving time; when some instances have not been solved within the time limit, we report the number of solved

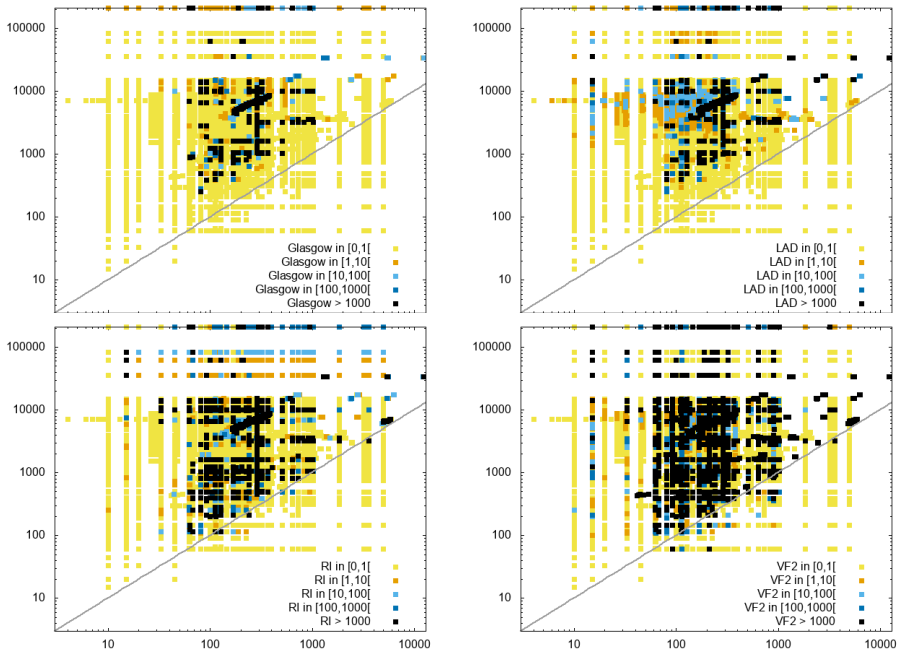

Fig. 1. Number of pattern edges (x-axis), target edges (y-axis), and solving time (colour) for non-induced SI: top left = Glasgow; top right = LAD; bottom left = RI; bottom right = VF2.

instances within the time limit, and we plot the evolution of the cumulative number of solved instances with respect to time (*i.e.*, the function $f(t) = \#\{i \in I : t_i^s \leq t\}$ where I is the set of instances, s a solver, and t_i^s the time spent by s to solve an instance $i \in I$).

We do not consider memory consumption as a performance measure as solvers never run out of memory, even for the largest instances (all solvers have polynomial memory complexities). However, CP solvers need more memory than PR solvers as they maintain candidate lists of target nodes for each non-mapped pattern vertex.

Different variants of Glasgow are described in [1]. We consider the *biased* variant, which is the default setting[1].

3 Does the solving time depend on graph sizes?

To study the relation between the solving time and the size of an instance, we plot in Fig. 1 and 2 the time spent by each solver on each instance. Each instance corresponds to a point (x, y) where x is the number of pattern edges, y the number of target edges, and the colour depends on the solving time: yellow if it is smaller than one second, and black if the instance has not been solved within 1000 seconds (if several instances have the same size, the colour corresponds to the average solving time for all these instances).

[1] Glasgow is available at https://github.com/ciaranm/glasgow-subgraph-solver

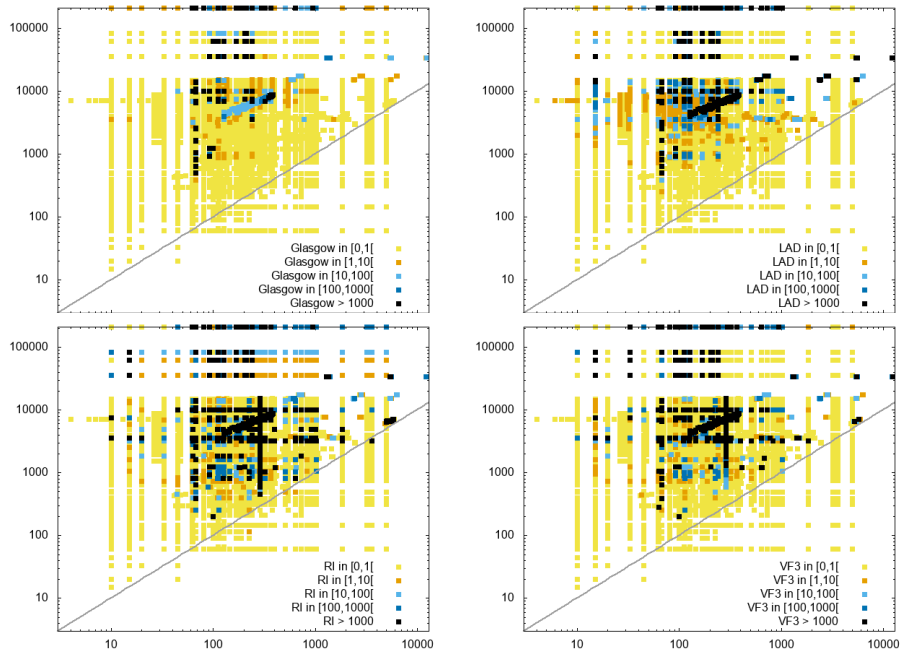

Fig. 2. Number of pattern edges (x-axis), target edges (y-axis), and solving time (colour) for induced SI: top left = Glasgow; top right = LAD; bottom left = RI; bottom right = VF3.

As expected for an NP-complete problem, these figures show us that *hardness does not depend on size*. Let us first consider the non-induced case, displayed in Fig. 1. Unsolved instances (black points) are not specially concentrated in the top right area of the plots (corresponding to the largest instances). The number of unsolved instances is quite different from a solver to another, but some black points are common to all solvers. Among the set of instances which are solved by none of the solvers, the smallest pattern (resp. target) graph has 62 edges and 30 nodes (resp. 400 edges and 86 nodes). Many much larger instances are solved in less than one second. The gray line separates instances that have more target edges than pattern edges (top left) from those that have less target edges than pattern edges (bottom right). All instances in the bottom right part are trivially infeasible. However, both VF2 and RI are not able to solve some of them.

When looking at the induced case in Fig 2, we also note that the unsolved instances are not necessarily those with the largest graphs and the number of unsolved instances is quite different from a solver to another. Among the set of instances which are solved by none of the solvers, the smallest pattern (resp. target) graph has 62 edges and 30 nodes (resp. 638 edges and 120 nodes). VF3 has much better results on induced SI than VF2 on non-induced SI, and it is always able to quickly solve instances that are trivially infeasible because they have less target edges than pattern edges.

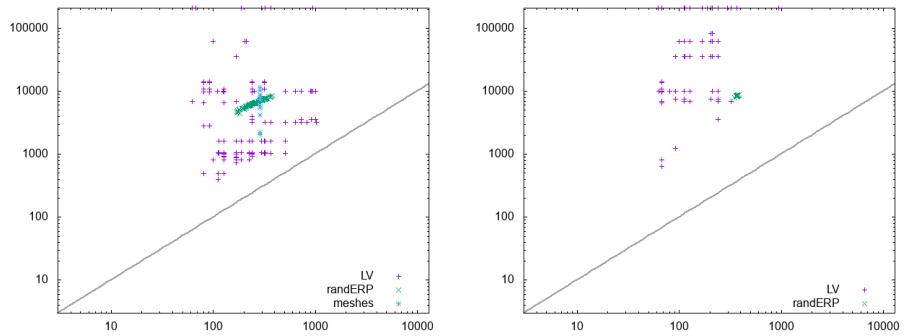

Fig. 3. Number of pattern edges (x-axis), target edges (y-axis), and classes (colour) of unsolved instances: left = non-induced SI; right = induced SI.

	Non-induced SI						Induced SI					
	feasibility		hardness				feasibility		hardness			
Class	yes	no	E	EH	H	U	yes	no	E	EH	H	U
images	52	6,250	2,555	3,747	0	0	50	6,252	2,764	3,538	0	0
meshes	88	2,930	2,361	553	93	11	0	3,018	2,492	521	1	0
LV	596	3,235	2,097	1,477	137	120	191	3,640	2,939	693	139	60
randERP	164	36	0	48	69	83	0	200	0	0	180	20
randER	270	0	0	203	67	0	270	0	72	141	57	0
randBVG	540	0	461	79	0	0	540	0	454	86	0	0
randM	360	0	309	51	0	0	360	0	313	45	2	0
randSF	80	20	4	96	0	0	80	20	75	25	0	0
All	2,150	12,471	7,787	6,254	366	214	1,491	13,130	9,109	5,053	379	80

Table 2. Number of feasible (yes), infeasible (no), easy (E), easy-or-hard (EH), hard (H), and unsolved (U) instances per class.

4 Where are the hard instances?

To have a better insight into where the hard instances are, we have partitioned each class of our benchmark into 4 separate groups, depending on instance hardness. As all instances but those of *randERP* have not been randomly generated with a model that allows us to predict hardness with respect to the phase transition location, we consider an empirical definition of instance hardness:

- an instance is *easy* if the four solvers are able to solve it within one second;
- an instance is *hard* if no solver can solve it within one second, but at least one solver can solve it within the time limit of 1000 seconds;
- an instance is *easy-or-hard* if at least one solver solves it within one second whereas at least one solver cannot solve it within one second;
- an instance is *unsolved* if no solver can solve it within the time limit of 1000 seconds.

In Fig. 3, we display the number of edges in pattern and target graphs of unsolved instances, and in Table 2, we give the number of instances in each group of each class. As expected, many *randERP* instances are unsolved or hard, and none of them is easy:

	easy instances				easy-or-hard instances								hard instances			
	G	L	V	R	G		L		V		R		G	L	V	R
Class	time	time	time	time	#u	time	#u	time	#u	time	#u	time	#u	#u	#u	#u
Non-induced SI																
images	.106	.374	.201	.002	0	(.18)	0	(4.07)	13	-	0	(0.01)	-	-	-	-
meshes	.153	.026	.044	.016	1	-	3	-	276	-	180	-	20	23	91	91
LV	.036	.017	.069	.008	12	-	9	-	886	-	206	-	18	32	130	76
randERP	-	-	-	-	0	(.21)	25	-	48	-	30	-	0	65	69	64
randER	-	-	-	-	0	(.88)	17	-	201	-	2	-	0	57	67	14
randBVG	.017	.119	.004	.003	0	(.07)	0	(2.37)	0	(.01)	2	-	-	-	-	-
randM	.051	.095	.013	.003	0	(.18)	0	(3.94)	18	-	2	-	-	-	-	-
randSF	.007	.004	.497	.001	0	(.11)	0	(.10)	80	-	15	-	-	-	-	-
All	.094	.146	.099	.008	13	-	54	-	1,522	-	437	-	38	177	357	245
Induced SI																
images	.136	.388	.002	.002	0	(.24)	0	(3.58)	0	(.00)	0	(.01)	-	-	-	-
meshes	.173	.024	.001	.009	0	(1.04)	0	(.16)	103	-	237	-	0	0	1	1
LV	.047	.026	.011	.024	4	-	8	-	52	-	96	-	10	53	51	65
randERP	-	-	-	-	-	-	-	-	-	-	-	-	0	175	175	175
randER	.021	.260	.061	.030	0	(.43)	12	-	2	-	1	-	0	47	5	7
randBVG	.018	.125	.002	.004	0	(.07)	0	(2.22)	3	-	4	-	-	-	-	-
randM	.052	.117	.003	.003	0	(.17)	0	(4.31)	6	-	0	(6.11)	0	0	1	0
randSF	.109	.065	.027	.023	0	(.14)	0	(.21)	2	-	10	-	-	-	-	-
All	.108	.146	.005	.012	4	-	20	-	168	-	348	-	10	275	233	248

Table 3. Results of Glasgow (G), LAD (L), VF2/VF3 (V), and RI (R) on non-induced (top) and induced (bottom) SI instances. #u is the number of unsolved instances within 1000s (for easy instances, #u = 0). When all instances are solved, we report the average solving time in seconds.

these instances are close to the phase transition and they are expected to be challenging despite their small size. However, not all unsolved instances are coming from *randERP*. This shows us that really hard instances may occur even if they have not been generated on purpose. For the non-induced case, *LV* and *meshes* respectively contain 120 and 11 unsolved instances, whereas for the induced case, *LV* contains 60 unsolved instances. In both cases, these instances are not the largest ones, and some of them are really small as illustrated in Fig. 3.

Many instances are easy (7,787 instances for the non-induced case, and 9,109 for the induced case), and these easy instances are coming from all classes but *randERP* and *randER* for the non-induced case, and all classes but *randERP* for the induced case.

In Table 2, we also give the number of feasible instance per class. Note that any instance feasible for the induced case is also feasible for the non-induced case. Three classes (*i.e.*, *randER*, *randBVG*, and *randM*) only contain feasible instances as they have been randomly generated in such a way that there always exists at least one solution. There is no obvious relation between feasibility and hardness: hard and unsolved groups contain both feasible and infeasible instances.

5 Experimental comparison of the solvers

In Table 3, we display the results of the four solvers on the different classes, grouped with respect to hardness. For easy instances (which are solved by all solvers), RI is an

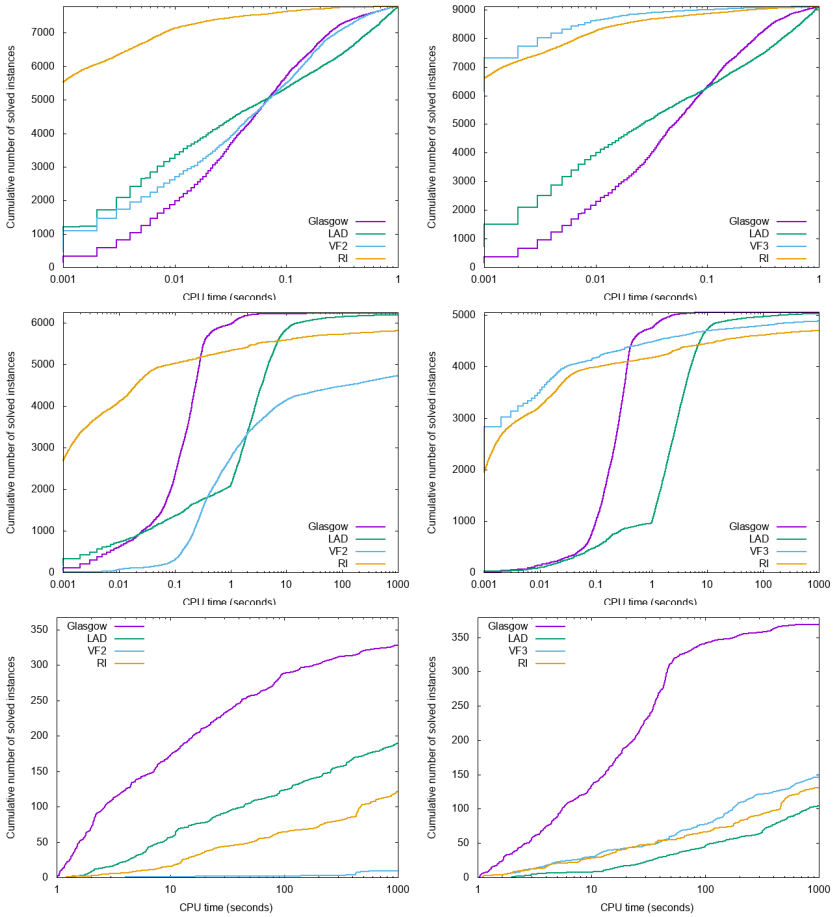

Fig. 4. Cumulative number of solved instances: top = easy instances; middle = easy-or-hard instances; bottom = hard instances; left = non-induced SI; right = induced SI.

order faster than the other solvers for the non-induced case, and VF3 is twice as fast as RI which is an order faster than Glasgow and LAD for the induced case. Hence, on easy instances, the fastest solvers clearly are RI for the non-induced case and VF3 for the induced case, and CP solvers are an order slower.

However, on easy-or-hard and hard instances, PR solvers solve less instances than LAD, and LAD solves less instances than Glasgow. More precisely, for the non-induced case, Glasgow (resp. LAD, VF2, and RI) fails at solving 51 (resp. 221, 1879, and 682) instances. For the induced case, Glasgow (resp. LAD, VF3, and RI) fails at solving 14 (resp. 295, 416, and 596) instances. Hence, on easy-or-hard and on hard instances, the best solver clearly is Glasgow for both the non-induced and the induced case. Actually most easy-or-hard instances are trivially solved by Glasgow in less than one second whereas PR solvers fail at solving many of these instances.

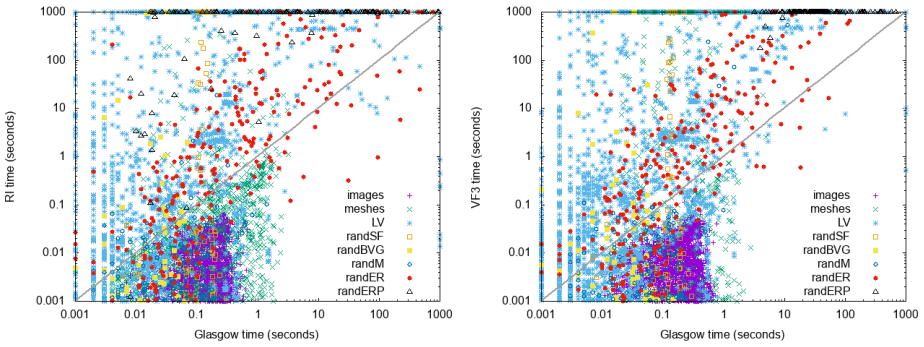

Fig. 5. Comparison of the best PR and CP solvers. On the left (resp. right), each point (x, y) corresponds to an instance which is solved in x seconds by Glasgow and y seconds by RI for the non-induced case (resp. VF3 for the induced case). When an instance is not solved by Glasgow (resp. RI or VF3), it is displayed on $x = 1,000$ (resp. $y = 1,000$).

For the non-induced case, if LAD is outperformed by Glasgow, it is able to solve much more instances than PR solvers. For the induced case, LAD is also outperformed by Glasgow and, if it is able to solve more instances than PR solvers on many classes, it is clearly outperformed by them on *randER* instances. Actually, LAD is the only solver which solves less instances for the induced case than for the non-induced case. This comes from the fact that LAD has been designed for the non-induced case. It has been extended to handle the induced case in a very naive way (by checking that target edges are preserved *a posteriori*), without exploiting properties specific to the induced case.

in Fig. 4, we plot the evolution of the cumulative number of solved instances with respect to time. For easy instances, RI (resp. VF3) dominates all other solvers for the non-induced (resp. induced) case, and it is able to solve more than $5,000$ (resp. $7,000$) instances in less than 0.001s. On these instances, CP solvers often need more time.

For easy-or-hard instances, RI (for the non-induced case) and VF3 (for the induced case) are able to solve more than $2,500$ instances in less than $.001$s. However, they fail at solving hundreds of instances which are easily solved by Glasgow, in less than one second, and the cumulative number of instances solved by Glasgow becomes larger than those of RI and VF3 after 0.3s.

For hard instances, Glasgow clearly outperforms all other solvers and it is able to solve much more instances.

In Fig. 5, we compare the best CP solver (*i.e.*, Glasgow) with the best PR solver (*i.e.*, RI for the non-induced case, and VF3 for the induced case) on a per instance basis. Every point below the gray line corresponds to an instance which is solved quicker by the PR solver than by Glasgow, and the wide majority of these points are on the left of the vertical line $x = 1$, corresponding to instances which are solved in less than one second by Glasgow. Every point above the gray line corresponds to an instance which is solved quicker by Glasgow than by the PR solver, and many of these points are on the

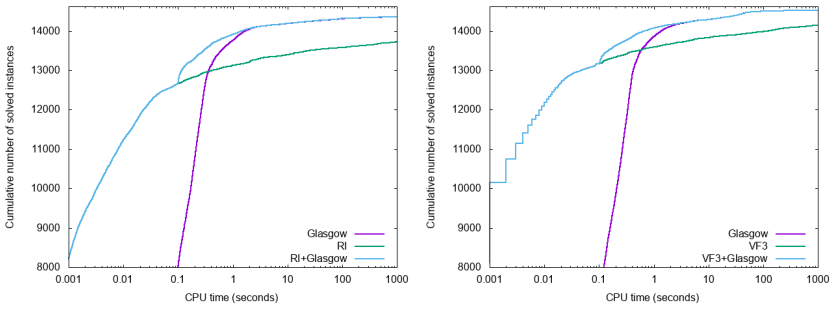

Fig. 6. Cumulative number of solved instances on the whole benchmark of RI (resp. VF3), Glasgow, and RI+Glasgow (resp. VF3+Glasgow) for non-induced SI (left) (resp. induced SI (right)).

horizontal line $y = 1,000$, corresponding to instances which are not solved by the PR solver within the time limit of $1,000$ seconds.

6 Combining solvers to take the best of them

Glasgow is complementary to the best PR solver (*i.e.*, RI for the non-induced case and VF3 for the induced case) as it needs more time on very easy instances, but it is able to solve more instances. We can take benefit of this complementarity as follows: we run the best PR solver with a time limit of t_1 seconds; if the instance has not been solved within this limit, we run Glasgow. The time limit t_1 should be long enough to allow the PR solver to solve easy instances, but not too long in order not to penalise the total solving time when the PR solver is not able to solve the instance. In Fig. 6, we display cumulative numbers of solved instances of the best PR solver, Glasgow, and the combined approach (denoted RI+Glasgow for the non-induced case, and VF3+Glasgow for the induced case) when the time limit t_1 is set to 0.1s. It shows us that this simple combination allows to take the best of both solvers: before 0.1s, the cumulative number of solved instances of RI+Glasgow (or VF3+Glasgow) is equal to the one of RI (or VF3), which is much greater than the one of Glasgow (not displayed because the y-axis starts at $8,000$ and Glasgow solves less than $8,000$ instances in 0.1s); after 0.1s, the cumulative number of solved instances of RI+Glasgow (or VF3+Glasgow) grows faster than the one of RI (or VF3) because Glasgow is able to solve instances which are not solved by RI (or VF3); finally, after a few seconds, the cumulative number of solved instances of RI+Glasgow (or VF3+Glasgow) is very close to the one of Glasgow as the delay of 0.1s due to the run of RI (or VF3) is negligible.

Of course, this very simple approach could be enhanced by considering more solvers (including more variants of each solver, using different ordering heuristics, for example). In this case, we may gather all solvers in a portfolio, and use an algorithm selection approach to dynamically select from the portfolio the solver which is expected to perform best for each new SI instance to solve, as proposed by Kotthoff *et al.* in [14].

7 Conclusion

This study has shown that there are many very easy SI instances which are solved in a few milliseconds by modern solvers, and that some of these instances may involve very large graphs with thousands of nodes. However, there are still small instances which cannot be solved within a reasonable amount of time by any of these solvers. It is important to evaluate solvers on these hard instances too as they do appear in real applications, though they are less frequent than easy instances.

A promising research direction for solving hard instances is to exploit multiple cores, and parallel SI solvers have been introduced in [1, 3, 16], for example. A special attention should be paid on performance measures used to evaluate these approaches. Indeed, measuring an average speed-up between a sequential and a parallel solver is not very meaningful when considering NP-complete problems because speed-ups are very different from an instance to another, and do not depend on instance sizes: for easy instances, speed-ups are usually very low, whereas for hard instances it is not rare to have super-linear speed-ups. Also, really hard instances are not solved within a reasonable amount of time, and speed-ups cannot be computed in this case. Let us illustrate this point on the parallel version of Glasgow (using 32 cores) described in [1]. On easy instances (solved in less than 1s by sequential Glasgow), the speed-up varies between 0.1 and 32, and the average speed-up is close to 1. On hard instances (that are not solved by sequential Glasgow within 1s, but are solved within 1000 seconds), the speed-up varies between 1 and 583, and the average speed-up is 14. However, parallel Glasgow is able to solve instances which are not solved by sequential Glasgow within 1000s and, if we include these instances, the average speed-up becomes greater than 19 (this is a lower bound of the speed-up as we only have a lower bound of the time of sequential Glasgow for unsolved instances). This shows us that the average speed-up does not give a clear picture of solver performance. Better insights are given by scatter plots that compare times on a per instance basis (as done in Fig. 5), or by the aggregate speed-up measure introduced in [12], which measures timeout ratio for solving a same number of instances. For instance, Sequential Glasgow solves $14,356$ instances within 1000s, and the hardest of these instances is solved in 939s. Parallel Glasgow solves $14,356$ instances within a timeout of 19s, and this gives an aggregate speed-up of $939/19 = 49$.

References

1. Archibald, B., Dunlop, F., Hoffmann, R., McCreesh, C., Prosser, P., Trimble, J.: Sequential and parallel solution-biased search for subgraph algorithms. In: 16th Int. Conf. on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (2019)
2. Audemard, G., Lecoutre, C., Modeliar, M.S., Goncalves, G., Porumbel, D.: Scoring-based neighborhood dominance for the subgraph isomorphism problem. In: 20th Int. Conf. on Principles and Practice of Constraint Programming CP. pp. 125–141 (2014)
3. Bombieri, N., Bonnici, V., Giugno, R.: Parallel searching on biological networks. In: 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP. pp. 307–314. IEEE (2019)
4. Bonnici, V., Giugno, R.: On the variable ordering in subgraph isomorphism algorithms. IEEE/ACM Trans. Comput. Biology Bioinform. 14(1), 193–203 (2017)

5. Carletti, V., Foggia, P., Saggese, A., Vento, M.: Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. IEEE Trans. Pattern Anal. Mach. Intell. 40(4), 804–818 (2018)
6. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: 12th Int. Joint Conf. on Artificial Intelligence IJCAI. pp. 331–340 (1991)
7. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. IJPRAI 18(3), 265–298 (2004)
8. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub)graph isomorphism algorithm for matching large graphs. IEEE Trans. Pattern Anal. Mach. Intell. 26(10), 1367–1372 (2004)
9. Damiand, G., Solnon, C., de la Higuera, C., Janodet, J.C., Samuel, E.: Polynomial algorithms for subisomorphism of nD open combinatorial maps. Computer Vision and Image Understanding (CVIU) 115(7), 996–1010 (2011)
10. De Santo, M., Foggia, P., Sansone, C., Vento, M.: A large database of graphs and its use for benchmarking graph isomorphism algorithms. Pattern Recogn. Lett. 24(8), 10671079 (2003)
11. Erdős, P., Rényi, A.: On random graphs I. Publicationes Mathematicae 6, 290–297 (1959)
12. Hoffmann, R., McCreesh, C., Ndiaye, S.N., Prosser, P., Reilly, C., Solnon, C., Trimble, J.: Observations from parallelising three maximum common (connected) subgraph algorithms. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR. LNCS, vol. 10848, pp. 298–315. Springer (2018)
13. Knuth, D.E.: The Stanford GraphBase - a platform for combinatorial computing. ACM (1993)
14. Kotthoff, L., McCreesh, C., Solnon, C.: Portfolios of subgraph isomorphism algorithms. In: 10th Int. Conf. on Learning and Intelligent Optimization LION. LNCS, vol. 10079, pp. 107–122. Springer (2016)
15. Larrosa, J., Valiente, G.: Constraint satisfaction algorithms for graph pattern matching. Mathematical. Structures in Comp. Sci. 12(4), 403–422 (2002)
16. McCreesh, C., Prosser, P.: A parallel, backjumping subgraph isomorphism algorithm using supplemental graphs. In: Principles and Practice of Constraint Programming, LNCS, vol. 9255, pp. 295–312. Springer (2015)
17. Mccreesh, C., Prosser, P., Solnon, C., Trimble, J.: When Subgraph Isomorphism is Really Hard, and Why This Matters for Graph Databases. Journal of Artificial Intelligence Research 61, 723 – 759 (2018)
18. Solnon, C.: Alldifferent-based filtering for subgraph isomorphism. Artif. Intell. 174(12-13), 850–864 (2010)
19. Solnon, C., Damiand, G., de la Higuera, C., Janodet, J.: On the complexity of submap isomorphism and maximum common submap problems. Pattern Recognition 48(2), 302–316 (2015)
20. Ullmann, J.R.: An algorithm for subgraph isomorphism. J. ACM 23(1), 31–42 (1976)
21. Zampelli, S., Deville, Y., Solnon, C.: Solving subgraph isomorphism problems with constraint programming. Constraints 15(3), 327–353 (2010)
22. Zampelli, S., Deville, Y., Solnon, C., Sorlin, S., Dupont, P.: Filtering for subgraph isomorphism. In: Principles and Practice of Constraint Programming - CP 2007. LNCS, vol. 4741, pp. 728–742. Springer (2007)