Enhancing PDR/IC3 with Localization Abstraction

Yen-Sheng Ho Alan Mishchenko Robert Brayton Niklas Een*

Department of EECS, UC Berkeley Google Inc.

{ysho, alanmi, brayton}@berkeley.edu niklas@een.se

Abstract

Property Directed Reachability (aka PDR/IC3) is the strongest engine presently used in formal verification tools. Localization abstraction is a way to reduce the complexity of a verification problem by cutting away irrelevant logic. Both methods are effective when used independently or when an abstracted model is passed to PDR. This paper proposes a new method of combining them by minimally changing the PDR engine. The method differs from previous work, which requires a larger implementation effort. Experiments show that the integrated engine is, on average, stronger than the baseline and produces inductive invariants that are smaller and depend on fewer variables, making them more useful in design analysis and debugging.

1. Introduction

Property Directed Reachability (PDR) is an elegant and powerful engine pioneered in 2010 by Aaron Bradley under the name of IC3 [3][5] and improved by ongoing research [1][7][8][9][11][13][18]. The engine continues to receive attention because of its ability to solve hard model checking problems, both satisfiable and unsatisfiable. The inductive invariants computed as a by-product of solving unclassifiable verification instances with PDR, are useful as certificates of correctness and as a means for design analysis. For example, the support of an invariant indicates what parts of the design are needed to prove the property.

Localization abstraction [20][6][15][16] is a method aimed at reducing the complexity of a verification instance by removing some logic. The remaining part of the instance is called an *abstraction.* An abstraction typically contains the property output of the original instance along with logic nodes and flip-flops deemed necessary to prove the property. The connections to the removed logic are called *pseudo primary inputs* (PPIs) and treated as free variables, which increases the behavior. As a result, if the abstraction is proved, the verification problem is solved. If a counter-example (CEX) is discovered, *abstraction refinement* adds new logic to rule it out, before a new proof is attempted. A taxonomy of abstraction methods can be found in [15].

The contribution of this paper is integrating PDR with an adaptive localization abstraction. As a result, the PDR engine is minimally modified to perform on-the-fly abstraction while solving a verification instance. The modified engine is capable of solving more problem instances than the baseline engine. Moreover, inductive invariants computed by the modified PDR are on average about 20% smaller than those computed by the original PDR. A smaller invariant is more representative of the verification problem and more suitable for design analysis.

The paper is organized as follows. Section 2 contains relevant background. Section 3 contains an overview of the original PDR algorithm. Section 4 describes modifications to the original algorithm needed to integrate it with abstraction. Section 5 compares the approach presented in this paper with previous work. Section 6 shows experimental results. Section 7 concludes the paper.

2. Background

We assume that the reader is familiar with the tenets of safety model checking and the implementation of PDR/IC3 [3][5][7]. Below we review PDR/IC3 as presented in [7] before discussing minimal changes to the baseline engine needed to enable on-the-fly abstraction.

It is assumed that the verification problem is presented to a model-checking engine as a sequential logic circuit with an all-0 initial state having a single property output. If the property holds, the output of the logic circuit evaluates to 0 for any state reachable from the initial state. If the property fails, the engine returns a CEX, which is a sequence of inputs taking the design from the initial state into a state where the output evaluates to 1. If the initial state is not constant-0, the sequential circuit can be equivalently transformed to ensure that the initial state is 0. Similarly, if there are more outputs than one, the problem can be transformed by ORing individual outputs together.

3. Overview of PDR

A simplified block diagram of PDR is shown in Figure 1.

The PDR engine performs an incremental computation of sets of CNF clauses over-approximating reachable states in each timeframe. A new frame is opened and bad states (the states where the property fails under some input) of this timeframe are enumerated. For each bad state, PDR checks whether it overlaps with the initial state, and if so, the verification problem is satisfiable and PDR terminates. If a bad state does not overlap with the initial state, ternary simulation is performed to expand a state minterm into a state cube, such that for all minterms belonging to this cube (including the original minterm), the property fails.

The expanded cube composed of bad states is called a *proof obligation* (POB) because, to prove the property, we need to show that none of the states contained in this cube are reachable from the initial state. The POBs are ordered in each time frame by the time they are generated .

* This work was done by the author while he was employed by UC Berkeley.

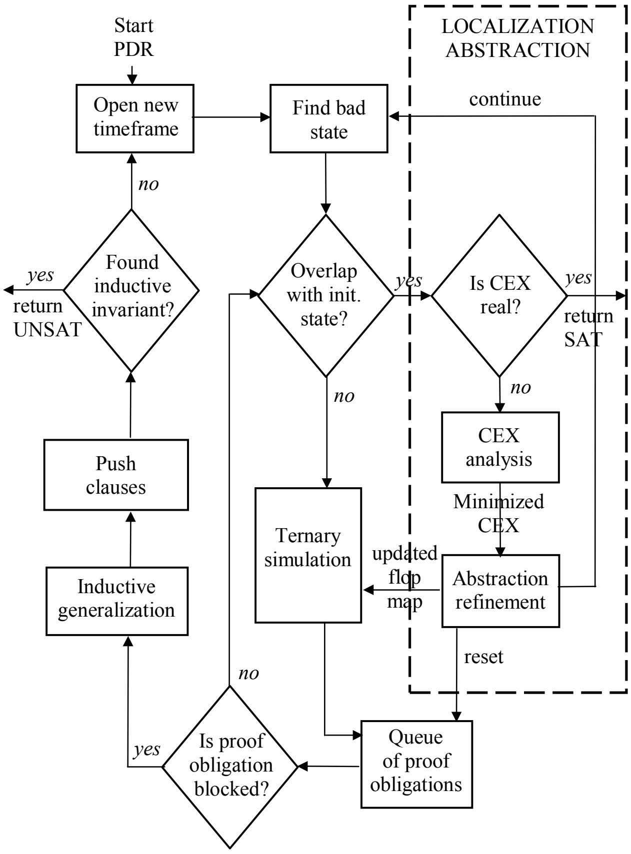

Figure 1. Overview of the PDR/IC3 algorithm.

The PDR engine retrieves POBs from the queue, one at a time, and checks if they can be blocked. A POB is *blocked* if all the previous states that reach the POB are ruled out by the reachable-state over-approximation computed so far. If the POB is not blocked, then there is a previous state, from which at least one state in the POB can be reached. This state is checked for being an initial state and, if not, a new POB is generated and queued.

If, on the other hand, the POB is blocked, it is generalized into a clause, which is added to the reachable state over-approximation under construction. When PDR has finished blocking all bad states in a given timeframe, and the queue of proof-obligations is empty, PDR attempts to move the clauses forward, that is, to prove that the clauses holding in a given timeframe, also hold in the next timeframe. If, in any timeframe, *all* the computed clauses are moved, these clauses form a *property-directed inductive invariant*.

The invariant is a Boolean function defined over the flip-flop output variables, which is characterized as follows: (a) it contains the initial state; (b) it does not contain bad states; and (c) for each state contained in the invariant, the next states reachable from it are contained in the invariant. When such an inductive invariant is found, the property is proved because there does not exist a sequence of reachable states, originating in an initial state, leading to a bad state.

4. Proposed algorithm

The performance of PDR is hampered when it takes a long time to converge on an inductive invariant. There can be several reasons for this: (1) the reachable state space may be irregular making it hard to separate reachable states from bad states by using a two-level representation such as a set of clauses; (2) it may be possible to express the inductive invariant in the two-level form but PDR fails to find it because the state space exploration is unfocused.

It may be hard to mitigate the first limitation of PDR without developing a brand-new engine, which computes an over-approximation in a non-clausal form. In this paper, we address the second limitation by making state-space exploration more focused. To this end, localization abstraction is added to the PDR engine, making the set of flop variables participating in the clauses grow in a more predicable manner, compared to the original engine. As a result, the state-space exploration becomes more focused and more likely to converge to an inductive invariant. The modified engine is PDR with Abstraction (PDRA).

The modifications needed to go from PDR to PDRA are shown in the block diagram in Figure 1 as boxes inside the dashed rectangle. The changes comprise counter-example (CEX) analysis and CEX-based abstraction refinement, affecting the PDR engine components as described below.

PDRA maintains an additional data-structure called *flop map*, remembering what flip-flops are used in the abstraction. A flip-flop is *used in the abstraction* if there is a clause containing a literal of the corresponding flop variable in any timeframe. Otherwise, a flop is not used. The flop map is empty at the beginning. It is incrementally updated by the abstraction refinement while enumerating bad states. The set of flops included in the flop map does not grow monotonically from frame to frame because the clauses containing certain flop variables may be subsumed later by stronger clauses, not containing these variables. As a result, some flop variables present in the flop map at an earlier time frame may disappear in the later time frames.

PDRA uses the flop map during ternary simulation. In PDR, ternary simulation converts a bad-state minterm into a bad-state cube while removing as many flop variables as possible in a given order. If a flop variable cannot be removed, it is added to the POB and may later appear in the generalized clause when the POB is blocked. As a result, even if a flop variable is not used in any of the clauses so far, the original PDR adds it whenever needed. In contrast, PDRA treats flops not used in the abstraction as pseudo-primary inputs (PPIs). This allows the derived clauses to continue depending only on the flops used in the abstraction at the risk of running into a spurious CEX.

This is why, when a CEX is detected by PDRA, a dedicated CEX analysis is performed, as described in [16] (Section 3.3 “Priority based abstraction refinement”). The analysis results in a set of PPIs needed for making the CEX fail the property output. These PPIs correspond to flops absent in the current abstraction. The next-state functions of these flops are added to the abstraction to rule out the given spurious CEX. Other spurious CEXes may be generated and ruled out in a similar manner.

At some point (when enough next-state logic functions have been added to the current abstraction) PDRA finishes the current timeframe without spurious CEXes. Then an additional cleanup step is done where PDRA checks if the flops added by refinement appear in the generated clauses. Frequently, some flops do not appear in these clauses and can be removed from the flop map before PDRA opens the next timeframe. The CEX-based refinement is the same as the refinement step in GLA [16], while the cleanup step is analogous to the proof-based cleanup in GLA.

In summary, PDRA maintains a data structure called flop map to remember what flops are used in the abstraction. The flop map is empty at the beginning and grows from one frame to another. When a new timeframe is opened, PDRA tries to maintain the set of used flops unchanged compared to the previous timeframe. To this end, additional flops required by ternary simulation are treated as PPIs. Once a spurious CEX is found, refinement is performed, the queue of POBs is emptied, and the enumeration of bad states continues, as shown by the block contained within the dotted line in Figure 1. If a real CEX or an inductive invariant is discovered, PDRA terminates.

The modifications described in this section can be implemented on top of an available PDR engine, such as the one in ABC [2]. The implementation requires adding approximately 80 lines of C language code, not counting the CEX analysis code, which is reused from [16].

5. Comparison with previous work

The proposed method comes close to some previous work [1][19][14][8]. In particular, [1] integrates PDR and localization abstraction at a high level, by making these two engines exchange information. Flop variables participating in bounded PDR runs are scored and used to guide the abstraction. This is different from our approach, which essentially consists of building a minimalistic localization abstraction engine within the PDR engine.

The first fully integrated approach combining PDR with localization abstraction was presented in [19]. However, the abstraction used there is "variable timeframe", as defined in [15], that is, in each timeframe, the abstraction states what flop outputs should be used to express clauses in the given timeframe. Our method is based on a simpler "fixed timeframe" abstraction used in [16].

The work of [14] combines PDR with abstraction by targeting datapath flip-flops to be abstracted. In contrast, our approach does not have information to distinguish control logic and datapath. It tries to abstract any flops not used in a precise over-approximation of the reachable state space. We believe that adopting the principles of [14] could make our approach even more effective.

Another integration of PDR with localization abstraction is described in [8]. It uses gate-level abstraction while our approach is flop-level. The difference between the two is discussed in [16]. It is also important to note that our implementation is simpler. Given a clear understanding, our abstraction can be developed on top of a working PDR engine in a matter of hours.

6. Experimental results

PDRA is part of two public verification tools: ABC [2] (command *pdr -t*) and ABC-ZZ [17] (command *treb -abs*). The baseline of *pdr* and *treb* is described in [7].

PDRA has been tested on HWMCC benchmarks [10] with inconclusive results because most of the testcases require preprocessing for PDR to be effective. Moreover, often a test case is solved by one flavor of PDR and not by others, making it hard to compare, except by the sheer number of cases solved.

Table 1 lists the runtimes, in seconds, taken by different PDR flavors to solve 77 unsatisfiable industrial verification instances of unknown origin. Empty entries indicate that the instance is not solved on a Linux workstation in 900 seconds. Table 1 shows several versions of PDR along with their corresponding abstracting versions (*pdr, pdr –t*), (*treb, treb -abs*), and (*pdr –nc, pdr –nct*). The last, *pdr –nc,* is a version of IC3 with improved generalization [11]. As claimed, all three versions were modified fairly easily using the ideas outlined in this paper.

The last row of Table 1 shows that the PDRs with abstraction solve more test cases than the PDRs without abstraction. The final row shows geometric averages of runtime for 41 out of the 77 test cases solved by all six flavors of PDR. The runtime overhead for PDRA is negligible, except for *treb -abs*, which takes 20% more time compared to its baseline, *treb*.

Table 2 compares different flavors of PDR on the 41 commonly solved test cases in terms of the the number of timeframes needed to converge to an invariant (Column "Frames"), and its clause count (Column "Size") and flop count (Column "Supp"). Table 2 demonstrates that when PDRA is used, the number of timeframes increases by about 10% on average, while the number of clauses and flops is reduced by 15-20% on average.

7. Conclusions

The paper describes a practical variation of the known model checking algorithm PDR/IC3. The idea is to add localization abstraction to the baseline algorithm to reduce the set of flop output variables used in the over-approximation. The modified engine performs better in terms of the number of cases solved with a slightly increased runtime. Furthermore, it reduces the size of the inductive invariants, making them more suitable for design analysis and debugging.

Future work will include

- Using structural reverse engineering to detect control flops and target abstraction to include the remaining flops that likely belong to a datapath.
- Exploring different abstraction refinement strategies, which might be better at ruling out counter-examples.
- Developing an application-specific SAT solver to speed up PDR/IC3 with and without abstraction.

8. Acknowledgements

This work was supported in part by SRC contract 2710.001 "SAT-based methods for scalable synthesis and verification" and NSF/NSA grant "Enhanced equivalence checking in cryptoanalytic applications".

The authors thank Zyad Hassan for making publicly available in ABC his implementation of IC3 with improved generalization [11] used in the experiments of this paper.

9. REFERENCES

[1] J. Baumgartner, A. Ivrii, A. Matsliah, and H. Mony. "IC3-guided abstraction". Proc. FMCAD'12, pp. 182–185. http://www.cs.utexas.edu/~hunt/fmcad/FMCAD12/029.pdf

[2] Berkeley Logic Synthesis and Verification Group. *ABC: A System for Sequential Synthesis and Verification*. http://www-cad.eecs.berkeley.edu/~alanmi/abc

[3] A. R. Bradley, "k-step relative inductive generalization," CU Boulder, Tech. Rep., Mar. 2010, http://arxiv.org/abs/1003.3649.

[4] A. R. Bradley, "SAT-based model checking without unrolling". *Proc. VMCAI'11*. http://ecee.colorado.edu/~bradleya/ic3/ic3_bradley.pdf

[5] A. R. Bradley, "Understanding IC3", *Proc. SAT'12*. http://theory.stanford.edu/~arbrad/papers/Understanding_IC3.pdf

[6] N. Een, A. Mishchenko, and N. Amla, "A single-instance incremental SAT formulation of proof- and counterexample-based abstraction", *Proc. FMCAD'10*, pp. 181-188.

[7] N. Een, A. Mishchenko and R. Brayton, "Efficient implementation of property-directed reachability", *Proc. FMCAD'11*. https://people.eecs.berkeley.edu/~alanmi/publications/2011/fmcad11_pdr.pdf

[8] K. Fan, M.-J. Yang, and C.-Y. Huang, "Automatic abstraction refinement of TR for PDR". *Proc. ASP-DAC'16*, pp. 121-126.

[9] A. Griggio and M. Roveri, "Comparing different variants of the IC3 algorithm for hardware model checking", *IEEE TCAD'16*,Vol.35(6).

[10] Hardware Model Checking Competition. http://fmv.jku.at/hwmcc14/

[11] Z. Hassan, A. R. Bradley, and F. Somenzi. "Better generalization in IC3". *Proc. FMCAD'13*, pp.157-164.http://www.cs.utexas.edu/users/hunt/FMCAD/FMCAD13/papers/85-Better-Generalization-IC3.pdf

[12] Y.-S. Ho, P. Chauhan, P. Roy, A. Mishchenko, and R. Brayton, "Efficient uninterpreted function abstraction and refinement for word-level model checking", *Proc. FMCAD'16*.

[13] A. Ivrii and A. Gurfinkel, "Pushing to the top", *Proc. FMCAD'15*, http://www.cs.utexas.edu/users/hunt/FMCAD/FMCAD15/papers/paper39.pdf

[14] S. Lee and K. A. Sakallah, "Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction". *Proc. CAV'14*, pp. 849–865.

[15] A. Mishchenko, N. Een, R. Brayton, J. Baumgartner, H. Mony, and P. Nalla, "Variable time-frame abstraction", *Proc. IWLS'12*, pp. 41-47. https://people.eecs.berkeley.edu/~alanmi/publications/2012/iwls12_vta.pdf

[16] A. Mishchenko, N. Een, R. Brayton, J. Baumgartner, H. Mony, and P. Nalla, "GLA: Gate-level abstraction revisited", *Proc. DATE'13*, pp. 1399-1404. https://people.eecs.berkeley.edu/~alanmi/publications/2013/date13_gla.pdf

[17] N. Een. ABC-ZZ. https://bitbucket.org/niklaseen/abc-zz

[18] M. Suda, "Triggered clause pushing for IC3", 2013, https://arxiv.org/pdf/1307.4966.pdf

[19] Y. Vizel, O. Grumberg, and S. Shoham. "Lazy abstraction and SAT-based reachability in hardware model checking". *Proc. FMCAD'12*, pp. 173–181. https://pdfs.semanticscholar.org/3195/c92c3c821f7f11949c9e99163dacf73bd267.pdf

[20] D. Wang, P.-H. Ho, J. Long, J. H. Kukula, Y. Zhu, H.-K. Tony Ma, R. F. Damiano, "Formal property verification by abstraction refinement with formal, simulation and hybrid engines". *Proc. DAC'01*.

Table 1: Comparing different flavors of PDR in terms of the number of solved cases and runtime on 77 industrial examples (implementations with abstraction, *pdr -t, treb -abs,* and *pdr -nct*, are compared against the baselines, *pdr*, *treb,* and *pdr -nc*).

Test	AND	FF	pdr	pdr -t	treb	treb-abs	pdr -nc	pdr -nct
Ex01	509	142				33.26		
Ex02	509	142				57.80		626.73
Ex03	2602	330	23.47	18.77	31.33	43.94	16.88	39.60
Ex04	2602	330	26.64	24.26	38.75	54.07	26.59	46.50
Ex05	1135	242				317.04		
Ex06	2602	330	29.25	21.49	15.62	45.30	26.42	42.70
Ex07	2602	330	30.08	22.17	22.51	51.75	23.65	50.42
Ex08	1135	242				42.86		
Ex09	19886	782		47.05		148.37		56.14
Ex10	19387	771	38.58	13.71	18.70	24.76	15.10	15.88
Ex11	15555	607				103.44	546.90	
Ex12	15555	607				101.85	544.19	
Ex13	21772	782		308.74	544.75	143.55	138.14	183.21
Ex14	21302	771	116.40	14.56	26.12	30.49	20.93	23.12
Ex15	15555	607				105.72	549.58	
Ex16	21772	782		304.82	556.40	147.49	155.16	182.67
Ex17	5777	726				728.77	141.88	82.70
Ex18	479	89	0.59	5.31	0.15	6.09	1.01	1.21
Ex19	20068	3785	9.18	54.57	38.59	81.98	7.22	20.27
Ex20	20066	3785	19.53	10.46	28.02	21.71	10.50	6.94
Ex21	20047	3785		11.05		38.42		12.46
Ex22	20098	3795		658.28		840.66		311.08
Ex23	9985	2654		640.58				169.56
Ex24	2122	353	10.85	13.31	20.51	22.63	16.99	18.71
Ex25	5043	869	11.53	15.54	28.69	38.89	24.76	40.91
Ex26	7408	965	41.18	560.69	80.60	885.90	26.54	
Ex27	18347	1207	142.47	154.26	243.24	515.71	155.65	167.72
Ex28	1755	384		18.66	74.78	46.32	16.12	16.54
Ex29	1746	383		3.72		16.97		23.31
Ex30	11945	781	14.63	13.42	24.01	26.05	16.51	17.69
Ex31	4452	731		50.33		29.82	167.96	34.09
Ex32	1979	368	89.62	79.61	40.55	63.09	38.84	97.01
Ex33	1917	360	58.79	66.04	38.47	56.96	36.20	56.58
Ex34	1840	348	54.29	51.00	64.73	40.53	30.73	55.92
Ex35	1762	335	20.74	29.36	39.28	46.66	24.54	22.53
Ex36	1697	327	17.53	32.17	28.92	29.61	44.57	18.20
Ex37	2675	178	380.46	284.26				
Ex38	2360	178	600.22		279.76	289.69	321.80	275.02
Ex39	1973	146	70.55	61.12	51.19	110.57	123.74	148.24

Table continues on the right hand side

Test	AND	FF	pdr	pdr -t	treb	treb-abs	pdr -nc	pdr -nct
Ex40	36851	2434				348.63		316.15
Ex41	36851	2434				92.10		
Ex42	9895	2249		37.53	14.25	4.56	37.73	8.38
Ex43	9897	2249				6.37	322.77	382.68
Ex44	36851	2434				353.64		314.82
Ex45	9460	1564	28.40	8.14	70.10	52.34	32.46	16.31
Ex46	531	131	2.55	4.34	4.57	6.00	4.93	6.61
Ex47	920	231	9.38	8.63	15.79	20.06	12.21	8.90
Ex48	952	249	24.80	34.62	120.18	36.98	19.84	22.09
Ex49	2052	413			52.44		36.15	74.71
Ex50	1072	253	24.83	38.27	67.43	43.76	21.77	21.98
Ex51	952	249	28.73	22.72	98.06	27.39	14.10	24.99
Ex52	930	241	9.1	17.62	27.44	19.70	18.48	11.84
Ex53	890	229	27.71	19.70	31.32	22.88	15.51	16.64
Ex54	920	231	9.91	8.79	15.63	20.19	11.94	8.85
Ex55	934	239	11.12	18.47	20.36	17.36	15.97	15.90
Ex56	952	249	35.61	27.54	33.61	27.27	19.31	17.22
Ex57	1948	397				297.88	44.77	72.25
Ex58	872	221	16.83	13.34	39.24	13.47	12.24	12.58
Ex59	966	237	30.29	18.57	33.86	45.23	27.67	18.70
Ex60	952	249	21.55	20.16	90.53	25.97	20.96	17.26
Ex61	1050	183	0.46	1.98	4.48	19.74	0.77	0.40
Ex62	1533	252			26.02	32.38	7.28	6.47
Ex63	3632	521	103.22	166.92	180.20	358.97	308.01	287.2
Ex64	1600	309	5.00	1.62	10.92	3.53	4.61	1.74
Ex65	1189	227	80.72	104.11	55.66	84.38	20.07	27.4
Ex66	9422	1324	108.60	165.03	116.82	267.66	148.11	158.88
Ex67	6199	972	873.41	461.90			271.85	200.09
Ex68	1233	171		480.73			798.93	
Ex69	16745	3113		284.45		308.88	281.91	406.61
Ex70	16700	3107	101.77			422.50	117.75	157.22
Ex71	16701	3107		502.31		104.58	45.32	70.61
Ex72	16701	3107	221.54		135.05	140.39	22.70	149.08
Ex73	12049	2389			151.12	239.02	20.85	244.87
Ex74	541	76	4.73	13.93	1.04	51.50	7.44	17.04
Ex75	528	76	10.05	8.69	1.13	48.31	7.17	13.51
Ex76	1228	208		688.81			257.81	419.00
Ex77	1177	195	2.75	1.69	84.52	2.12	8.64	1.76
Solved			47	58	51	71	64	67
Time, %			1.000	1.047	1.400	1.812	0.973	1.038

Table 2: Comparing different flavors of PDR in terms of the frame count and the invariant size on 41 industrial examples (implementations with abstraction, *pdr -t, treb -abs,* and *pdr -nct*, are compared against the baselines, *pdr*, *treb,* and *pdr -nc*).

			pdr			pdr -t			treb			treb -abs			pdr -nc			pdr -nct		
Test	AND	FF	Frame	Size	Supp	Frame	Size	Supp	Frame	Size	Supp	Frame	Size	Supp	Frame	Size	Supp	Frame	Size	Supp
Ex03	2602	330	9	4222	228	15	3929	178	11	3906	196	12	3914	179	9	3998	208	9	4099	178
Ex04	2602	330	15	4108	228	16	4069	186	11	3926	203	14	4047	193	15	4112	226	16	4097	206
Ex06	2602	330	11	4197	209	14	4073	178	15	3858	184	10	3879	193	11	4127	206	16	4096	186
Ex07	2602	330	12	4285	256	13	4110	178	10	3845	194	14	3945	198	15	4151	236	18	4120	196
Ex10	19387	771	4	3104	379	3	2626	99	3	2577	103	3	2562	96	4	2824	177	3	2627	99
Ex14	21302	771	4	3504	442	4	2636	99	3	2587	108	3	2563	96	4	2798	135	3	2640	99
Ex18	479	89	14	74	65	20	306	61	11	53	48	25	140	58	15	139	66	17	110	46
Ex19	20068	3785	33	661	256	62	1194	227	35	359	192	65	276	161	30	356	267	57	348	215
Ex20	20066	3785	60	1285	382	107	523	42	59	486	98	63	212	43	74	693	121	75	228	40
Ex24	2122	353	11	2134	242	13	1932	191	7	2014	237	9	1782	167	10	2104	230	12	1972	166
Ex25	5043	869	4	4123	60	7	4139	68	4	4136	58	8	4130	67	4	4132	59	7	4182	110
Ex27	18347	1207	17	2403	1077	17	2457	1077	17	2217	1077	17	1378	1078	17	2410	1077	17	2432	1077
Ex30	11945	781	8	634	247	9	603	210	8	582	240	8	563	187	9	601	243	9	602	222
Ex32	1979	368	24	3398	339	50	2466	340	19	1981	339	26	1594	339	22	1942	338	44	2447	340
Ex33	1917	360	21	3174	333	29	2732	331	21	2184	331	24	1553	328	21	1955	336	22	2389	333
Ex34	1840	348	44	1930	320	36	2462	317	26	2103	315	20	1744	315	36	1375	320	40	1919	320
Ex35	1762	335	22	1619	310	30	1737	305	42	1409	307	24	1519	302	22	1700	305	20	1571	305
Ex36	1697	327	22	1271	298	26	1635	298	18	1257	292	20	1456	296	25	1984	299	28	1105	298
Ex39	1973	146	8	4534	137	8	4096	126	8	3746	132	8	3859	127	9	3898	137	8	4045	135
Ex45	9460	1564	59	1022	246	58	659	208	67	909	209	62	743	208	80	1121	265	58	865	207
Ex46	531	131	8	1056	120	9	1168	121	8	1057	119	9	950	116	8	1186	119	9	1085	118
Ex47	920	231	14	1637	174	14	1464	169	14	1664	186	20	1595	167	17	1237	179	19	1179	169
Ex48	952	249	15	2364	233	20	3074	235	15	4933	235	14	2975	228	17	1833	233	19	2090	237
Ex50	1072	253	19	2208	236	17	2949	230	16	2532	232	16	2510	229	15	1835	239	18	1785	237
Ex51	952	249	16	2853	234	21	2485	231	15	3900	234	13	1781	229	21	1418	235	19	2427	237
Ex52	930	241	15	1596	192	21	1972	193	16	3516	216	15	2037	196	16	1749	206	17	1222	191
Ex53	890	229	20	2831	212	16	2459	212	15	2452	207	20	1455	205	17	1812	214	16	1759	208
Ex54	920	231	14	1637	174	14	1464	169	14	1664	186	20	1595	167	17	1237	179	19	1179	169
Ex55	934	239	18	1597	186	15	2294	192	17	2781	206	18	1336	183	23	1846	195	17	1953	182
Ex56	952	249	21	3015	233	21	2369	228	21	2907	234	16	1853	231	21	1446	235	17	2154	229
Ex58	872	221	15	2326	195	19	1789	170	16	2702	198	15	1402	152	21	1428	193	17	1341	175
Ex59	966	237	19	2489	220	17	2580	217	15	2521	219	18	2665	219	20	1975	223	18	1968	219
Ex60	952	249	15	2653	233	17	2629	233	17	4232	232	19	1752	227	17	2285	234	17	1859	224
Ex61	1050	183	8	124	84	10	109	67	13	123	81	14	362	82	11	101	76	10	84	53
Ex63	3632	521	95	1441	513	116	1460	512	122	1222	509	115	1241	510	177	1741	514	152	1670	513
Ex64	1600	309	81	303	138	15	321	136	81	353	138	12	244	130	65	306	140	19	302	139
Ex65	1189	227	9	2831	216	10	2974	217	10	2602	216	9	1801	217	9	1422	216	9	1626	216
Ex66	9422	1324	18	1440	1323	25	1455	1323	16	1405	1323	18	1414	1324	19	1400	1323	22	1394	1323
Ex74	541	76	43	23	15	179	10	17	39	70	18	416	12	18	54	63	21	160	67	24
Ex75	528	76	53	17	15	109	19	17	39	83	19	369	12	17	61	65	21	142	67	22
Ex77	1177	195	21	482	139	16	304	66	28	4088	138	13	258	64	28	926	178	15	313	68
Geo			1.000	1.000	1.000	1.161	0.976	0.829	0.968	1.089	0.886	1.098	0.798	0.816	1.085	0.926	0.948	1.130	0.871	0.843