IPC: A Benchmark Data Set for Learning with Graph-Structured Data

Patrick Ferber [1] **Tengfei Ma** [2] **Siyu Huo** [2] **Jie Chen** [2 3] **Michael Katz** [2]

Abstract

Benchmark data sets are an indispensable ingredient of the evaluation of graph-based machine learning methods. We release a new data set, compiled from International Planning Competitions (IPC), for benchmarking graph classification, regression, and related tasks. Apart from the graph construction (based on AI planning problems) that is interesting in its own right, the data set possesses distinctly different characteristics from popularly used benchmarks. The data set, named IPC, consists of two self-contained versions, grounded and lifted, both including graphs of large and skewedly distributed sizes, posing substantial challenges for the computation of graph models such as graph kernels and graph neural networks. The graphs in this data set are directed and the lifted version is acyclic, offering the opportunity of benchmarking specialized models for directed (acyclic) structures. Moreover, the graph generator and the labeling are computer programmed; thus, the data set may be extended easily if a larger scale is desired. The data set is accessible from `https://github.com/IBM/IPC-graph-data`.

1. Introduction

Benchmark data sets are indispensable in the evaluation of machine learning models for graph-structured data. With the surging interest in graph representation learning, a rich collection of data sets constructed from real-life applications becomes important for the validation of effectiveness of any existing or newly proposed method, and for demonstrating its widespread applicability. We introduce a new labeled data set, IPC, compiled from AI plannig tasks described in the Planning Domain Definition Language (PDDL) (McDermott, 2000).

[1]University of Basel [2]IBM Research [3]MIT-IBM Watson AI Lab. Correspondence to: Jie Chen <chenjie@us.ibm.com>, Michael Katz <Michael.Katz1@ibm.com>.

Presented at the ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data Copyright 2019 by the author(s).

In this data set, each planning task is represented as a directed graph, which has target values and whose nodes are equipped with features. Planning tasks described in PDDL admit a concise representation in transition graphs, however too big to fit in any conceivable size memory. Recent advances in planning allow to encode some structural information of a task in graphs of manageable size. Two examples are the problem description graph (Pochter et al., 2011), for a *grounded* task representation, and the abstract structure graph (Sievers et al., 2019b), for a *lifted* representation. Hence, our data set consists of two versions of graphs (IPC-grounded and IPC-lifted) for the same set of tasks; each version may be used independently. There are 2439 planning tasks in total, pre-split for training, validation, and testing. Moreover, the lifted version is acyclic.

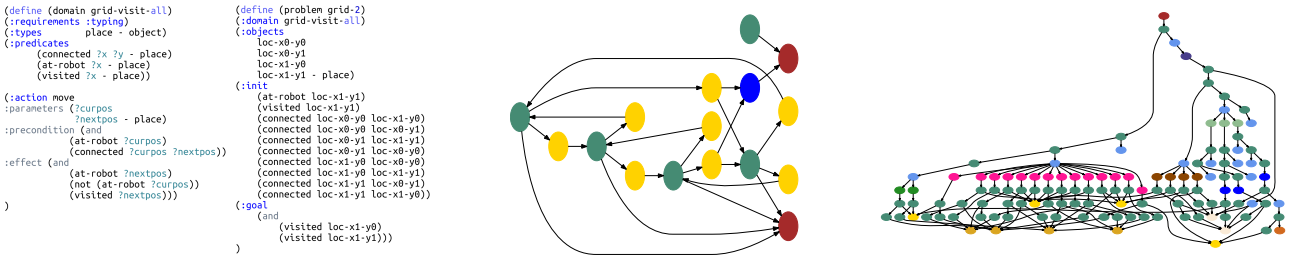
Accompanied with the tasks are performance results for 17 cost-optimal domain-independent planners, each of which attempts to solve a task under a timeout limit $T = 1800$ seconds. Hence, the target values for each graph are the CPU times of these planners, gathered on the same hardware. For practical reasons, if a planner cannot solve the task before timeout, the target value is artificially set as 10000.

The background on AI Planning and graph construction, including example problem domains, node feature definition, and how the data set can be extended, are presented in Section 2. The characteristics of this data set are different in several aspects from those of the commonly used benchmark data sets for graph kernels and graph neural networks: The sizes of our graphs not only are substantially larger but also vary significantly. The imposed challenges and implications are elaborated in Section 3. We present an example use of the data set in Section 4 and conclude in Section 5.

2. Data Set Construction

PDDL tasks are defined over a first-order language $\mathcal{L}$ that consists of predicates, functions, a set of natural numbers, variables, and constants. Given $\mathcal{L}$, a *normalized* (Helmert, 2009) PDDL task is a tuple $\Pi = \langle \mathcal{O}, \mathcal{A}, I, G \rangle$ of *schematic operators*, *schematic axioms*, *initial state specification*, and *goal specification*, in so-called *lifted* representation.

Most tools for solving the planning problems (a.k.a. planners) perform grounding as a first step, followed by trans-



(a) Problem (in PDDL)

```
(define (domain grid-visit-all)
(:requirements :typing)
(:types        place - object)
(:predicates
      (connected ?x ?y - place)
      (at-robot ?x - place)
      (visited ?x - place))

(:action move
:parameters (?curpos
             ?nextpos - place)
:precondition (and
            (at-robot ?curpos)
            (connected ?curpos ?nextpos))
:effect (and
            (at-robot ?nextpos)
            (not (at-robot ?curpos))
            (visited ?nextpos)))
)
```

```
(define (problem grid-2)
(:domain grid-visit-all)
(:objects
    loc-x0-y0
    loc-x0-y1
    loc-x1-y0
    loc-x1-y1 - place)
(:init
    (at-robot loc-x1-y1)
    (visited loc-x1-y1)
    (connected loc-x0-y0 loc-x1-y0)
    (connected loc-x0-y0 loc-x0-y1)
    (connected loc-x0-y1 loc-x1-y1)
    (connected loc-x0-y1 loc-x0-y0)
    (connected loc-x1-y0 loc-x0-y0)
    (connected loc-x1-y0 loc-x1-y1)
    (connected loc-x1-y1 loc-x0-y1)
    (connected loc-x1-y1 loc-x1-y0))
(:goal
    (and
        (visited loc-x1-y0)
        (visited loc-x1-y1)))
)
```

(b) Grounded graph

(c) Lifted graph

Figure 1. An example planning task (described by using PDDL) and the constructed graphs.

lation into a SAS$^+$ language (Bäckström & Nebel, 1995). In SAS$^+$, a task $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{A}, s_0, s_\star \rangle$ consists of finite-domain *variables*, *ground operators*, *ground axioms*, *initial state*, and *the goal*.

Our data set consists of two graphical representations per planning task. These representations losslessly encode the information in the planning task and are often used for the computation of *structural symmetries* (Shleyfman et al., 2015). The graph obtained from the grounded representation SAS$^+$ is called the *problem description graph (PDG)* (Pochter et al., 2011). We present here the definition extended to support conditional effects and axioms and refer the reader to Sievers et al. (2019a) for further details.

Definition 1 *Let* $\Pi = \langle \mathcal{V}, \mathcal{V}_d, \mathcal{O}, \mathcal{A}, s_d, s_0, s_\star \rangle$ *be a* SAS$^+$ *task. The* problem description graph *of* Π *is the digraph* $\langle N, E \rangle$ *with nodes*

$$N = \{n_0, n_\star\} \cup \{n_v \mid v \in \mathcal{V}\} \cup N_f \cup N_\mathcal{O} \cup \{n_a \mid a \in \mathcal{A}\},$$

where $N_f = \{n_v^d \mid v \in \mathcal{V}, d \in \mathit{dom}(v)\}$ *and* $N_\mathcal{O} = \{n_o \mid o \in \mathcal{O}\} \cup \{n_o^e \mid o \in \mathcal{O}, e \in \mathit{effs}(o)\}$*, and edges*

$$E = E_0 \cup E_\star \cup E_v \cup E_a \cup E_o, \text{ where}$$

$$\begin{aligned}
E_0 &= \{\langle n_0, n_v^d \rangle \mid s_0[v] = d\} \\
E_\star &= \{\langle n_g, n_v^d \rangle \mid v \in \mathit{vars}(s_\star), s_\star[v] = d\} \\
E_v &= \{\langle n_v, n_v^d \rangle \mid d \in \mathit{dom}(v)\} \\
E_a &= \{\langle n_a, n_v^d \rangle \mid a \in \mathcal{A}, v \in \mathit{vars}(\mathit{pre}(a)), \mathit{pre}(a)[v] = d\} \\
&\quad \cup \{\langle n_a, n_v^d \rangle \mid a \in \mathcal{A}, \mathit{var}(a) = v, \mathit{val}(a) = d\} \\
E_o &= \{\langle n_o, n_v^d \rangle \mid o \in \mathcal{O}, v \in \mathit{vars}(\mathit{pre}(o)), \mathit{pre}(o)[v] = d\} \\
&\quad \cup \{\langle n_o, n_o^e \rangle \mid e \in \mathit{effs}(o)\} \\
&\quad \cup \{\langle n_v^d, n_o^e \rangle \mid \langle c, \cdot, \cdot \rangle \in \mathit{effs}(o), v \in \mathit{vars}(c), c[v] = d\} \\
&\quad \cup \{\langle n_o^e, n_v^d \rangle \mid \langle \cdot, v, d \rangle \in \mathit{effs}(o)\}.
\end{aligned}$$

The graph obtained from the lifted PDDL representations is called the *abstract structure graph (ASG)* (Sievers et al., 2019b). Planning tasks in PDDL can be naturally modeled as abstract structures, which, in turn, can be represented as graphs. In what follows we present the definitions of *abstract structures* and *abstract structure graphs*, referring the reader to Sievers et al. (2019b) for further details.

Definition 2 (Sievers et al., 2019b) *Let* S *be a set of symbols, where each* $s \in S$ *is associated with a* type $t(s)$*. The set of* abstract structures *over* S *is inductively defined as follows:*

- *each symbol* $s \in S$ *is an abstract structure, and*
- *for abstract structures* $A_1, \ldots, A_n$*, the set* $\{A_1, \ldots, A_n\}$ *and the tuple* $\langle A_1, \ldots, A_n \rangle$ *are abstract structures.*

Using the language $\mathcal{L}$ of a PDDL task Π, each part of Π can inductively be defined as an abstract structure, with the symbols of $\mathcal{L}$ forming the basic abstract structures. Finally, abstract structures can be naturally turned into a graph.

Definition 3 (Sievers et al., 2019b) *Let* A *be an abstract structure over* S*. The* abstract structure graph *ASG*$_A$ *is a digraph* $\langle N, E \rangle$*, defined as follows.*

- N *contains a node* A *for the abstract structure* A*. If* N *contains a node for* $A' = \{A_1, \ldots, A_n\}$ *or* $A' = \langle A_1, \ldots, A_n \rangle$*, it also contains the nodes for* $A_1, \ldots, A_n$*.*
- *For every set (sub-)structure* $A' = \{A_1, \ldots, A_n\}$ *there are edges* $A' \to A_i$ *for* $i \in \{1, \ldots, n\}$*.*
- *For every tuple (sub-)structure* $A' = \langle A_1, \ldots, A_n \rangle$*, the graph contains auxiliary nodes* $n_1^{A'}, \ldots, n_n^{A'}$*, an edge* $A' \to n_1^{A'}$*, and edges* $n_{i-1}^{A'} \to n_i^{A'}$ *for* $1 < i \leq n$*. For each component* A_i*, there is an edge* $n_i^{A'} \to A_i$*.*

Note that the edges in ASGs are from the abstract structures to their sub-structures, which results in acyclic graphs.

In both PDG and ASG, the node features are one-hot according to the self-explanatory node type indicated in the above definitions.

The aim in classical planning is to find a sequence of ground operators that, if applied to the initial state, will necessarily transform it into a goal state. Such a sequence is called a

plan. Assigning a quantitative cost to each ground operator, the cost of a plan is defined as the sum over the costs of its operators. The goal of cost-optimal classical planning is to find a provably cheapest plan. There exist dozens if not hundreds of highly parameterized methods for heuristic guidance computation, giving rise to an enormous possible number of planners. As even the classical planning is PSPACE-hard, there cannot be one planner that will work well on all possible planning problems. Thus, finding a planner that works well on a given planning problem is a challenging task.

While planners are often domain-independent, in the sense that they depend only on the information encoded in PDDL, the planning tasks encode computational problems from various domains. These domains range from puzzles or one-person games (e.g., towers of Hanoi, 15-puzzle, freecell, and sokoban), to real-life domains (e.g., task planning and automated control of autonomous systems: greenhouse logistics, rovers, elevators, satellites), as well as emerging domains (e.g., genome editing distance computation).

Many of the existing domains were introduced through International Planning Competitions, which were held regularly since 1998. Each such competition, intended for comparing the performance of domain-independent planners, introduced new, previously unseen domains on which submitted planners were tested. In many cases, the authors of the domains supplied not only the planning tasks, but also the generator that allowed for creating additional tasks. Some of these generators can be found at, e.g., `https://bitbucket.org/planning-researchers/pddl-generators`. Hence, these generators may be used to extend the current data set with little effort, although for benchmarking purpose we did not include any such task in the data set.

3. Statistics

A number of graph statistics, compared with those of commonly used datasets (Kersting et al., 2016) for benchmarking graph kernels and graph neural networks, are reported in Table 1 and Figures 2 and 3 in the supplementary material. Observations follow.

1. **The IPC graphs are significantly larger.** The graphs in other data sets under comparison generally have tens to hundreds of nodes, but 39% of the graphs in IPC-grounded and 63% in IPC-lifted have over 1,000 nodes. The largest graph in IPC-grounded has 87,140 nodes, and the number for IPC-lifted is 238,909.

2. **Note that the size of the largest graph is often the memory bottleneck indicator for graph neural networks, because the batch size is at least this number in stochastic training.** Hence, our data set poses substantial challenges for the computation of many neural graph models.

3. **The sizes of the IPC graphs are highly skewed, compared to those of other data sets.** For many machine learning tasks, especially in the unsupervised setting, the notion of similarity is key to clustering and categorization. When the sizes of two graphs significantly differ, the intuition of similarity is challenged. After all, what does it mean by saying "a graph with 10 nodes is similar to another graph with 100,000 nodes?"

4. **The lifted graphs are the most sparse, compared to the grounded ones and graphs in other data sets.**

5. **Similar to many other data sets, the IPC graphs are not necessarily connected.** However, the main connected component generally dominates. Hence, graph neural networks still suffer the memory bottleneck caused by the exceedingly large graphs.

6. **Despite the difference in size and density, the IPC graphs have a moderate diameter, similar to other data sets.** The number of layers in a graph neural network of neighborhood-aggregation style is often questioned beyond hyperparameter tuning; and speculation attributes to the diameter of the graphs. Meanwhile, it has been widely acknowledged that neighborhood aggregation is a type of Laplacian smoothing and too many layers lead to oversmoothing (Li et al., 2018; Xu et al., 2018; Klicpera et al., 2019). The diameter statistics may be useful for the analysis of the role of small-world structures handled by graph neural networks.

4. Example Use

For an illustration of the use of the data set, we focus on the problem of cost-optimal planning, whose goal is to solve as many tasks by using cost-optimal planners as possible, each given a time limit T. Hence, for each of the 17 target values, we convert it to 0 if the value $\leq T$ and 1 otherwise. For each target, the problem becomes a binary classification and thus a probability value between 0 and 1 is output. We select the planner corresponding to the smallest probability and confirm success if its actual planning time is smaller than the timeout limit T. Test accuracy (percentage of successfully solved tasks) is reported.

Three methods for comparison are (a) an image-based CNN whereby the gray-scale image is converted from the adjacency matrix of the graph; (b) a graph convolutional network (GCN) (Kipf & Welling, 2017) with attention readout; and (c) a gated graph neural network (GG-NN) (Li et al., 2016). For details of the CNN architecture, see Katz et al. (2018).

Table 1. Statistics of IPC, compared with that of the other commonly used benchmark data sets.

	IPC-grounded	IPC-lifted	REDDIT-MULTI-12k	REDDIT-BINARY
Type	directed	DAG	undirected	undirected
#Graphs	2,439	2,439	11,929	2,000
Total #Nodes	6,233,856	9,816,948	4,669,116	859,254
Max #Nodes	87,140	238,909	3,782	3,782
Mean (Std) #Nodes	2,555.9 (6,099.0)	4,025.0 (14,507.6)	391.4 (428.7)	429.6 (554.1)
Mean (Std) Ave Degree[1]	12.3 (131.0)	2.9 (35.1)	4.7 (27.6)	4.6 (41.3)
Mean (Std) #CC[2]	1.09 (0.61)	1.14 (0.49)	2.81 (2.65)	2.48 (2.47)
Mean (Std) Diam[3,4]	8.2 (2.3)	17.1 (1.5)	10.9 (3.1)	9.7 (3.1)

[1] "Ave Degree" is the average node degree (of the undirected version of the graph).
[2] "CC" means connected components (of the undirected version of the graph).
[3] "Diam" means diameter. Because a graph may consist of multiple connected components, we define the diameter as the maximum of the diameters of each connected component.
[4] For large graphs, the diameter is too costly to compute. Hence, for IPC, only the diameters of 94.3% of the graphs are computed. For other data sets, diameters of all graphs are computed.

	COLLAB	NCI1	DD	PROTEINS	ENZYMES	MUTAG
Type	undirected	undirected	undirected	undirected	undirected	undirected
#Graphs	5,000	4,110	1,178	1,113	600	188
Total #Nodes	372,474	122,747	334,925	43,471	19,580	3,371
Max #Nodes	492	111	5,748	620	126	28
Mean (Std) #Nodes	74.5 (62.3)	29.9 (13.6)	106.5 (284.3)	39.1 (45.8)	32.6 (15.3)	18.0 (4.6)
Mean (Std) Ave Degree[1]	132.0 (158.5)	4.3 (1.6)	10.1 (3.4)	7.5 (2.3)	7.6 (2.3)	4.4 (1.5)
Mean (Std) #CC[2]	1 (0)	1.19 (0.57)	1.02 (0.18)	1.08 (0.52)	1.24 (3.61)	1 (0)
Mean (Std) Diam[3,4]	1.9 (0.3)	13.3 (5.1)	19.9 (7.7)	11.6 (7.9)	10.9 (4.8)	8.2 (1.8)

The data set has been pre-split for training, validation, and testing. Table 2 reports the test accuracy. Additionally, we re-split the training/validation combination as a form of cross validation, whereby we fix the test set because it comes from the most recent International Planning Competition. Two forms of random re-splits are possible. One is to preserve the domains of the planning tasks (i.e., tasks from the same domain cannot appear in both training and validation), and the other is free from this restriction. We call the former *domain split* and the latter *random split*. For each type of re-split, we perform ten randomizations. Table 3 reports the test accuracy together with standard deviation. From both tables, one sees that the lifted graphs yield much higher accuracy and GCN outperforms the other two methods.

Table 2. Percentage of solved tasks in the test set.

Method	Grounded	Lifted
CNN	73.1%	86.9%
GCN	80.7%	87.6%
GG-NN	77.9%	81.4%

Table 3. Percentage of solved tasks in the test set (lifted graphs). Multiple training/validation splits.

Method	Domain Splits	Random Splits
CNN	82.1% (6.6%)	86.1% (5.5%)
GCN	85.6% (5.5%)	87.2% (3.5%)
GG-NN	76.6% (5.8%)	74.4% (2.7%)

5. Conclusions

We have described a new data set, IPC, for benchmarking graph-based learning models (e.g., graph kernels and graph neural networks) in classification, regression, and related uses. The graphs are constructed from AI planning tasks appearing in International Planning Competitions, without requiring human efforts for labeling, and may be extended with random instances of planning problems. The data set has distinctively different statistics from other popularly used benchmarks: the graphs are much larger and their sizes vary substantially. Moreover, the lifted version of the data set is comprised of directed acyclic graphs, enabling the development of specialized graph models. We anticipate that the data set is a valuable inclusion to the current collection of commonly used benchmarks for validating the effectiveness of existing and forthcoming graph methods.

References

Bäckström, C. and Nebel, B. Complexity results for SAS^+ planning. 11(4):625–655, 1995.

Helmert, M. Concise finite-domain representations for PDDL planning tasks. *AIJ*, 173:503–535, 2009.

Katz, M., Sohrabi, S., Samulowitz, H., and Sievers, S. Delfi: Online planner selection for cost-optimal planning. In *IPC-9 planner abstracts*, 2018.

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. Benchmark data sets for graph kernels, 2016. URL http://graphkernels.cs.tu-dortmund.de.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Klicpera, J., Bojchevski, A., and Gnnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. In *ICLR*, 2016.

McDermott, D. The 1998 AI Planning Systems competition. 21(2):35–55, 2000.

Pochter, N., Zohar, A., and Rosenschein, J. S. Exploiting problem symmetries in state-based planners. In *AAAI*, 2011.

Shleyfman, A., Katz, M., Helmert, M., Sievers, S., and Wehrle, M. Heuristics and symmetries in classical planning. In *AAAI*, 2015.

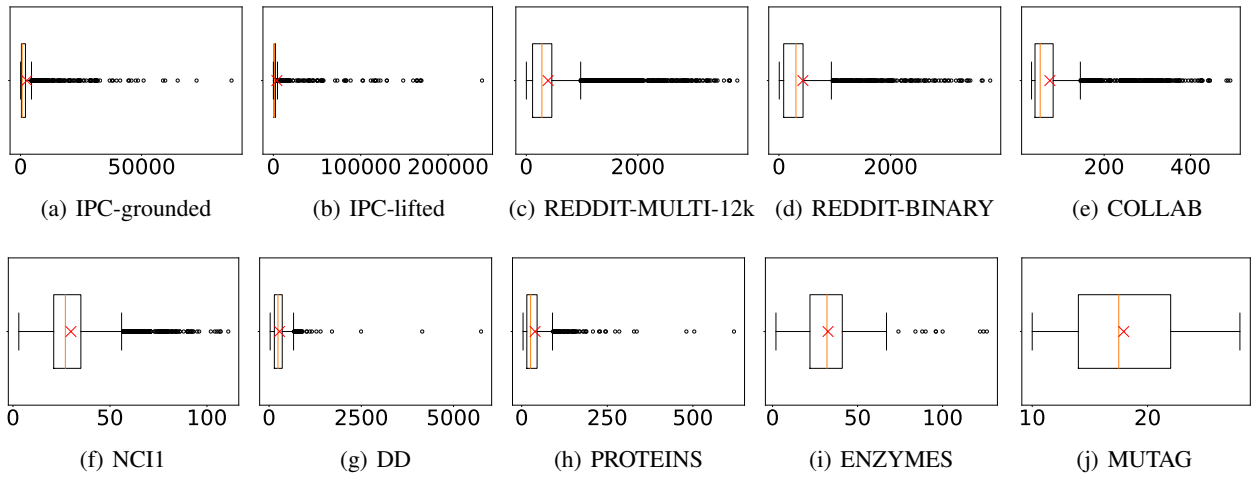
Sievers, S., Katz, M., Sohrabi, S., Samulowitz, H., and Ferber, P. Deep learning for cost-optimal planning: Task-dependent planner selection. In *Proc. AAAI 2019*, 2019a.

Sievers, S., Rger, G., Wehrle, M., and Katz, M. Theoretical foundations for structural symmetries of lifted pddl tasks. In *Proc. ICAPS 2019*, 2019b.

Xu, K., Li, C., Tian, Y., Sonobe, T., ichi Kawarabayashi, K., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

A. Additional Information of Graph Statistics

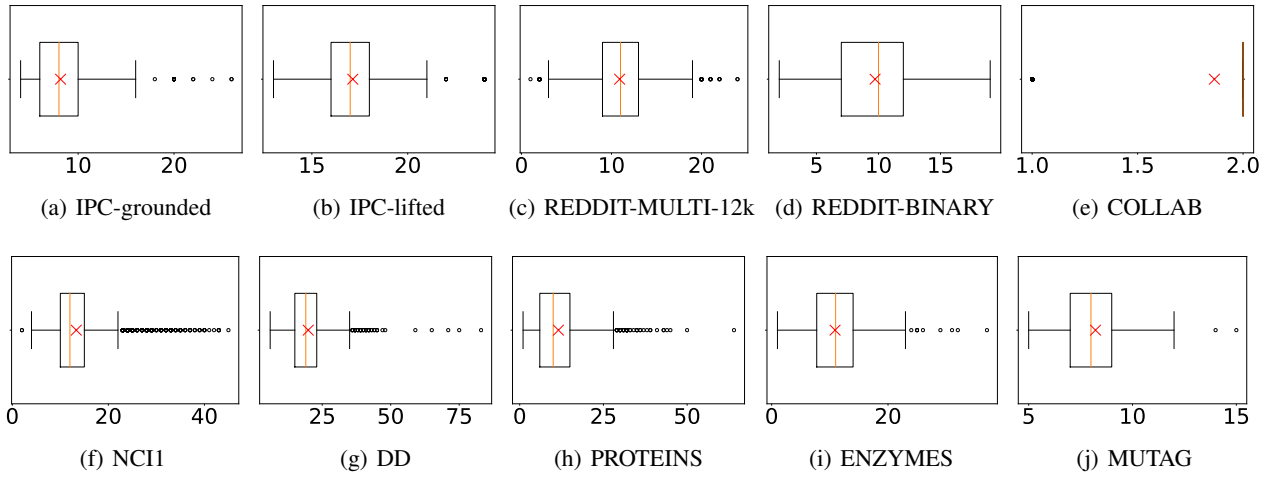
See Figures 2 and 3.



(a) IPC-grounded (b) IPC-lifted (c) REDDIT-MULTI-12k (d) REDDIT-BINARY (e) COLLAB

(f) NCI1 (g) DD (h) PROTEINS (i) ENZYMES (j) MUTAG

Figure 2. Box plot of the graph size distribution. The orange bar marks the median; the red cross the mean.



(a) IPC-grounded (b) IPC-lifted (c) REDDIT-MULTI-12k (d) REDDIT-BINARY (e) COLLAB

(f) NCI1 (g) DD (h) PROTEINS (i) ENZYMES (j) MUTAG

Figure 3. Box plot of the diameter distribution. The orange bar marks the median; the red cross the mean.