

Neural Network Based Approach for Learning Planning Action Models

Alfredo Milani[1], Rajdeep Niyogi[2], and Giulio Biondi[3(✉)]

[1] Department of Mathematics and Computer Science, University of Perugia, Perugia, Italy
milani@unipg.it

[2] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee 247667, India
rajdpfec@iitr.ac.in

[3] Department of Mathematics and Computer Science, University of Florence, Florence, Italy
giulio.biondi@unifi.it

Abstract. Artificial Intelligence (AI) planning technology represents one of the most efficient solutions to guide goal driven agents in domain problems which are mostly characterized by problem solving features. The current proliferation of physical and virtual autonomous agents have therefore triggered a growing interest in the automatic acquisition of action models, which can take advantage of the observation capabilities of agents, which sense world states through their sensors. In this work we present a Neural Network (NN) based approach for learning AI planning action models by observation in noisy environments. The system learns by observing a set of execution patterns of the same action in different contexts. The perceptions of both pre/post action execution states are used to train the NN learning component and are assumed to be affected by noise, which could be due to inaccurate reading or malfunctioning of the sensors. Preliminary experiments shows that the proposed NN learning module seems to be more resilient to unforeseen situations with respect to traditional propositional-based approaches to action model learning.

Keywords: Planning · Neural Networks

1 Introduction

Artificial Intelligence (AI) planning and scheduling has been one the first research area of AI [6] and has obtained many relevant successful results [1]. In recent years the ubiquitous successful application of Neural Networks (NN) based machine learning approaches has given a great contribution to a large number of different AI fields such as visions, speech recognition and synthesis including learning agents behaviour [23]. The successful application of deep NN for learning agent behaviours, such as driving a car or playing a video game console, cannot

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11624, pp. 526–537, 2019.
https://doi.org/10.1007/978-3-030-24311-1_38

be replicated for goal oriented agents, whose utility function is characterized by efficient problem solving features. For instance, a good video-game playing agent is characterized by obtaining a certain utility by a policy for reactive actions, e.g. points [15], while a problem solving agent is primarily characterized by exactly solving a goal, e.g. executing the right plan of sequence of actions to disassemble an engine to reach and substitute a faulty component, rather than doing it partially; in this situation, "nearly reaching the faulty piece" is totally useless. In such cases having a preconditions/effects model of the world to which to apply an AI planning solution is much more efficient and effective, especially in domains where replanning is not allowed or irreversible states exists. In those "problem solving characterized" domains, action knowledge and explicit action reasoning can develop a plan reaching a goal which can change from problem to problem [20], whereas a behavioural agent is basically characterized by the same goal to reach in different context, e.g.deciding an action reactively given a state [16], in order to maximize a smooth utility function.

The current proliferation of physical and virtual autonomous agents has triggered a growing interest in the automatic acquisition of action models, thus allowing rapid adaptation to new agents and avoiding any bias that an expert domain modeller, usual a human, could introduce [21]. The idea is being able to rapidly training a domain learning component by observations [2].

Several approached have been proposed in the planning literature for the problem of learning planning domain by observations [24]; most of the proposed solutions focus on a totally logic based reasoning approach and assume a deterministic and totally observable domain. The typical input of those learners is represented by a sequence of actions [5,19], or perfect state descriptions [13], which is quite unlikely in a real situation. Real world is dynamical, sometimes unpredictable and full of noise, given by the inaccuracy of sensors information.

To be able to do domain learning based on the state actually perceived by an agent would allow to take advantage of the real observation capabilities of agents, which sense the external world through their sensors and build a world state representation upon them [3,22], even if sensors are faulty [27] or perceive unpredictable situations. Possible applications range from supervised-learn applications, where an agent is trained by repeatedly showing real examples of single actions [17,18,25], similarly to teaching a small child to properly manage a spoon, to *self-learn applications*, where an agent in unknown environments observes other agents executing unknown actions, trying to augment the similarity [7,9] between the agent view of the world and the state transition semantic [10,11] embedded in action representation.

Our focus was on developing the ability to predicting the relevant world states changes before and after an action execution, thus allowing the reuse of the learned actions in a planning context. The basic idea to implement this solution to use the pre/post action execution states perceived by the agent to train a NN based learning component where the perceptions can be affected by noise, which could be due to inaccurate reading or malfunctioning of the sensors. The objective is that the trained NN component will be able to predict the state

following the action execution in a given input state, i.e. the *state transition* in search space terms [4,8,14]. The trained NN component can then be used by a planner, e.g. a forward planner [12], as the actual action representation on which planning reasoning can take place. In the next Sect. 2 some basic notions of planning theory are recalled and the main motivations and features of the NN-based learning action model are described; experiments design, evaluation metrics and results are presented and discussed in Sect. 3, together with details of the noise model used. Conclusions are finally drawn in Sect. 4.

2 Learning Planning Action Model

2.1 Planning: States, Actions, Domains

The planning problem, according to well known and classical definitions [6,26], can be characterized as the problem of finding a plan, i.e. a sequence of actions, that, if executed in an *initial state* I will produce a final goal state F, i.e. a state which verifies a certain goal G. More formally in propositional or predicative planning, given a set P of boolean symbols, i.e. propositional symbols or ground predicates, a *state* s is a function $s : P- > \{T, F\}$, which defines a truth assignment for each symbol $p \in P$. An action a is defined by two sets of predicates symbols $pre(a)$ and $post(a)$ respectively called *preconditions* and *postconditions* or *effects* of a, where symbols are assigned either T, *positive* conditions, or F *negative* conditions. An actions a is executable in a state s if all the predicates $pre(a)$ are assigned in state s the truth values that they have in $pre(a)$. Assuming that, in a given planning domain, S is the set of all possible states defined over P and A is the set of all possible ground actions in the domain, then the function $execution : S \times A- > S$, defines the state s_2 resulting from executing $a \in A$ in state $s_1 \in S$, if a is not executable in s_1 then $s_2 = s_1$, i.e. the action does not produce any effect; if a is executable in s_1 then s_2 is obtained by assigning all the symbols in $pre(a)$ according to their assignment in a's postconditions, and leaving all the other symbols assignments unchanged. It is straightforward to see that any action a can be seen as a function which tranforms states into states. The planning problem can then be stated as, given a tuple (P, I, G, A), where P is a set of predicate symbols, I is an *initial state* and G a *goal state* defined on P, A is a set of actions with pre/post conditions on P, the problem of finding a sequence of actions $[a_1, ..., a_n]$, $a_i \in A$ such that a_1 is executable in I, for *in* a_i is executable in $execute(a_{i-1}, execute(a_{i-2}, ..., execute(a_1, I)))$ and $execute(a_n, execute(a_{n-1}, ..., execute(a_1, I))) = G$, i.e. if the sequence of actions transforms the initial state I into a goal state G.

2.2 The Neural Networks Based Action Learning Model

The basic idea of the NN based *action learning model* is to design an agent doing supervised learning by observations, based on the agent's internal boolean predicates state representation. The supervising process is making the learning

agent aware that a particular action a has been executed by labeling a the observed transition of the world state. In other words, the objective is to devise a system which is able to learn by example the transition function for a. Each submitted training example can be modeled by a triple $(s1, s2, a)$, where s_1 represents the state perceived before, and s represents the state perceived after the execution of action a.

It is straightforward to think to apply *Neural Networks* techniques to a complete representation of state predicates, where each pre or post state can be seen as a vector of length $|P|$ of binary bits representing the truth values of the corresponding predicate. If we assume to promote the vector binary components to real numbers (namely 1.0 and 0.0), the problem can therefore be modeled as a *regression task* solved by a NN, where the predicted output state vector is required to be as close as possible to the real expected one after actions execution. A typical way to achieve this goal is using Mean Squared Error (MSE) as loss function during the NN training phase. The network structure will consist of an initial layer with size $=|P|$ and sigmoid activation function, followed by a variable number of hidden layers, i.e. 5, 7 or 10, of size $2 * |P|$ and ReLU activation function, and a final output layer with the same size as the input one.

In the test phase for a triple (s_1, s_2, a), the execution of a given action a in a given state s_1, i.e. submitting a state s_1 to the trained network $NN_trained$, will produce a predicted state vector $s_predicted{=}NN_trained(s_1)$ consisting of real numbers values, within [0,1], for each predicate. A more useful *boolean assignment* s_pred can be obtained by applying to it a *thresholding operation*, with a typical a threshold value $\alpha = 0.5$. After thresholding a bit-by-bit comparison of s_pred with the real expected output state s_2 is performed.

In order to illustrate the concept, let consider the *blocksworld* domain. The state of the world at a given time is usually described by a set of predicates associated to their truth values, i.e. True or False; such a set completely defines the actual state. In particular, in the problem analyzed, three predicates are defined:

- $(ON\ BLOCK_i\ BLOCK_j)$, indicating that $BLOCK_i$ is on top of $BLOCK_j$
- $(TABLE\ BLOCK_i)$, indicating that $BLOCK_i$ sits on the table
- $(CLEAR\ BLOCK_i)$, indicating that there is no other $BLOCK_j$ on top of $BLOCK_i$.

Let N be the number of blocks in the world; the set P of all the possible predicates is composed of a ON predicate for each pair in the cartesian product of the set of blocks (excluding self-pairs, i.e. $(ON\ BLOCK_i\ BLOCK_i)$), plus a $(TABLE\ BLOCK_i)$ and a $(CLEAR\ BLOCK_i)$ for each block in the world. The cardinality of P is then $|P| = N \cdot (N-1) + N + N$. If a total ordering on the blocks is given, it is possible to build an ordered vector *ctx* containing the truth values, 1 or 0, for each of the possible predicates. An action $(PUTON\ BLOCK_i\ BLOCK_j)$, with $BLOCK_i$ on top of $BLOCK_k$, changes the state producing as effects $(ON\ BLOCK_i\ BLOCK_j)$ and $(CLEAR\ BLOCK_k)$, and conversely negating the previously true predi-

cates ($ON\ BLOCK_i\ BLOCK_k$) and ($CLEAR\ BLOCK_k$). A visual sample of such move is provided in Fig. 1; in this example $N = 10$ and $P = 110$.

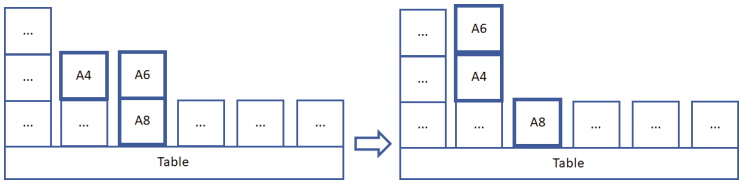

Fig. 1. A sample of pre and post state for the action (PUTON A6 A4)

In Fig. 2 the architecture and the workflow of the proposed action learning system is shown.

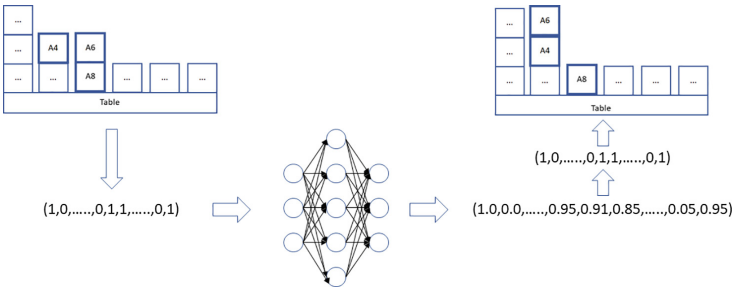

Fig. 2. The learning planning action model workflow

3 Experiments Design and Results

Experiments have been designed in order to vary the amount of noise introduced during the training and test phase; the same amount of noise is applied both to the training and to the test sets. The ground truth used to compute the accuracy of prediction is represented by the perfect no-noise states.

The data set samples used in the experiments are represented by triples $(s1, s2, a)$, where s_1/s_2 represents the perceived pre/post action a states, and have been generated by a particular technique of *random consistency*.

The NN implementation used in the experiments is based on the Keras Python Framework with Tensorflow background was used to build the neural network model.

3.1 Noise Model

The *noise model* we have used assumes that some sensors can be faulty or inaccurate, or the interpretation of sensors data into true/false predicates can be inaccurate. Whatever the reason, the noise will produce in the agent's perceived state model a wrong value for certain predicates, i.e. a certain bit in the bit-vector representing the state will be flipped from true/false or vice-versa. In order to tune and control the noise we used two parameters: p_{states} establishing the percentage of noisy states samples to be altered and p_{state} which determine the percentage of the predicates of each single noisy state sample which will be randomly modified. For instance, let $N = 200000$, $|P| = 110$, $p_{states} = 0.9$ and $p_{state} = 0.01$; ≈ 180000 sample states records will be altered, and for each sample state $110 * 0.01 =\approx 1$ bit will be flipped. The alteration is uniformly distributed among the sample states and inside each sample state, so that at the end of the alteration phase each predicate truth value will have been inverted approximately the same number of times. Since each sample defines a pair of states *pre/post* execution, any noise alteration induced in a predicate of the *pre* state is operated in the state after execution too.

3.2 Evaluation Metrics

Accuracy of the state predicted by the trained NN was chosen as the evaluation metric. The expected output vector and the predicted vector are compared bit-by-bit; only those which completely match are counted as correct.

3.3 The Random Consistent States Generator

In order to generate the dataset for training and testing, a *random consistent states* generator was developed for the *blocksworld* domain. The sample states are randomly generated as variations of the *blocksworld* using the three predicates previously described (i.e. *table*, *on* and *clear*) with the characteristic of being consistent with the usual constraints: single block stacking and blocks gravity (one block can be directly on one block only, a tower of blocks has the lower block on the table etc,); the post state is generated from the pre state just applying the semantic of the considered action a. The random consistent states generator is characterized by the following parameters:

- The *number of samples* N to be generated
- The *number of blocks* in the world, named sequentially using the "A*" style, e.g. A1, A2, A3...
- The *number of towers*, i.e. the number of available locations on the table; note that some locations could be empty after the generation.
- The three blocks subject of the move, e.g. "A1", "A2" and "A3", which represent the action (PUTON,A1,A3), with the pre-condition ON(A1,A2).
- The maximum height *maxheight* of a tower, i.e. the maximum number of blocks that can be stacked.
- The type of samples to be generated

The sample generator outputs N tuples of the form (preVector, postVector, move, maxHeight, sampleType), each representing an instance of the move in a different, but consistent, scenario. preVector and postVector are one-hot encoding vectors that associate each of the possible predicates in the domain with its truth state, i.e. 0 if the predicate is false, 1 otherwise; sampleType can be either *positive*, meaning that the move should be executable in the generated state, or *negative*, for states in which the action is not allowed. In the latter case, the post state is equal to the pre state, i.e. trying to execute the action when it is not possible leads to observing no changes in the environment.

3.4 The Data Sets

For the experiments, two data sets were produced with the generator described in Sect. 3.3; the sample state triple represent instances of the move (PUTON,A6,A4), with the precondition (ON A6,A8). Note that, for the move to be executable, in positive samples two other predicates must hold: (CLEAR A6) and (CLEAR A4). The first data set, POS, contains 200000 positive records; the second, POS/NEG, includes the records in POS as well as 200000 negative records, for a total of 400000 records equally balanced between the two types.

3.5 Test Series 1: Tuning the NN

The purpose of the first series of test was both to tune the neural network structure in terms of hidden layers, to assess the action model learning capabilities and to establish a performance upper bound of the model in absence of noise. The performance of the network have been tested on a clean data set, i.e. without noise. The training process was repeated for both the POS and POS/NEG data sets, in order to choose the best settings combination for the network.; for test series 1, as well as test series 2 and 3, performances were evaluated using a ten-fold cross-validation. Results are reported in table in Fig. 3. The best performances for both data sets has been obtained by including seven hidden layers and training for 100 epochs. In such configuration settings the system reaches nearly optimal performance with no overfitting; the settings thus determined were applied for the subsequent experiments using a noisy environment.

3.6 Test Series 2: Positive only Training Samples

In this test series, the network was fed with positive samples, i.e. showing the agent only perceived pre/post states of correct execution and the action under training. Random noise was applied in different percentages to the data sets; in the selected training samples, both the pre and post state were perturbed together, i.e. the same bits were flipped, whereas in the test records only the pre states were affected by noise, and the predicted states were compared against the perfect clean, not noisy post states.

Results for Test 2 are reported in table in Fig. 4. With only a quarter of the training samples altered, the performances of the model are stable, showing an

Dataset	Training Epochs	Hidden Layers	Avg Accuracy	Variance
POS	50	5	0.888195	0.001286
POS	100	5	0.880915	0.000827
POS	50	7	0.939730	0.000434
POS	**100**	**7**	**0.963085**	**0.000243**
POS	150	10	0.893290	0.000956
POS	200	10	0.901670	0.001315
POS/NEG	50	5	0.855028	0.001616
POS/NEG	100	5	0.888728	0.000654
POS/NEG	50	7	0.843653	0.000957
POS/NEG	**100**	**7**	**0.903338**	**0.000352**
POS/NEG	150	10	0.737005	0.004638
POS/NEG	200	10	0.763370	0.002550

Fig. 3. Results of test 1. Values are averaged over 10 folds.

% of Noisy Records	% Noise per record	Avg Accuracy	Variance
25	1	0.814800	0.000413
25	5	0.752745	0.000136
25	10	0.730915	0.002585
25	20	0.745075	0.000008
50	1	0.686275	0.000116
50	5	0.560100	0.000252
50	10	0.491115	0.000185
50	20	0.488590	0.000053
75	1	0.579250	0.000112
75	5	0.377300	0.000140
75	10	0.247105	0.000045
75	20	0.231825	0.000018

Fig. 4. Results of test 2. Values are averaged over 10 folds.

appreciable ability of the system to resist to the increase in noise. However, with higher percentages, such resistance is much lower.

3.7 Test Series 3: Positive and Negative Training Samples

In the third test series, the neural network has been fed with the POS/NEG training state samples. As mentioned before negative examples correspond to cases where, notwithstanding the action is not executable, i.e. at least one action precondition is not verified, the tentative of executing the action does not produce appreciable changes in the post state, i.e. the action is not executed. In the table in Fig. 5 the results of the test are reported; figures are on average lower or on par with the corresponding network in test 2. In the table in Fig. 6 the number of correctly and wrongly predicted samples in test 3 are reported, for both positive and negative samples and at various levels of noise; figures are averaged over 10 folds. It can be observed that, regardless of the amount of noise applied to the data, positive samples are easier to predict, i.e. more positive

samples are correctly predicted than negative ones. In Fig. 8 the percentage of correctly predicted records in test 3 for the two types of records, positive and negative, is shown, grouped by the noise percentages. It can be seen that the gap between the two types, initially negligible, increases with the amount of noise. This could mean that a higher percentage of noise in the negative samples effectively hides that there should be no changes between the pre and post state. On the contrary, the network is able to better understand and learn the fixed changes in truth values of the predicates that occur in the positive samples. A further comparison between the results of test 2 and 3 in Fig. 7 shows that the accuracy score is similar for the networks trained and tested on the two different data sets; however, when the amount of noise per record becomes grows to 20%, the network predicting positive samples only delivers better performances. This is consistent with the results in Fig. 8.

% of Noisy Records	% Noise per record	Avg Accuracy	Variance
25	1	0.785740	0.000379
25	5	0.724420	0.000610
25	10	0.727368	0.000141
25	20	0.701115	0.000084
50	1	0.685510	0.000408
50	5	0.590473	0.000221
50	10	0.499955	0.000172
50	20	0.433398	0.000172
75	1	0.585430	0.004902
75	5	0.462773	0.000212
75	10	0.261468	0.000405
75	20	0.144035	0.000118

Fig. 5. Results of test 3. Values are averaged over 10 folds.

% of altered records	% of alt. bits	Correct Positive	Correct Negative	Wrong Positive	Wrong Negative
25	1	16331	15098	3669	4902
25	5	14924	14053	5077	5947
25	10	14945	14150	5055	5850
25	20	14826	13219	5174	6782
50	1	14362	13058	5638	6942
50	5	12405	11214	7595	8786
50	10	10624	9375	9376	10626
50	20	9704	7632	10296	12368
75	1	12513	10904	7487	9096
75	5	9887	8624	10113	11376
75	10	6027	4432	13973	15568
75	20	4598	1164	15402	18836

Fig. 6. Number of correctly and wrongly predicted samples

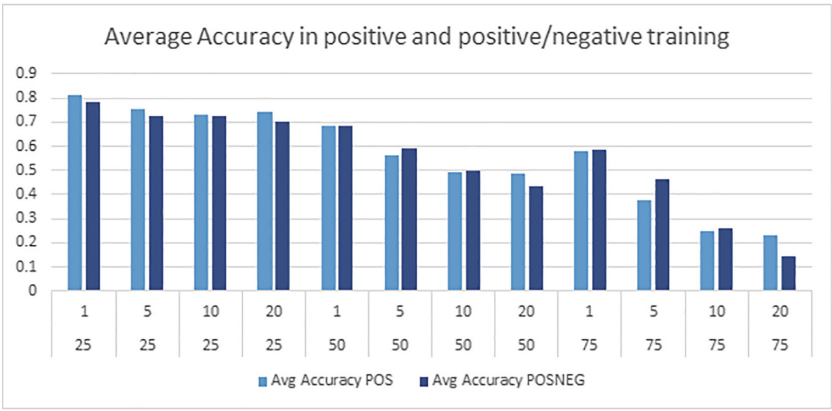

Fig. 7. Comparison of performance with positive only and positive and negative samples

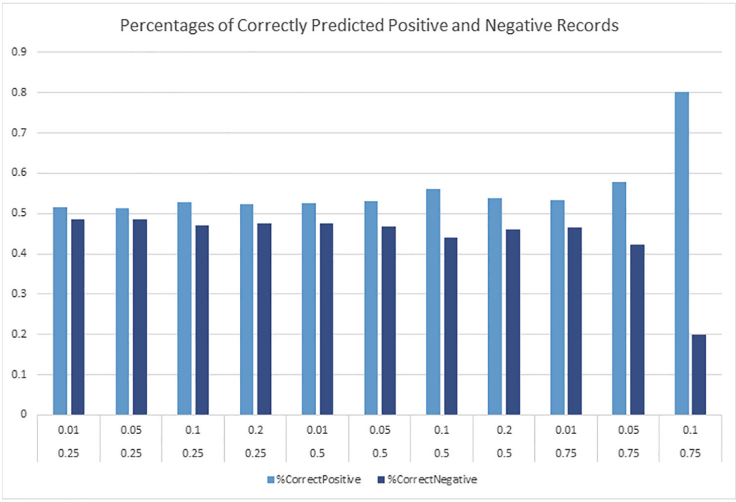

Fig. 8. Percentages of correctly predicted positive and negative records

4 Conclusions

We have introduced an original neural network-based architecture to learn planning action models by observations. Experiments show that the accuracy obtained is quite promising and the proposed model is able to be resilient with respect to noise, which is indeed a quite realistic hypothesis for learning agents situated in the real world. To the best of our knowledge this is the first planning action learning model based on NN, which is able to cope with noise affected perceptions. Moreover another innovative contribution of this work is the introduction of a new concept of action representation, where the resulting learned

action is not described in terms of preconditions effects, like in classical planning [6], but is represented by the trained NN, which able to forecast the state after the execution of an action, i.e. to compute the state successor function. An interesting future development will be to investigate the consequences of exploring the search space, e.g. to do forward search, by using trained NN for each available actions instead of classical pre/post conditions representation.

References

1. Ai-Chang, M., et al.: MAPGEN: mixed-initiative planning and scheduling for the mars exploration rover mission. IEEE Intell. Syst. **19**(1), 8–12 (2004). https://doi.org/10.1109/MIS.2004.1265878
2. Arora, A., Fiorino, H., Pellier, D., Métivier, M., Pesty, S.: A review of learning planning action models. Knowl. Eng. Rev. **33**, e20 (2018). https://doi.org/10.1017/S0269888918000188
3. Balac, N., Gaines, D.M., Fisher, D.: Learning action models for navigation in noisy environments (2000)
4. Chiancone, A., Franzoni, V., Niyogi, R., Milani, A.: Improving link ranking quality by quasi-common neighbourhood, pp. 21–26. IEEE Press (2015). https://doi.org/10.1109/ICCSA.2015.19
5. Cresswell, S., Gregory, P.: Generalised domain model acquisition from action traces. In: ICAPS 2011 - Proceedings of the 21st International Conference on Automated Planning and Scheduling (2011)
6. Fikes, R.E., Nilsson, N.J.: STRIPS: a new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3–4), 189–208 (1971). https://doi.org/10.1016/0004-3702(71)90010-5
7. Franzoni, V., Leung, C.H.C., Li, Y., Mengoni, P., Milani, A.: Set similarity measures for images based on collective knowledge. In: Gervasi, O., et al. (eds.) ICCSA 2015. LNCS, vol. 9155, pp. 408–417. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21404-7_30
8. Franzoni, V., Li, Y., Mengoni, P.: A path-based model for emotion abstraction on Facebook using sentiment analysis and taxonomy knowledge, pp. 947–952. IEEE Press (2017)
9. Franzoni, V., Mencacci, M., Mengoni, P., Milani, A.: Heuristics for semantic path search in Wikipedia. In: Murgante, B., et al. (eds.) ICCSA 2014. LNCS, vol. 8584, pp. 327–340. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09153-2_25
10. Franzoni, V., Milani, A.: PMING distance: a collaborative semantic proximity measure, vol. 2, pp. 442–449. IEEE Press (2012). https://doi.org/10.1109/WI-IAT.2012.226
11. Franzoni, V., Milani, A., Pallottelli, S., Leung, C., Li, Y.: Context-based image semantic similarity, pp. 1280–1284. IEEE Press (2016). https://doi.org/10.1109/FSKD.2015.7382127
12. Hoffmann, J.: FF: the fast-forward planning system. AI Mag. **22**, 57 (2001)
13. Jiménez, S., De La Rosa, T., Fernández, S., Fernández, F., Borrajo, D.: A review of machine learning for automated planning. Knowl. Eng. Rev. **27**(4), 433–467 (2012). https://doi.org/10.1017/S026988891200001X

14. Leung, C.H.C., Li, Y., Milani, A., Franzoni, V.: Collective evolutionary concept distance based query expansion for effective web document retrieval. In: Murgante, B., et al. (eds.) ICCSA 2013. LNCS, vol. 7974, pp. 657–672. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39649-6_47
15. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529 (2015). https://doi.org/10.1038/nature14236
16. Molineaux, M., Aha, D.W.: Learning unknown event models. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2014, pp. 395–401. AAAI Press (2014)
17. Mourao, K.: Learning probabilistic planning operators from noisy observations. In: 31st Workshop of the UK Planning & Scheduling Special Interest Group (PlanSIG 2013) (2014)
18. Mourao, K., Zettlemoyer, L.S., Petrick, R.P.A., Steedman, M.: Learning STRIPS operators from noisy and incomplete observations, October 2012
19. Newton, M.A.H., Levine, J.: Implicit learning of compiled macro-actions for planning. Front. Artif. Intell. Appl. **215**, 323–328 (2010)
20. Safaei, J., Ghassem-Sani, G.: Incremental learning of planning operators in stochastic domains. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 644–655. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-69507-3_56
21. Shah, M., et al.: Knowledge engineering tools in planning: state-of-the-art and future challenges. Knowl. Eng. Plann. Sched. **53**, 53 (2013)
22. Shen, W.M.: Discovery as autonomous learning from the environment. Mach. Learn. **12**(1), 143–165 (1993). https://doi.org/10.1023/A:1022827618816
23. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484 (2016)
24. Wang, X.: Learning by observation and practice: an incremental approach for planning operator acquisition. In: Machine Learning Proceedings 1995, January, pp. 549–557 (1995). https://doi.org/10.1016/B978-1-55860-377-6.50074-8
25. Weber, B.G., Mateas, M., Jhala, A.: Learning from demonstration for goal-driven autonomy. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2012, pp. 1176–1182. AAAI Press (2012)
26. Winograd, T.: Understanding Natural Language. Academic Press Inc., Orlando (1972)
27. Zhuo, H.H., Kambhampati, S.: Action-model acquisition from noisy plan traces. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI 2013, pp. 2444–2450. AAAI Press (2013)