A Learning Based Approach for Planning with Safe Actions

Rajdeep Niyogi[1](✉), Saurabh Sharma[1], Michal Vavrecka[2], and Alfredo Milani[3]

[1] Indian Institute of Technology Roorkee, Roorkee 247667, India
rajdpfec@iitr.ac.in, saurabhsharma090394@gmail.com
[2] Czech Technical University, 166 36 Prague 6, Czechia
vavrecka@fel.cvut.cz
[3] University of Perugia, 06123 Perugia, Italy
milani@unipg.it

Abstract. Given a configuration involving some objects in an environment, the planning problem, considered in this paper, is to find a plan that rearranges these objects so as to place a new object. The challenging aspect here involves deciding when an object can be placed on top of another object. Here only defining standard planning operators would not suffice. For instance, using these operators we can come up with actions that may be performed at a state but it should not be performed. So we introduce the notion of *safe actions* whose outcomes are consistent with the laws of physics, commonsense, and common practice. A safe action can be performed if a robot performing the action knows the knowledge of the situation. We developed a knowledge engine using a supervised learning technique. However, unlike the common task of learning functions, our approach is to *learn predicates*–that evaluate to binary values. By learning such a predicate a robot would be able to decide whether or not an object A can be placed on top of another object B. We give a method to handle new objects for which the predicates have not been learned. We suggest a nondeterministic planning algorithm to synthesize plans that contain only safe actions. Experimental results show the efficacy of our approach.

Keywords: Planning · Learning · Predicate · Safe actions

1 Introduction and Motivation

Service robots are used to manipulate objects to achieve goal directed activities in automated/smart environments (e.g., home, cafeteria, office) [1–4]. In order to accomplish goals, a robot should possess the following skills: (i) perceptual skills which refer to the ability to acquire information from the environment, (ii) motor skills which refer to the ability to actuate in the environment, and (iii) cognitive skills which refer to the ability to synthesize plans to achieve a goal [5]. Planning is the reasoning side of acting [6]. In order to perform a simple action like pick-and-place, a robot should apply commonsense, physics knowledge, and take into account the current situation or context [7]. For certain tasks, in order for planning to be successful, the agent should learn some relationships of the objects.

Consider a table, in a kitchen, that is divided into four locations where each location is occupied with a plate; assume for simplicity that all the plates are identical and only one object can occupy each location. There is no other free space available. If a human is asked to place a cup on the table, she would possibly do any one of the following: (i) place the cup on any plate, (ii) pick any plate and place it on top of an adjacent plate, or make a tower of all the plates and then place the cup in a vacant location. There are different perspectives for looking at this task. These are, for instance, grasping which is concerned with how to hold the cup (by the handle of the cup or by one side); placing which refers to how the hand should move between the objects to place the cup, and if both the cup and plate are made of glass then care should be taken to place the cup softly on the plate; and capability of picking the cup (a weak hand may not be able to pick the cup properly). In this paper we take a different perspective which is not about these physical abilities required for placing an object; rather it is about reasoning when an object can be placed on top of another object. Humans have acquired this cognitive skill from their experience with handling objects of different types, observation, and/or learning from situations.

When a human asks the home robot to place the cup on the table, she expects the robot not just to place it somewhere on the table, but to first tidy up the table and then keep the cup. The goal states would be to place the cup on top of either plate A, plate B, plate C, or plate D. Such goal states correspond to the above mentioned point (i). Let us reconsider the kitchen situation described above but now all the locations are occupied by a cup and the task is to place another cup on the table. In this case none of the goal states (place the cup on top of either cup A, cup B, cup C, or cup D) is feasible since it is not a common practice of keeping a cup on top of another cup.

If a robot can place a cup on a plate, it can as well place a cup on another cup; nothing precludes this from happening. A robot is expected to perform safe actions. Informally, a safe action does not violate the laws of physics, commonsense, or common practice. For instance, the action of placing a cup on top of another cup is not a safe action, whereas placing a cup on a plate is a safe action. Thus if we wish to make the robots act like humans, it becomes imperative that they are embedded with adequate knowledge base that enables them to act autonomously and not just merely perform actions. This requires the robot to know (by learning) the commonsense knowledge of the situation (e.g., Do not put a cup on a cup or Clean the table before placing cups) and also physical knowledge (e.g., If you place a laptop on a cup then it will fall). Anyone who is used to a kitchen environment would first tidy up the table and then place the cup. This is the intent of the request and this corresponds to the goal state discussed above for point (ii). Here again if we replace the plates with cups, since a cup is normally not placed on top of another cup, a vacant location cannot be created and thus the request cannot be fulfilled.

For the planning scenarios described above, a precondition may hold, but an action should not be performed. We emphasize here that it is not the case that an action cannot be performed; rather it is the case where an action should not be performed. Here, only an operator definition will not suffice. So we introduce the notion of safe actions whose outcomes are consistent with physics, commonsense, and common practice. A safe action can be performed if a robot performing the action knows the knowledge of the situation. We developed a knowledge engine using a supervised learning technique. However, unlike the common task of learning functions, our approach is to learn predicates–that evaluate to binary values. By learning such a predicate a robot would be able to decide whether or not an object A can be placed on top of another object B. Once the robot learns the predicate, planning can be carried out and the plans contain only safe actions.

The remaining part of the paper is organized as follows. The preliminaries are given in Sect. 2. Predicate learning is given in Sect. 3. The proposed planning algorithm is given in Sect. 4 and the experimental results are given in Sect. 5. Related work is given in Sect. 6. The conclusions are given in Sect. 7.

2 Preliminaries

Definition 1 *(Planning domain). A planning domain can be formally defined as $D = \langle S, A, s_0, \delta, F\rangle$ where S is a nonempty finite set of states $S \subseteq 2^{AP}$ where AP is a nonempty finite set of atomic propositions or ground predicates for a domain, A is a nonempty finite set of executable actions, s_0 is the initial state, δ is a transition function $\delta : S \times A \rightarrow S$, F is a set of final states $F \subseteq S$.*

Definition 2 *(Planning problem). Given a planning domain D, a planning problem is to find a plan (a sequence of actions) which when executed at an initial state would lead to a final state.*

2.1 Planning Operator

A planning operator O is defined using preconditions $pre(O)$ and postconditions $post(O)$. The semantics of a planning operator O is given as follows: O is applicable at a state s if $pre(O)$ holds at s, and $post(O)$ holds in the resulting state s'. Thus a planning operator transforms one state to another. An action is a ground instance of an operator.

In the standard Blocks-world domain, the blocks are assumed to be identical in shape and size. An operator to put an object x on top of another object y (say, by a robot hand) can be defined as follows where $clear(y)$ means that nothing is on top of y and $handempty$ means that the robot is not holding any object.

put-on-top(x, y)
$precondition : holding(x) \wedge clear(y) \wedge x \neq y$
$postcondition : on(x, y) \wedge handempty \wedge not(clear(y))$

Consider a simple situation in this domain where the blocks B are C are placed on a table, and a robot is holding block A. By performing put-on-top(A,B), A is placed on top of B; similarly by performing put-on-top(A,C), A is placed on top of C. In this domain there is no additional constraint for placing one block on top of another block, and so any block can be placed on top of another subject to satisfying the preconditions. Let us consider an instance of the operator say put-on-top($laptop, cup$). If the precondition holds in a state, this action can be performed at the state and in the resulting state the *laptop* would be on top of the *cup*. Although an action can be performed at a state if the preconditions hold in the state, it should not be performed at the state in this case; the resulting configuration is unstable and the *laptop* may fall. Thus when arbitrary objects are considered, the above operator definition will not be suitable since an agent performing the action should know or decide whether it would be appropriate to place one object on top of another. In the following we suggest variants of the put-on-top operator to see if such reasoning can be captured at the syntactic level.

2.2 Limitations of Planning Operators

Situation I: An operator that would allow placing an object on top of another object when both the objects are of the same type.

put-on-top-same-type(x, y)
$precondition : (type(x) = type(y)) \wedge holding(x) \wedge clear(y) \wedge x \neq y$
$postcondition : on(x, y) \wedge handempty \wedge not(clear(y))$

For example, when both the objects are, say, plate or book, the operator would be applicable at a state where the precondition holds. However, when both the objects are cup then this operator should not be performed.

Situation II: An operator that would allow placing an object on top of another object when both the objects are of different type.

put-on-top-diff-type(x, y)
$precondition : (type(x) \neq type(y)) \wedge holding(x) \wedge clear(y) \wedge x \neq y$
$postcondition : on(x, y) \wedge handempty \wedge not(clear(y))$

This operator should not be performed when $x = laptop$ and $y = cup$.

Thus, operator definition alone is not sufficient for the problem under consideration.

Safe Action: An action a, applicable at state s ($pre(a)$ holds at s), is considered safe if $post(a)$ is consistent with physics, commonsense, and common practice.

Safe Plan: A safe plan is a plan that consists of only safe actions.

The problems with the operators defined in Situations I and II are that they are not safe. Thus in order to define an operator for placing one object on top of another object we first define a predicate $CAN\_PLACE(A, B)$ which is true if A can be placed on B, false otherwise.

put-on-top-safe(x, y)

$precondition : CAN\_PLACE(x, y) \wedge holding(x) \wedge clear(y) \wedge x \neq y$
$postcondition : on(x, y) \wedge handempty \wedge not(clear(y))$

Thus in order for a robot to perform safe actions, it should know or learn the truth/falsity of the CAN_PLACE predicate. We suggest an approach for learning this predicate in the next section.

3 Predicate Learning

The predicate $CAN\_PLACE(A, B)$ is true if A can be placed on B, false otherwise. An important property of the predicate is that it is not symmetric in general, $CAN\_PLACE(A, B)$ does not imply $CAN\_PLACE(A, B)$. From the truth/falsity of the $CAN\_PLACE(A, B)$ predicate, no conclusion be made about the predicate $CAN\_PLACE(A, B)$ since these two predicates are not logically connected at all, in general.

The CAN_PLACE predicate instances are to be learned for all the pairs of objects. For instance, $CAN\_PLACE(cup, cup)$, $CAN\_PLACE(cup, plate)$, $CAN\_PLACE(cup, stapler)$, etc. Thus, if there are n objects, the total number of instances of the predicate would be n^2 and this many predicates should be learned.

The robot has to decided if an object A can be placed on top of another object B based solely on 2D images of the objects. An image in RGB format is first converted to gray scale. This is done since two images of an object with different colors would be classified into its unique category (denoted by an integer identifier ID). An existing object recognition techniques is used for the purpose. We have taken 10 categories of objects and for each category around 60 images are taken from Caltech vision, Bing image search, and Google image search.

A neural network (NN) is used for learning the predicate. The network structure will consist of an initial layer of size 2 (corresponding to a pair of objects), followed by a variable number of hidden layers, and a final output layer of size 1. The training of the NN is done by taking input pairs (e.g., ID_A, ID_B) and providing the output which is the associated truth value of the predicate instance $CAN\_PLACE(A, B)$. The predicates learned for a set of objects are shown in the Table 1, where A is an object from a column, B is an object from a row in $CAN\_PLACE(A, B)$. Any object in the set can be placed on a 'book'. A 'cup' can also be placed on a 'plate', so the cell CAN_PLACE(cup,plate) is 1.

3.1 Handling a New Object

Once a robot has learned the CAN_PLACE predicate it can decide if an object A can be placed on another object B. If a robot is faced with new object(s) for which the predicates have not been learned, the robot would get stuck and the goals cannot be accomplished. At some point of time a robot would always encounter a new object. So it is not possible to anticipate all such objects in advance and learn the corresponding predicates. Thus we suggest a method for mapping a new object to some known object (for which a predicate has already

Table 1. Learned predicate instances.

$CAN\_PLACE(A, B)$	Cup	Plate	Stapler	Laptop	Book
Cup	0	0	0	0	0
Plate	1	1	0	0	0
Stapler	0	0	0	0	0
Laptop	0	0	1	0	1
Book	1	1	1	1	1

been learned) based on some physical properties (Table 2) of the object which are easily observable. These properties are:

1. Overall shape: The overall shape of an object can be either simple or complex. Examples of objects with simple shapes are book, plate. Examples of objects with complex shapes are stapler, cup. There are some objects (e.g., laptop) that can be both simple and complex depending on its state. The numbers 1, 2 are used to represent complex shape and simple shape respectively.
2. Base shape: It can be either simple or complex. Simple shapes are circle (characterized by the top view of a cup), rectangle (characterized by book, laptop). Stapler is an example of a complex shape. The numbers 1, 2, 3 are used to represent circle, rectangle, and complex shape respectively.
3. Base area: It is calculated in sq. cm.

Table 2. Sample data set of objects of different category.

Overall shape	Base shape	Base area	Category
2	2	153	Book
1	2	600	Laptop
1	3	500	Stapler

Whenever a robot picks an object, it can easily identify these properties of the object. Supervised learning (SVM) is used to learn these properties corresponding to given categories. The testing data set consists of 20 vectors for each category. The results of learning the properties corresponding to each category are shown in Table 3. For example, out of the 20 vectors for 'laptop', 15 are correctly classified as 'laptop' and 5 are incorrectly classified as 'book'.

When a new object (other than these five categories) is encountered, these three properties of the object are identified. Based on these properties, the new object is mapped to an object in the closest category for which the predicate is known, i.e., book, laptop, cup, plate, and stapler. Now the new object is treated as the object in the category. If the new object is a "newspaper" the properties are identified as:

Table 3. Learning the physical properties of each category.

True class\predicted class	Cup	Plate	Stapler	Laptop	Book
Cup	20	0	0	0	0
Plate	0	19	1	0	0
Stapler	0	1	19	0	0
Laptop	0	0	0	15	5
Book	0	0	0	5	15

1. Overall Shape is simple, so the value is 2.
2. Base Shape is rectangle, so the value is 2.
3. Base area of a newspaper can be calculated by measuring its length and width. For a normal newspaper which is half folded, its length is 42 cm and width is 33 cm. So the area is 42 × 33 i.e., 1386 sq.cm.

The newspaper with properties represented as a vector [2, 2, 1386] is mapped to a laptop. So for deciding if a newspaper can be placed on a cup reduces to deciding if a laptop can be placed on top of a cup for which the predicate $CAN\_PLACE(laptop, cup)$ is already known.

4 Proposed Planning Approach

Definition 3 *(Planning problem with safe actions)***.** *Given a planning domain D, a configuration involving some objects in an environment, the planning problem is to find a safe plan that rearranges these objects so as to place a new object.*

We assume that all objects can be placed on the table: $\forall x. CAN\_PLACE(x, table)$ holds. Let $TOWER_y$ denote a tower of objects with y at the bottom of the tower, and putdown($TOWER_y$) be an action that removes all the objects on top of y including y.

```
Planning algorithm:
Plan-to-place-object
Input: an object $x$ to be placed in an Environment $E$
Output: $PLAN$-that consists of safe actions or failure
begin
while there exists an object $x$ to be placed in $E$ do
$PLAN := \emptyset$
    Case 1: Place $x$ in $E$ without rearrangement.
    if there exists an object $y$ in $E$ such that
        $(CAN\_PLACE(x, y) \wedge clear(y))$ hold
        then report success with $PLAN$ := put-on-top-safe$(x, y)$
    else report failure
    Case 2: Place $x$ in $E$ by rearranging only top objects.
```

```
        if there exists objects $y', y, z$ in $E$ such that
        $(CAN\_PLACE(y, z) \wedge clear(z) \wedge on(y, y') \wedge CAN\_PLACE(x, y'))$ hold
            then report success with
                $PLAN$ := put-on-top-safe($y, z$);put-on-top-safe($x, y'$)
        else report failure
        Case 3: Place $x$ within a tower of objects in $E$.
        if there exists objects $y, y'$ in the tower such that
        $(on(y, y') \wedge CAN\_PLACE(x, y') \wedge CAN\_PLACE(y, x))$ hold
            then report success with
                $PLAN$ := putdown($TOWER_y$);put-on-top-safe($x, y'$);
                          put-on-top-safe($TOWER_y, x$)
        else report failure
    od
    end
```

Features of the Algorithm: The algorithm is nondeterministic since for a given initial configuration different plans are possible for placing a particular object. For instance, if every region has only one plate and the goal is to place a cup, then both Case 1 and Case 2 holds. In this scenario any one of the cases would be considered nondeterministically and the corresponding plan would be returned. Similarly, if every region has a tower of $k > 1$ plates, and the goal is to place another plate, then all the three cases hold. Thus for the design of the algorithm, the nondeterminism is necessary (and not based on choice) since we only want to achieve the goal and moreover no goal configuration is given as input. This aspect differentiates the planning problem considered in this paper with conventional planning problems where an initial and goal configuration is given as input. Finally, the order of execution of the Cases is irrelevant.

4.1 Illustration of the Algorithm

We illustrate the working of the algorithm by taking different situations. The environment E consists of n-regions. In the following examples there are 4 regions that are named R1 (top left), R2 (top right), R3 (bottom right), R4 (bottom left).

1. Goal: place a stapler. There is no space available in any region and the stapler cannot be placed on top of any object. This is an instance of Case 2. So the steps involved are: first, remove the plate from R1 and place it on top of the plate in R2; now place the stapler in R1 (Fig. 1).
2. Goal: place a plate. There is no space available in any region and the stapler cannot be placed on top of any object. This is an instance of Case 3. So the steps involved are: first, remove the cup from R4; then place the plate on top of the book in R4; now place the cup on top of plate in R4 (Fig. 2).
3. Goal: place a cup. There is no space available in any region and a cup cannot be placed on top of a cup. Neither Case 2 nor Case 3 is applicable. So the object cannot be placed in any region (Fig. 3).

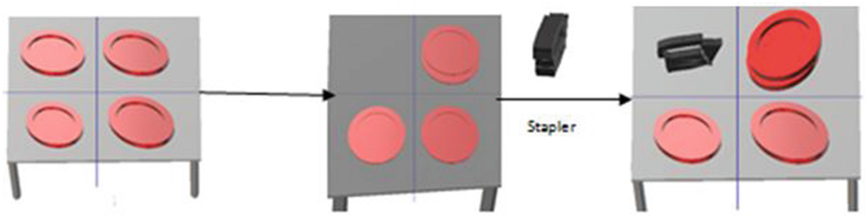

Fig. 1. Planning using Case 2.

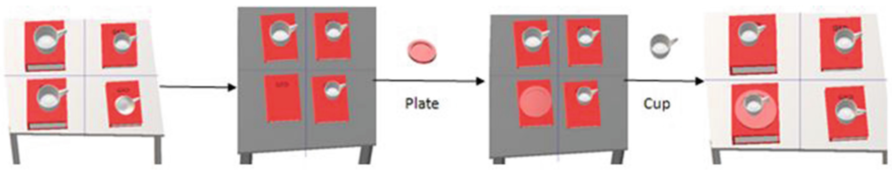

Fig. 2. Planning using Case 3.

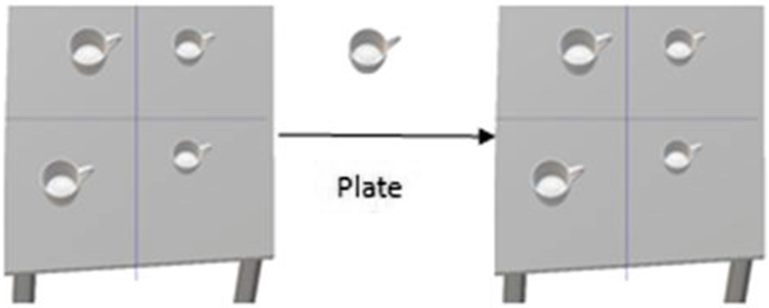

Fig. 3. No rearrangement is possible.

4.2 Placing a New Object for Which the Predicate has Not Been Learned

1. Goal: place a cellphone. Since no predicate is learned for cellphone, so overall shape, base shape, and base area are identified for cellphone and the final vector is [2, 2, 150] which is closest to a book. Thus the problem reduces to placing a book. This is an instance of Case 2. The steps involved are: first, remove plate on top of R1; then place it on top of plate in R4; now cellphone is placed in R1 (Fig. 4).
2. Goal: place a newspaper. Since no predicate is learned for newspaper, so, overall shape, base shape, base area are identified for newspaper and the final vector is [2, 2, 550], which is closest to a book. Thus the problem reduces to placing a book. This is an instance of Case 3. The steps involved are: first, remove cup from R1; then place newspaper on top of book already present in R1; now place cup on top of R1 (Fig. 5).

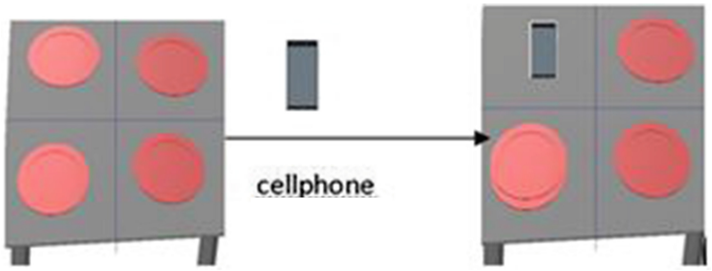

Fig. 4. Placing a new object: cellphone.

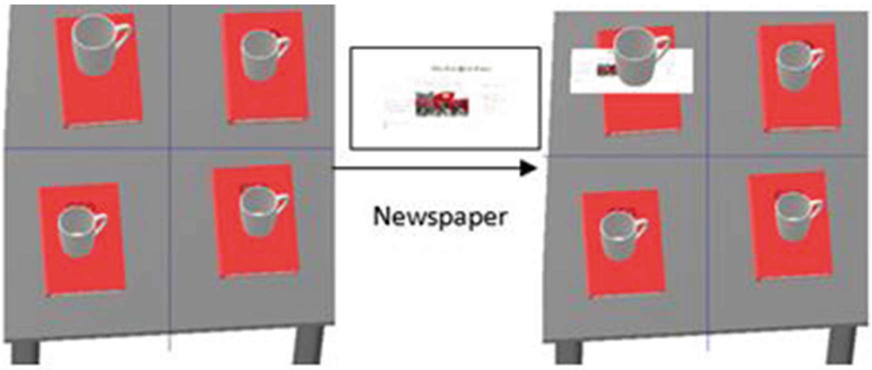

Fig. 5. Placing a new object: newspaper.

5 Experimental Results

All the experiments are performed using MATLAB on a personal computer having the configuration–Intel core i5 @ 2.5 GHz with 4 GB RAM.

I) We considered different initial configurations and for each configuration, different objects were selected for placing by fixing the number of regions. The average planning time taken for different instances is shown in Fig. 6. The time taken increases almost linearly with the number of objects.

II) We took different initial configurations and for each configuration, the number of regions was varied for a fixed number of objects to be placed in the configuration. The average planning time taken for different instances is shown in Fig. 7a and 7b. The time taken decreases almost logarithmically with the number of regions.

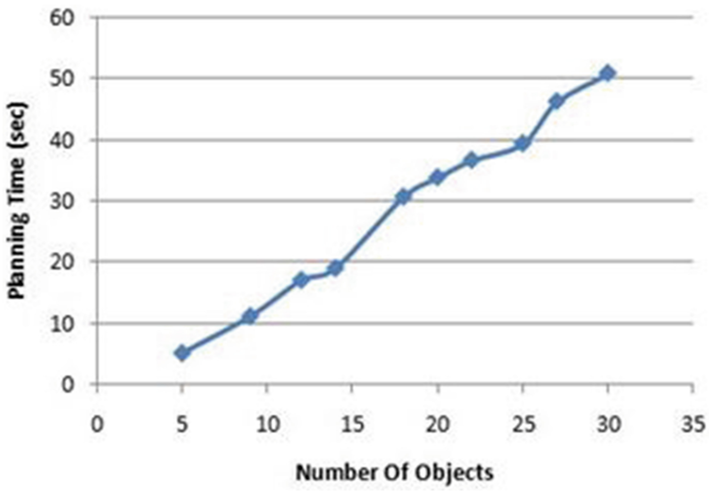

Fig. 6. Planning time versus number of objects.

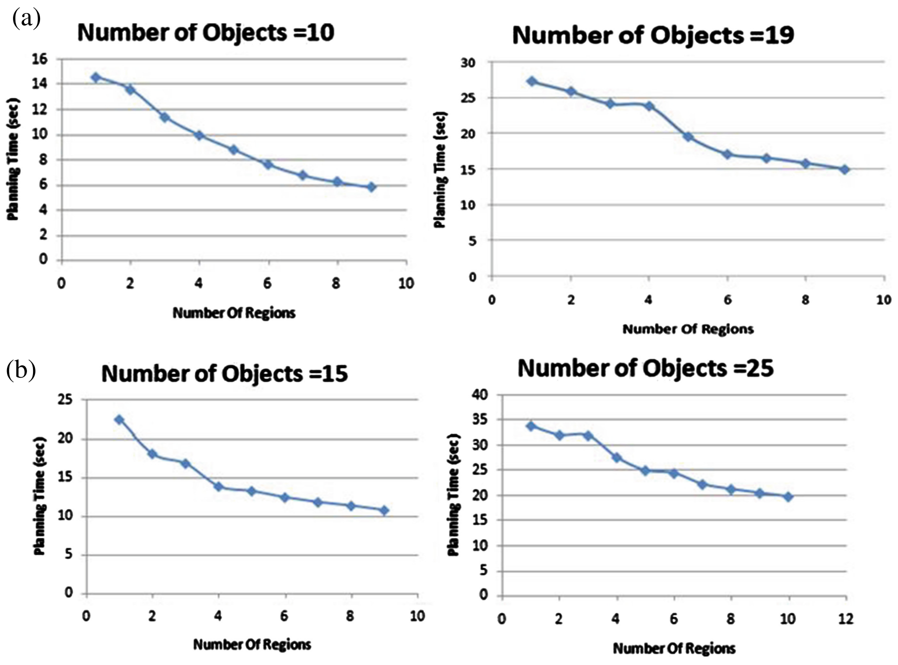

Fig. 7. (a) Planning time versus number of regions for #objects 10, 19. (b) Planning time versus number of regions for #objects 15, 25.

6 Related Work

The problem of placing an object properly has been considered in [8,9]. How to place an object depends on the shape of the object and the placing environment. For example, a plate has to be placed vertically in the slots of a dish rack but horizontally on a table. A supervised learning approach is suggested for finding good placements given point-clouds of the object and the placing area. The methods [8,9] combine the features that capture support, stability and preferred configurations, and use a shared sparsity structure in the parameters. A rearrangement planning problem is considered in [10] where a robot is expected to work in a clutter by interacting with multiple objects to achieve its goal. For this the robot may either try to avoid other objects and find a path to reach the desired object (goal) or it may remove some objects on its path to reach the goal. In [11] the problem of placing an object on cluttered table surfaces is considered. If enough space is not available for placing an object for directly placing an object, the planning algorithm suggested in [11] tries to find a sequence of linear push actions so as to obtain the necessary space. However, none of these works [8–11] consider the reasoning required for placing one object on top of another object.

7 Conclusions

In this paper we considered a goal directed object manipulation task that requires cognitive skills like planning and learning. We showed that by learning a predicate $CAN\_PLACE(A, B)$, using a simple neural network, a robot can determine whether to place an object A on top of another object B, which in turn allows the robot to perform safe actions. We have suggested a method by which a robot can handle a new object for which the predicate has not been learned. We proposed a nondeterministic planning algorithm to place an object. We have implemented our approach and the experimental results are quite promising. Our ongoing work aims to develop a robotic system that can function in a real world scenario using the concepts developed in this paper.

Acknowledgements. The authors thank the anonymous reviewers of ICCSA 2020 for their valuable suggestions. The first author was in part supported by a research grant from Google.

References

1. Wilson, G., et al.: Robot-enabled support of daily activities in smart home environments. Cogn. Syst. Res. **54**, 258–272 (2019)
2. Abdulkareem, A., Ogunlesi, V., Afolalu, S.A., Onyeakagbu, A.: Development of a smart autonomous mobile robot for a cafeteria management. Int. J. Mech. Eng. Technol. **10**(1), 1672–1685 (2019)

3. Ersen, M., Oztop, E., Sariel, S.: Cognition-enabled robot manipulation in human environments: requirements, recent work, and open problems. IEEE Robot. Autom. Mag. **24**(3), 108–122 (2017)
4. Kemp, C.C., Edsinger, A., Torres-Jara, E.: Challenges for robot manipulation in human environments [grand challenges of robotics]. IEEE Robot. Autom. Mag. **14**(1), 20–29 (2007)
5. Huamán Quispe, A., Ben Amor, H., Christensen, H.I.: A taxonomy of benchmark tasks for robot manipulation. In: Bicchi, A., Burgard, W. (eds.) Robotics Research. SPAR, vol. 2, pp. 405–421. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-51532-8_25
6. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Elsevier, Amsterdam (2004)
7. Beetz, M., Besler, D., Haidu, A., Pomarlan, M., Bozcouglu, K.A., Bartels, G.: KnowRob 2.0: a 2nd generation knowledge processing framework for cognition-enabled robotic agents. In: ICRA, pp. 512–519 (2018)
8. Jiang, Y., Zheng, C., Lim, M., Saxena, A.: Learning to place new objects. In: ICRA, pp. 3088–3095 (2012)
9. Jiang, Y., Lim, M., Zheng, C., Saxena, A.: Learning to place new objects in a scene. Int. J. Robot. Res. **31**(9), 1021–1043 (2012)
10. King, J.E., Cognetti, M., Srinivasa, S.S.: Rearrangement planning using object-centric and robot-centric action spaces. In: ICRA, pp. 3940–3947 (2016)
11. Cosgun, A., Hermans, T., Emeli, V., Stilman, M.: Push planning for object placement on cluttered table surfaces. In: IROS, pp. 4627–4632 (2011)