Constraint-based Case-Based Planning Using Weighted MAX-SAT

Hankui Zhuo[1], Qiang Yang[2], and Lei Li[1]

[1] Software Research Institute, Sun Yat-sen University, Guangzhou, China.
zhuohank@gmail.com, lnslilei@mail.sysu.edu.cn
[2] Computer Science and Engineering, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong. qyang@cse.ust.hk

Abstract. Previous approaches to case-based planning often finds a similar plan case to a new planning problem to adapt to solve the new problem. However, in the case base, there may be some other cases that provide helpful knowledge in building the new solution plan. Likewise, from each existing case there may be only certain parts that can be adapted for solving the new problem. In this paper, we propose a novel constraint-based case-based planning framework that can consider all similar plans in a case base to the current problem, and take only portions of their solutions in adaptation. Our solution is to convert all similar plan cases to constraints, and use them to solve the current problem by maximally exploiting the reusable knowledge from all the similar plan cases using a weighted MAX-SAT solver. We first encode a new planning problem as a satisfiability problem, and then extract constraints from plan cases. After that, we solve the SAT problem, including the extracted constraints, using a weighted MAX-SAT solver and convert the solution to a plan to solve the new planning problem. In our experiments, we test our algorithm in three different domains from International Planning Competition (IPC) to demonstrate the efficiency and effectiveness of our approach.

1 Introduction

Automatic planning aims to find an action sequence that transforms an initial state to a goal state. Researchers have built different algorithms to solve the problem efficiently. Classical planning involves the generation of plans by state or partial plan space search in order to satisfy a given goal [16, 17]. Because planning is often difficult to do, case-based Reasoning has been introduced as a general problem-solving paradigm that makes use of the notion of analogy. Case-based reasoning uses domain-specific knowledge of previously experienced, concrete problem solutions in order to solve a new problem. It accomplishes this task by finding a similar past case and reusing it in the new problem situation. Part of its feasibility is founded on psychological studies, where it is found that humans often solve new problems by analogy. Several studies have given empirical evidence for the dominating role of specific, previously experienced situations in human problem solving.

In previous case-based planning approaches, one way is to build a plan by transformational analogy which is a problem-solving technique in which a pre-selected plan,

defined as a sequence of actions, is modified to solve a new problem [20]. Possible modifications to the plan include removing actions, adding new actions, and changing the parameters from actions. The CHEF system constructs cooking recipes, which are plans because recipes are sequences of cooking steps such as boiling a certain amount of water [7]. These recipes are modified depending on factors such as the ingredients currently available. Interest on case-based planning has revived recently, e.g., in the work of [5].

The previous approaches on case-based planning mainly focus on adapting *a single* similar plan case in its *entirety*, which is found from the set of plan cases, to a new one that solve a new planning problem. However, there may be some other cases in the same case base that can provide some helpful knowledge in building the new plan, although they may not be so similar to the new problem. Similarly, it may not be the case where a plan is completely reusable; instead, each case that is being reused may only partially contribute to the new solution. Thus, it is important for us to develop a new method to exploit all the partially helpful information to help build a new plan, in order to improve the planning efficiency and improve its effectiveness.

In this paper, we propose a novel case-based planning framework called MAXCBP, which stands for *using a weighted **MAX**-SAT solver to do **C**ases-**B**ased **P**lanning* . In this algorithm, we extract the useful information from all the similar plan cases in the form of constraints. We also formulate the new planning problem as a constraint via a procedure similar to Graphplan[6]. We add the two sets of constraints to help search for a plan for a new planning problem.

In particular, our MAXCBP algorithm works in the following three steps. First, we encode the planning problem as a set of clauses (a satisfaction problem). Secondly, we extract constraints that can be also converted as a set of clauses, from all the plan cases. Finally, we assign weights to all the clauses and solve them with a weighted MAX-SAT solver. We then convert the final solution to a plan solution to the new planning problem. Compared to previous approaches, our algorithm has the advantage that it can selectively use partial knowledge from each case solution as needed to solve a new problem. In addition, we make use of all available cases in the case base to solve a new problem.

The rest of the paper is organized as follows. We first give some related work to this paper, and then address our problem definition. After that, we describe the detailed steps of our algorithm. In the experiment section, we evaluate our algorithm in three planning domains. Finally, we conclude the paper and discuss future work.

2 Related Work

2.1 Case-Based Planning

Case-based planning (CBP) uses a set of cases that consist of a past problem, a goal and a plan that makes a transition from the problem to the goal [7]. Given a new problem, CBP systems retrieve one or more cases whose problem is similar to the current one and adapt the plans contained in the retrieved cases to achieve the new goal. Case retrieval involves an intelligent search among all cases to find the ones that are adequate to solve

the new problem. The result of the plan adaptation process is a solution that includes parts that are derived from selected case and new parts derived by first-principle based planning. [5] analyzes the current state of art in case-based plan adaptation research. This work presents six dimensions for categorizing various aspects of existing case-based plan adaptation algorithms, including the type of transformation, the role of the case content, the use of case merging, the representation formalism, and the computational complexity of the algorithm. It uses these dimensions as a framework to compare various systems.

[11] presents a scheme for learning the case quality based on its utility during a validation phase. The quality obtained determine the way in which these cases are preferred in the retrieval and replay processes. It shows that the planning performance can be improved when case utilities are used. [9] proposes a general framework for transformational analogy. It demonstrates that transformational analogy does not meet a crucial condition for a well-known worst-case complexity scenario, and that plan adaptation can be computationally harder than planning from the scratch, is not applicable for transformational analogy. [8] develops an on-line case-based planning framework. In this framework, when a plan is retrieved, a plan dependency graph is inferred to capture the relations between actions in the plan. Then the plan is adapted in real-time using its plan dependency graph, which allows the system to create and adapt plans in an efficient and effective manner while performing the task.

2.2 Planning as Satisfiability and Weighted MAX-SAT

[2] develops a formal model of planning based on satisfiability. The satisfiability approach uses the best-known logical formalization of planning, based on the situation calculus [4]. In this system, the execution of an action is explicitly represented by the application of a function to a term representing the state in which the action is performed. The satisfiability approach provides not only a more flexible framework for stating different kinds of constraints on plans, but also a more accurate theory behind modern constraint-based planning systems. [3] describes a two-phase algorithm for MAX-SAT and weighted MAX-SAT problems. Firstly, it uses the GSAT heuristic to find a good solution to the problem. Secondly, it uses an enumeration procedure based on the Davis-Putnam-Loveland algorithm to find a provably optimal solution.

2.3 Learning for Planning

Planning problems are often formulated as heuristic search and the choice of the heuristic function plays a significant role in the performance of planning systems. [12] proposes an approach to learning heuristic functions from previously solved problem instances in a given domain. The approach is based on approximate linear programming, which is commonly used in reinforcement learning. [13] presents a novel approach for boosting the scalability of heuristic planners based on automatically learning domain-specific search control knowledge from planning contexts in the form of relational decision trees. The contexts are defined as the set of helpful actions extracted from the relaxed planning graph of a given state, the goals remaining to be achieved, and the

static predicates of the planning task. [14] introduces a novel feature space for representing control knowledge. It defines features in terms of information computed via relaxed plan extraction, which has been a major source of success for non-learning planners, which gives a new way of leveraging relaxed planning techniques in the context of learning. The authors show that the approach is able to surpass state-of-the-art non-learning planners across a wide range of planning competition domains.

3 Problem Formulation

In this work, we consider the restrained form of STRIPS model [15], leaving more complex models such as ADL as our further work. A planning domain is defined in this work as $\Sigma = (S, A, \gamma)$, where S is the set of states, A is the set of action models, γ is the deterministic transition function $S \times A \rightarrow S$. Each action model in A is composed of three parts: an action name with zero or more arguments, a set of preconditions which should be satisfied before the action is executed, and a set of effects which are the results of executing the action. A planning problem can be defined as $\mathcal{P} = (\Sigma, s_0, g)$, where s_0 is an initial state, and g is a goal state. A solution to a planning problem is an action sequence $(a_0, a_1, \ldots, a_n)$ called a plan, which makes a projection from s_0 to g. Each a_i is an *action schema* composed of an action name and zero or more arguments. Furthermore, a *plan case* is defined as $T = (s_0, a_0, a_1, \ldots, a_n, g)$. Notice that the intermediate states between actions can be computed using the action models. In this paper, we denote a set of *plan cases* as PC.

Our problem can be formulated as follows. We are given a planning domain Σ, a set of *plan cases* PC and a new plan problem P. Our algorithm MAXCBP outputs a plan for the new plan problem P using the helpful information from PC. An example problem description is given in Table 1, which is from the domain of *blocks*[3].

4 The MAXCBP Algorithm

Our solution is to first encode the new planning problem into a constraint satisfaction problem. The idea similar to Graphplan [6], where we consider K steps of actions in which to solve a problem. When we solve the problem using a constraint satisfaction problem formulation, the solution may be time-consuming, Thus, we make use of the knowledge from the case base PC, which we convert to another set of constraints. The two sets of constraints are combined into a new satisfiability problem, which is solved using a MAXSAT solver.

In the following, we first give an overview of our algorithm MAXCBP. A detailed description of each steps will be given in sections 4.1-4.4.

4.1 Encoding Planning Problem

In step 1 of Algorithm 1, we encode a planning problem $\mathcal{P}$ as a satisfiability problem [2], which is simply a set of axioms with the property that any model of the axioms

[3] http://www.cs.toronto.edu/aips2000/

Table 1. an example problem description

input: the domain of *blocks*	
predicates	action models
(on ?x - block ?y - block) (ontable ?x - block) (clear ?x - block) (handempty) (holding ?x - block)	(pick-up ?x - block) preconditions: (clear ?x)(ontable ?x)(handempty) effects: (holding ?x)(not(ontable ?x))(not(clear ?x))(not(handempty)) (put-down ?x - block) **(preconditions & effects omitted)** (stack ?x - block ?y - block) **(preconditions & effects omitted)** (unstack ?x - block ?y - block) **(preconditions & effects omitted)**

input: plan cases
plan case 1: (ontable A) (ontable B) (clear A) (clear B) (handempty), (pick-up A) (stack A B), (handempty) (on A B) plan case 2: (ontable B) (on A B) (clear A) (handempty), (unstack A B) (put-down A) (pick-up B) (stack B A), (on B A) (ontable A) ...

input: a new planning problem	
initial state s_0	goal g
(ontable A) (clear A) (holding B)	(on A B) (ontable B)

output: a plan for the new planning problem
(put-down B) (pick-up A) (stack A B)

Algorithm 1 Overview of our MAXCBP algorithm

Input: a new planning problem $\mathcal{P} = (\Sigma, s_0, g)$, a set of plan cases PC;
Output: a plan P for the planning problem $\mathcal{P}$;

1: we first consider a plan graph according to Graphplan [6], where the plan is of length k (k is set to one initially, and incremented by one if no solution is found). We encode the planning problem $\mathcal{P}$ as a set of clauses C (a satisfaction problem) for a solution plan of up to length k;
2: build constraints C1-C5 from PC;
3: assign each constraint of C1-C5 with its appearing frequency as its weight;
4: assign each clause in C with a high weight that the clause should hold;
5: solve all the weighted constraints with a weighted MAXSAT solver;
6: if no solution is found, we increment k by one, and go to step 1.
7: convert the solved result to a plan P;
8: **return** P;

corresponds to a valid plan. Some of these axioms describe the initial s_0 and goal states g. For the example in Table 1, s_0 and g can be described as

$$(ontable\ A\ 1) \wedge (clear\ A\ 1) \wedge (holding\ B\ 1) \wedge (on\ A\ B\ \infty) \wedge (ontable\ B\ \infty)$$

The other axioms describe the actions in general, which include the standard effect and frame axioms, plus others that rule out the anomalous models.

First, we rule out the possibility that an action executes despite the fact that its preconditions are false. This can be done by asserting that an action implies its precon-

ditions as well as effects; e.g., for preconditions of the action $pick\text{-}up$ in Table 1,

$$\forall x, i.(pick\text{-}up\ x\ i) \rightarrow (clear\ x\ i) \wedge (ontable\ x\ i) \wedge (handempty\ i)$$

and for effects,

$$\begin{aligned}\forall x, i.(pick\text{-}up\ x\ i) \rightarrow (holding\ x\ i{+}1) \wedge \neg(clear\ x\ i{+}1) \\ \wedge\neg(ontable\ x\ i{+}1) \wedge \neg(handempty\ i{+}1)\end{aligned}$$

It is interesting to note that in this formulation preconditions and effects are treated symmetrically.

Next, we state that only one action occurs at a time, e.g.,

$$\forall x, x', i.(x \neq x') \rightarrow \neg(pick\text{-}up\ x\ i) \vee \neg(pick\text{-}up\ x'\ i)$$

Finally, we assert that some action occurs at every time step. This is not a significant restriction, since we can always introduce an explicit "do nothing" action if desired. For the action $pick\text{-}up$ in Table 1, the axiom schema is

$$\forall i < N.\exists x, (pick\text{-}up\ x\ i)$$

(An existentially-quantified formula expands to the disjunction of its instantiations.)

If a planning problem is specified by asserting a complete initial state then these axioms guarantee that all models correspond to valid plans. This is so because every model contains a sequence of actions whose preconditions are satisfied, and the execution of an action in a state completely determines the truth-values of all propositions in the next state. The only model of the simple two step planning problem is the intended model containing $(put{-}\mathrm{down\ B\ 1})$, $(pick{-}\mathrm{up\ A\ 2})$ and $(stack\ A\ B\ 3)$. A simple planning system can be constructed by linking a routine that instantiates such a given set of axiom schemas and initial and goal state specifications to a Boolean satisfiability algorithm.

4.2 Building Constraints

In step 2, we wish to build constraints to represent the relationship between actions in PC. We observe that there are five kinds of relationships between each two actions in a plan case, i.e.,

1. one action provides a precondition for its subsequent action (we call this relationship as an *add-pre constraint*);
2. one action adds an effect but deleted by its subsequent action (we call this relationship as an *add-del constraint*);
3. one action deletes an effect but added by its subsequent action (we call this relationship as a *del-add constraint*);
4. one action shares a precondition with its subsequent action (we call this relationship as a *pre-pre constraint*);
5. a precondition of one action is deleted by its subsequent action (we call this relationship as a *pre-del constraint*).

We denote a plan case $pc \in PC$ as $pc = (a_1, a_2, \ldots, a_n)$, each action pair in pc as $\langle a_i, a_j \rangle$ where $1 \leq i < j \leq n$. We formulate the idea as follows.

C1: *add-pre constraints* For each action pair $\langle a_i, a_j \rangle$, the idea that a_i provides a precondition for a_j is, there is a proposition p which is added by a_i and used as a precondition of a_j. We denote a list of effects added by a_i as add_i, and a list of preconditions of a_j as pre_j. Then, we can formulate this constraint as follows.

$$p \in add_i \wedge p \in pre_j$$

where parameters of p are included by a_i and a_j. Intuitively, the action a_j requires that a_i should be executed first in a plan, that a_j can be executed and produce some useful effects.

C2: *add-del constraints* For each action pair $\langle a_i, a_j \rangle$, the idea that a_i adds an effect but deleted by a_j is, there is a proposition p which is added by a_i and used as a precondition of a_j. We denote a list of effects deleted by a_j as del_j. Then, we can formulate this constraint as follows.

$$p \in add_i \wedge p \in del_j$$

Intuitively, a_j needs to be executed to deleted a redundantly added effect by a_i, that no unnecessary actions will be executed after a_j in a plan.

C3: *del-add constraints* For each action pair $\langle a_i, a_j \rangle$, there is a proposition p which is deleted by a_i and added by a_j. Then, this constraint can be formulated by

$$p \in del_i \wedge p \in add_j$$

This constraint specifies that, a_j is needed to add an effect which is unexpectedly deleted by a_i.

C4: *pre-pre constraints* For each action pair $\langle a_i, a_j \rangle$, there is a proposition p which is a precondition of a_i and a_j. Then, this constraint can be formulated by

$$p \in pre_i \wedge p \in pre_j$$

This constraint specifies that different actions may be executed together under the same preconditions, e.g., frame axioms which are not changed between these actions in a plan.

C5: *pre-del constraints* For each action pair $\langle a_i, a_j \rangle$, there is a proposition p which is a precondition of a_i but deleted by a_j. Then, this constraint can be formulated by

$$p \in pre_i \wedge p \in del_j$$

This constraint specifies that a_j should be executed to delete p that actions with the precondition p will not be executed again in a plan.

With respect to the restrained form of STRIPS model, we assert that the above five kinds of constraints encode all the possible relationship between actions. Before giving proof to this conclusion, we provide the following two requirements according to the restrained form of STRIPS model, i.e.,

R1: A proposition p added by action a_i should not be a precondition of a_i, i.e.,

$$p \in add_i \rightarrow p \notin preState_i$$

where $preState_i$ is a list of propositions that exist before a_i is executed. Notice that $pre_i \subseteq preState_i$.

R2: A proposition p deleted by action a_i should be a precondition of a_i, i.e.,

$$p \in del_i \rightarrow p \in pre_i$$

Then, we have the theorem under the conditions of R1-R2, as shown in the following. Notice that, when we consider a proposition referring to an action pair $\langle a_i, a_j \rangle$, we assume that there is no other actions between a_i and a_j that affect the proposition.

Theorem: constraints C1-C5 encode all the possible relationships between two actions of $\langle a_i, a_j \rangle$ in a plan.

proof: For each action pair $\langle a_i, a_j \rangle$, the relationships between a_i and a_j can be specified as whether or not, a proposition p in add_i,pre_i, or del_i is also in add_j, pre_j, or del_j. That is to say, there are nine kinds of relationships: $\{p \in add_i \wedge p \in pre_j, p \in add_i \wedge p \in add_j, p \in add_i \wedge p \in del_j, p \in del_i \wedge p \in pre_j, p \in del_i \wedge p \in pre_j, p \in del_i \wedge p \in add_j, p \in del_i \wedge p \in del_j, p \in pre_i \wedge p \in pre_j, p \in pre_i \wedge p \in add_j$, and $p \in pre_i \wedge p \in del_j\}$. In another word, we only need to prove that $\{p \in add_i \wedge p \in add_j, p \in del_i \wedge p \in pre_j, p \in del_i \wedge p \in del_j$, and $p \in pre_i \wedge p \in add_j\}$ can be deduced by C1-C5, or they are contradictive with R1-R2.

First, if $p \in add_i \wedge p \in add_j$ holds, then $p \in preState_j$ holds. That is to say, $p \in add_j \wedge p \in preState_j$ holds, which is contradictive with R1. Thus, $p \in add_i \wedge p \in add_j$ is contradictive with R1.

Second, if $p \in del_i \wedge p \in pre_j$ holds, then $p \notin preState_j \wedge p \in pre_j$ holds. Since $pre_j \subseteq preState_j$ holds, $p \notin preState_j \Rightarrow p \notin pre_j$, which implies $p \notin pre_j \wedge p \in pre_j$, i.e., contradiction is generated.

Third, if $p \in del_i \wedge p \in del_j$ holds, then $p \notin preState_j \wedge p \in del_j$ holds. And then $p \notin preState_j \wedge p \in pre_j$ holds. Similar to the second one, contradiction will be generated.

Finally, from C3, we have $p \in del_i \wedge p \in add_j$. Then we have $p \in pre_i \wedge p \in add_j$ by R2. On the other hand, if we have $p \in pre_i \wedge p \in add_j$, then we have $p \in pre_i \wedge p \notin preState_j$ by R1, which implies $p \in del_i$. Thus, $p \in del_i \wedge p \in add_j$ holds. That is to say, $p \in pre_i \wedge p \in add_j$ is unnecessary, since C3 and R1-R2 have encoded the information it provides.

Briefly, by considering R1-R2, C1-C5 have encoded all the possible relationships between a_i and a_j. □

4.3 Assigning Weights

By steps 1-2, we have built a list of constraints. In this section, we present how to assign weights to constraints, which corresponds to steps 3-4 of Algorithm 1. Our basic idea is that (1) the weights of constraints built by step 1 should be high enough to ensure a planning problem being solved correctly; (2) the correct information included by the plan cases PC corresponds to the constraints frequently satisfied by PC, while the

other information corresponds to the constraints infrequently satisfied. Thus, to explore the correct information, we calculate the frequency of constraints satisfied by PC as weights. Based on this idea, we give the algorithm of assigning weights to constraints in Algorithm 2.

Algorithm 2 assigning weights to constraints

Input: a planning domain Σ, a set of plan cases PC;
Output: the weights W_1, W_2, W_3, W_4, W_5 for C1, C2, C3, C4, C5;

1: $C_1 = C_2 = C_3 = C_4 = C_5 = \emptyset$ are sets of constraints of C1-C5 respectively;
2: **for** each plan case $pc \in PC$ **do**
3: **for** each two actions a_i and a_j in pc **do**
4: **if** $i < j$ and there is a proposition p that is not affected by actions between a_i and a_j (can be asserted by executing the actions using Σ) **then**
5: **for** k = 1 to 5 **do**
6: **if** p, a_i and a_j form a constraint c that satisfies Ck **then**
7: put c in C_k;
8: **end if**
9: **end for**
10: **end if**
11: **end for**
12: **end for**
13: **for** k = 1 to 5 **do**
14: unify C_k into variable form;
15: count the appearing number of each constraint in C_k; the results are stored in a vector W_k, viewed as weights;
16: **end for**
17: **return** W_1, W_2, W_3, W_4, W_5;

In step 14 of Algorithm 2, each constraint in C_k is unified by substituting all the parameters of p, a_i and a_j with unified variables. We get the weights of constraints of C1-C5. Since we wish to solve a new planning problem correctly, we set weights (denoted as W_0) of constraints (clauses, denoted as C) generated in step 1 of Algorithm 1 as high as possible, by considering the effect of R1-R2 simultaneously. To do this, we first find the maximal value from $W_1, W_2, \ldots, W_5$, which is denoted as w_{max}. Then, we set W_0 by the following way:

$$\forall k > 0, W_0(k) = \beta w_{max}$$

where $W_0(k)$ is a weight of kth constraint (clause) in C. β is a parameter to adjust the value of $W_0(k)$. By setting different value of β, the weights of constraints in C will be changed. As a result, the importance of the constraints in C will be changed correspondingly in the whole solving process of our algorithm `MAXCBP`. In our subsequent experiments, we will test different value of β to see its effect on the experiment result.

4.4 Obtaining a Final Solution Plan

In steps 5-7 of Algorithm 1, we solve the weighted constraints of C and C1-C5 using a weighted MAX-SAT solver, the result of which is an assignment to all the axioms of constraints. With the assignment, we attain a plan by this way: first, we select all the axioms assigned with a *true* value; and then we convert all the selected axioms, which represent actions being executed or not, to a plan. Next, we will give a whole example for our algorithm MAXCBP in the following.

Example: *For the planning problem in Table 1, the solving process of our algorithm* MAXCBP *is shown in Fig. 1. In this figure, the omitted parts denoted by "..." are the ones can be builded similarly by what are builded prior to them. In steps 3 and 4, the numbers "2" and "$\beta * 2$" are weights of C1-C5 and C respectively. In step 6, except the ones assigned to be true, there are other propositions assigned to be false or true (e.g., initial state and goal), which are not shown in the figure (denoted by "..."). From this example, we can see the detail steps about how to find a plan using a weight MAX-SAT solver.*

5 Experiment Results

In this section, we evaluate our algorithm MAXCBP in the following three benchmark planning domains: *blocks*, *depots*[4] and *driverlog*[4]. We generated 150 plan cases from each domain. Furthermore, we generated 50 planning problems from each domain, which will be solved by our algorithm MAXCBP. We evaluate MAXCBP by testing the running time and average length of all the plans according to different number of cases and different value of β which is a coefficient of w_{max}. Notice that we consider the efficiency and effectiveness of our algorithm MAXCBP by testing the running time and average length of plans respectively. We run our algorithm on the PC with CPU 2.26GHZ and memory 1GMB. We define the average length of all the plans as

$$A = \frac{\sum_{p \in P} lengthof(p)}{|P|}$$

where P is a set of plans and the procedure $lengthof(p)$ returns the length of the plan p. In the following, we give the experimental results according to different number of cases and different values of β.

5.1 Different Number of Cases

In this experiment, we test the running time of our algorithm according to different number of plan cases being used in three different domains. The result is shown in Fig. 2, where the vertical line denotes the CPU time which is taken to solve all the 50 planning problems, and the horizontal line denotes the number of plan cases. In this figure, likewise for Fig. 4, the horizontal line signed with "without cases" shows the running result with the SATPLAN [5] without exploiting any information of plan

[4] http://planning.cis.strath.ac.uk/competition/

[5] http://www.cs.rochester.edu/u/kautz/satplan/index.htm

---- Begin ----
Step 1: initial state and goal: (ontable A 1) ^ (clear A 1) ^ (holding B 1) ^ (on A B ∞) ^ (ontable B ∞)
action: forall x, i. (pick-up x i) → (clear x i) ^ (ontable x i) ^ (handempty i)
…
other constraints: forall x, x', i. (x ≠ x') → ¬(pick-up x i) V ¬(pick-up x' i)
forall i < N, exists x. (pick-up x i)

Step 2: C1-C5 from plan cases 1 (likewise for plan case 2):
C1 add-pre constraints: (holding A 2) ∈ add(pick-up$_1$) ^ (holding A 2) ∈ pre(stack$_2$)
C2 add-del constraints: (holding A 2) ∈ add(pick-up$_1$) ^ (holding A 2) ∈ del(stack$_2$)
…

Step 3: assigning weights of C1-C5:
2 (holding A 2) ∈ add(pick-up$_1$) ^ (holding A 2) ∈ pre(stack$_2$)
2 (holding A 2) ∈ add(pick-up$_1$) ^ (holding A 2) ∈ del(stack$_2$)
…

Step 4: assigning weights of C:
β*2 (ontable A 1) ^ (clear A 1) ^ (holding B 1) ^ (on A B ∞) ^ (ontable B ∞)
β*2 …

Step 5: solving weighted clauses from Step 3 and Step 4 using a weighted MAX-SAT solver, the result is:
(put-down B 1) true
(pick-up A 2) true
(stack A B 3) true
…

Step 6: convert the result of Step 5 to a plan to the new planning problem:
(put-down B) (pick-up A) (stack A B)

---- End ----

Fig. 1. an example of solving a new planning problem using MAXCBP

cases, and the curve signed with "with cases" shows the running result of our algorithm MAXCBP.

From Fig. 2, we find that the running time of our algorithm MAXCBP that exploits the information of plan cases is generally lower than the one without using any information. Furthermore, from the curves in Fig. 2 (a)-(c), we also find that the CPU time of our algorithm MAXCBP goes down when the number of plan cases goes up. That is because, the more the plan cases are given, the more the information can be used, that a plan can be found by MAXCBP more efficiently with the help of the information.

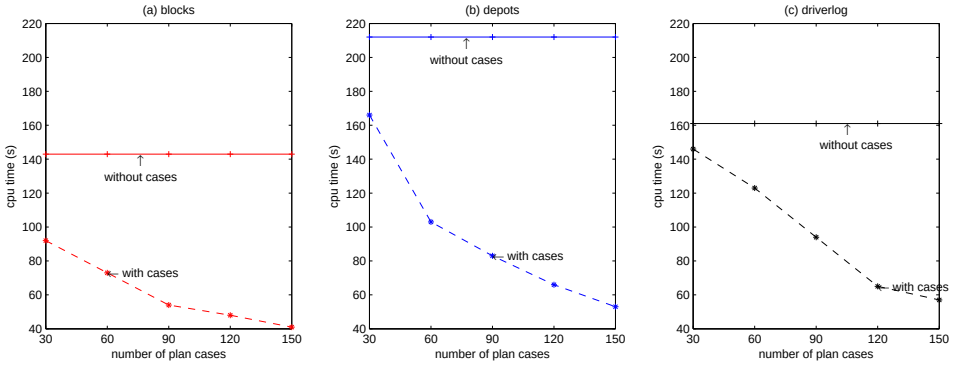

Fig. 2. the CPU time with respect to the number of plan cases

To see the effect our algorithm MAXCBP introduces, we show the result of the average of plan length A with respect to the number of plan cases in Fig. 3. From this figure, we find that the curves are generally go down when the number of plan cases increases. That is because, the information of plan cases can help MAXCBP to find a *shorter* plan, rather than a *longer* one. Generally speaking, a *shorter* plan to a problem suggests that the problem is solved more efficiently (with fewer actions).

5.2 Different Values of β

In section 4.3, we assign the weights of the clauses C as $W_0(k) = \beta w_{max}$, where different β will result in different CPU time or plan length. The results can be seen from Fig. 4 and 5, where we fix the number of plan cases as 150 and test β with the values from 1 to 5.

For the CPU time in Fig. 4, we find that the CPU time is generally lower when using the information of plan cases than without it, by comparing the curves denoted with "with cases" for planning using the case base, and the horizontal lines denoted with "without cases" to denote planning from first principles without using the case base, from Fig. 4 (a)-(c). After testing the weight factor β, we find that the CPU time generally increases with the value of β . That is because, when β increases, the information that plan cases can provide is reduced, which means MAXCBP will take more CPU time to find a plan when using less information from the plan cases.

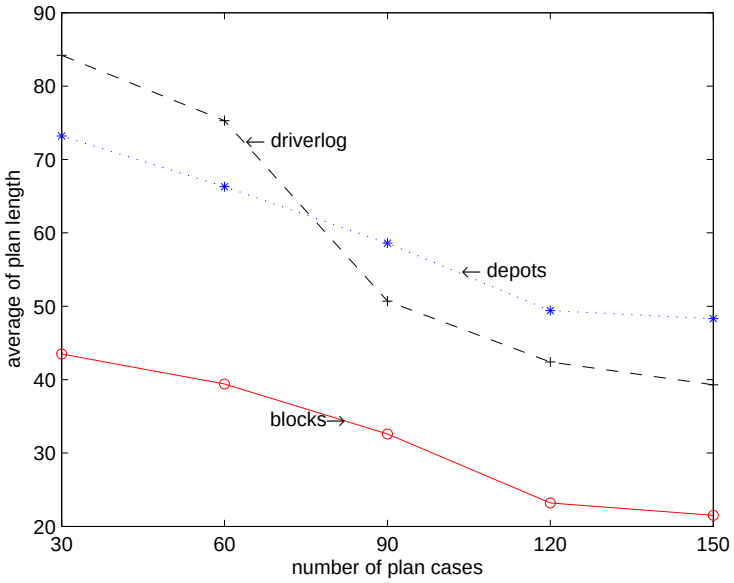

Fig. 3. the average of plan length with respect to the number of plan cases

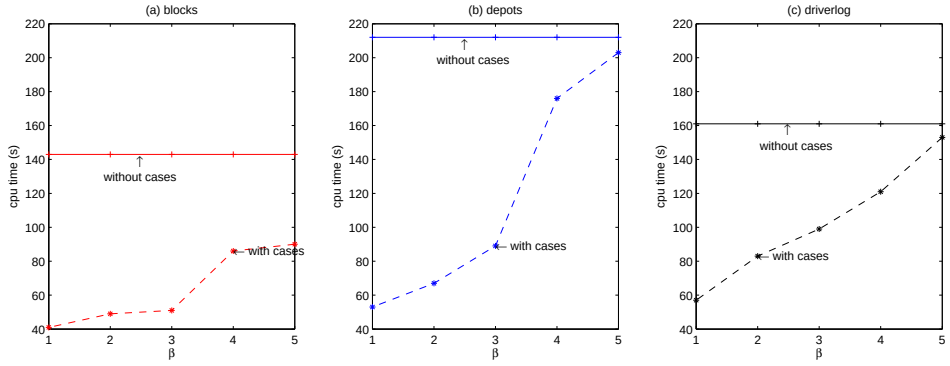

Fig. 4. the CPU time with respect to different values of β

For the plan length in Fig. 5, we find that the average length of plans for all the 50 problems is generally going up when the value of β increases. Similar to Fig. 4, when β increases, the effect of plan cases on helping finding a shorter plan is weakened, that the plans being found by MAXCBP will be longer.

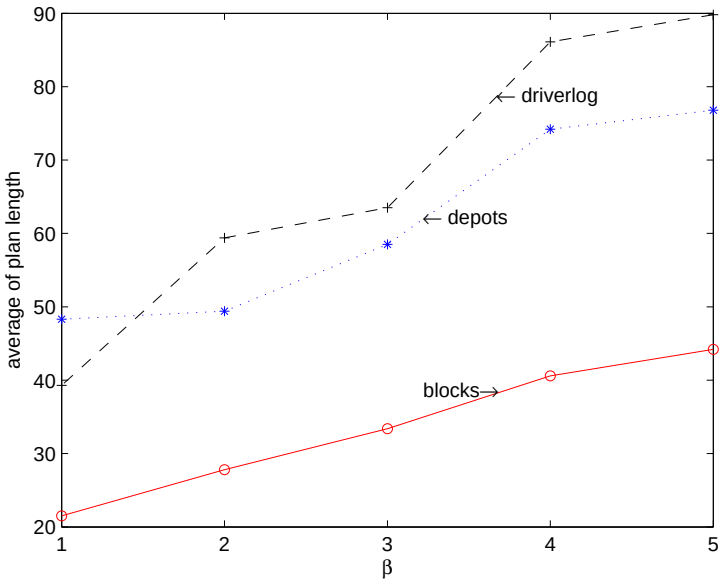

Fig. 5. the average length of plans with respect to different values of β

From Fig. 2-5, we conclude that, our algorithm MAXCBP, which is to maximally exploit the information of plan cases, will help improve the efficiency and effectiveness of finding a plan to a new planning problem.

6 Conclusion

In this paper, we presented a novel approach for case-based planning called MAXCBP. Our algorithm makes maximal use of the cases in the case base to find a solution by maximally exploiting the information of plan cases via using a weighted MAX-SAT solver. Our system can take a piece of the useful plan knowledge in the form of constraints even when the entire plan may not be useful for solving a new problem. Our empirical tests show that our method is both efficient and effective in solving a new planning problem. In real world applications, attaining a set of plan cases by hand is difficult and time-consuming. Thus, in our future work, we will consider the situation that plan cases are observed automatically by machine such as sensors. In this situation,

plan cases will contain noise, which makes our task more difficult. Thus, one of our future works is to extend the framework by considering more noisy cases.

Acknowledgment

We thank the support of Hong Kong CERG Grant HKUST 621307, NEC China Lab.

References

1. Vithal Kuchibatla and Hector Munoz-Avila. An Analysis of Transformational Analogy: General Framework and Complexity. ECCBR, 458-473, 2006.
2. Henry Kautz and Bart Selman. Planning as Satisfiability. ECAI, 1992.
3. Brian Borchers and Judith Furman. A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems. Journal of Combinatorial Optimization, 2(4), 299-306, 1998.
4. John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. Machine Intelligence, Pages 463-502, 1969.
5. Hector Munoz-Avila and Michael T. Cox. Case-Based Plan Adaptation: An Analysis and Review. IEEE Intelligent Systems, 2007.
6. Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. Artificial Intelligence, Vol.(90), pages 1636-1642, 1997.
7. Hammond K. J. Case-Based Planning: Viewing Planning as a Memory Task. San Diego, CA: Academic Press, 1989.
8. Neha Sugandh, Santiago Ontanon and Ashwin Ram. On-Line Case-Based Plan Adaptation for Real-Time Strategy Games. AAAI, 702-707, 2008.
9. Vithal Kuchibatla and Hector Munoz-Avila. An Analysis on Transformational Analogy: General Framework and Complexity. ECCBR, 2006.
10. Javier Bajo, Juan Manuel Corchado and Sara Rodriguez. Intelligent Guidance and Suggestions Using Case-Based Planning. ICCBR, 2007.
11. Tomas de la Rosa, Angel Garcia Olaya and Daniel Borrajo. Using Cases Utility for Heuristic Planning Improvement. ICCBR, 2007.
12. Marek Petrik and Shlomo Zilberstein. Learning Heuristic Functions Through Approximate Linear Programming. ICAPS, 2008.
13. Tomas de la Rosa, Sergio Jimenez and Daniel Borrajo. Learning Relational Decision Trees for Guiding Heuristic Planning. ICAPS, 2008.
14. Sungwook Yoon, Alan Fern and Robert Givan. Learning Control Knowledge For Forward Search Planning. JMLR, 9 (APR), 683-718, 2008.
15. Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving, Artificial Intelligence, 189-208, 1971.
16. Qiang Yang. Intelligent Planning: A Decomposition and Abstraction Based Approach. Berlin, Germany: Springer Verlag, 1997.
17. David Chapman. Planning for Conjunctive Goals. Artificial Intelligence 32: 333-377, 1987.
18. Wilkins, D. E. Recovering from Execution Errors in SIPE. Computational Intelligence 1: 33-45, 1985.
19. Selman, B., Levesque. H. and Mitchell, D. Hard and Easy Distributions of SAT Problems. In Proc. of the 10th National Conference on Artificial Intelligence, 440-446. San Jose, CA: AAAI Press/MIT Press, July 1992.
20. Carbonell, J.G. Learning by analogy: formulating and generalizing plans from past experience. Machine Learning: An Artificial Intelligence Approach. R.S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.). Tioga, Palo Alto, California, 1983.