Applied Mathematics & Information Sciences
An International Journal

A Planning Heuristic Based on Subgoal Ordering and Helpful Value

Weisheng Li[1] , *Peng Tu*[1] *and Junqing Liu*[2]

[1] College of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing 400065, China
[2] College of Computer and Information Technology, China Three Gorges University, Yichang 443002, China

Received: Jul 8, 2011; Revised Oct. 4, 2011; Accepted Oct. 6, 2011
Published online: 1 August 2012

Abstract: This work is concerned with how to improve the efficiency of heuristic search in a planning system. Utilize the dependency relations between the variables in a goal, a subgoal ordering method is first used to guide the heuristic search in a more reasonable way. The idea of helpful value in a goal is then introduced. A more accurate heuristic cost can be achieved by using the helpful value when we compute the heuristic cost. Finally, a heuristic algorithm combined subgoal ordering with helpful value is proposed. The algorithm is implemented in the planning system Fast Downward. The experimental results show the efficiency of the proposed heuristic search algorithm on the benchmarks of International Planning Competitions (IPC) 2008.

Keywords: Intelligent planning, Subgoal ordering, Helpful value, Heuristic.

1. Introduction

As an important research field in artificial intelligence, intelligence planning has got much attention of researchers in recent years. Planners based on the ideas of heuristic search are very popular in intelligence planning area due to their efficiency in solving problems. Several of the well-known heuristic state search planners are such as HSP[1], FF[2], Fast Downward[3, 4], and so on. HSP use the additive heuristic for solving domain-independent plan. The relaxed plan heuristic is used in FF, which encodes the cost of a specific relaxed plan.

Causal graph heuristic is used in Fast Downward to capture the underlying causal structure of the domain, which is based on hierarchical decomposition of planning tasks. In Fast Downward, a planning problem described by PDDL is translated to a multiple planning task (MPT) to reduce the state space. The additive heuristic[5] is used in causal graph heuristic, and the additive heuristic function of a state is defined as the sum of heuristic costs of all the variables in the goal. In Fast Downward, The sum of all the transition conditions cost is added into the cost of the variables transition. If variables in the goal or in the conditions of one transition are dependent, additive heuristic is inadmissible. When there are positive interactions between these variables, additive heuristic may overestimate the cost.

In this paper, a heuristic search based on dependency relations of the variables in a goal is proposed. It combines the dependency relations of the variables of goals and helpful values of goal state. By ordering the variables of a goal, we can get a more reasonable subgoal[6] sequence. Every time we take the first subgoal in the subgoal sequence to compute the heuristic cost. Then, by extracting the positive interactions between the variables, one can compute the helpful value in each subgoal. Use the heuristic cost to minus the helpful value, one can get a more accurate heuristic estimates to guide the search. The experimental results show that the plan length can be effectively reduced in our method.

The rest of the paper is organized as follows. The preliminaries of PDDL descriptions language and heuristic search are introduced in Section 2. Section 3 shows the architecture of Fast Downward. Section 4 illustrates the dependency relations of the variables of goal and helpful value. Section 5 provides a detailed description of the heuristic based on dependency relations of the variables of goal. Our results are proposed in section 6. In the last section, we summarize the paper and propose our future work.

* Corresponding author: e-mail: liws@cqupt.edu.cn

2. Preliminaries

We first introduce the preliminaries of The Planning Domain Definition Language (PDDL) and heuristic.

2.1. PDDL

PDDL[7] is the standard encoding language for classical planning tasks. It is widely used in the international planning competitions to express the planning tasks.

A planning task is a 4-tuple

$$Q = (F, A, I, G) \tag{1}$$

In Equation (1), the finite set F describes the domain propositional state variables, with any state s expanded in the search $s \subseteq F$. The finite set A describes the domain actions in a 3-tuple term, each action $a \in A$ is associated preconditions $pre(a) \subseteq F$, add effects $add(a) \subseteq F$, delete effects $del(a) \subseteq F$. We call an action is applicable in a state $s \subseteq F$, if the precondition of the action $pre(a) \subseteq s$. The result of the application $s[a]$ is given by its add-effect $add(a)$ and its delete-effect $del(a)$ removing. Similarly, applying an action sequence $a_1, a_2, \ldots, a_n$ to a state s is donated as $s[a_1, a_2, \ldots, a_n]$. In particular, if $G \subseteq s[a_1, a_2, \ldots, a_n]$, we call the sequence $< a_1, a_2, \ldots, a_n >$ a plan.

2.2. Heuristic

Relaxing a planning problem is a basic process for heuristic. We use the heuristic to estimate the cost of current state. Relaxed planning graph (RPG) is to reason by Graph-plan over the delete-relaxed problems. It is NP-hard to find optimal relaxed plan for a task, still there are good approximation strategies, and one of them is the delete-relaxed plan. The delete-relaxed planning Q' ignores the delete effects of actions in a planning task Q. A delete-relaxed action $a' \in A'$, with its preconditions $pre(a') = pre(a)$, add effects $add(a') = add(a)$, and delete effects $del(a') = \phi$. The definition of relaxed planning problem are

$$Q' = (F, A', I, G) \tag{2}$$

and

$$A' = \{(pre(a), add(a), \phi) | \\ (pre(a), add(a), del(a)) \in A\} \tag{3}$$

The Fast Downward planning system uses the causal graph heuristic in which the additive heuristic is used. The steps of heuristic search are shown as follows.

Step 1. Take the initial state I as current state S.

Step 2. In the domain actions A, Find all the available actions sequence $(a_1, a_2, \ldots, a_n)$, which means that $pre(a_i) \subseteq S(1 \leq i \leq n)$.

Step 3. Execute these actions on the current state S, one will get reachable state sequence $(s_1, s_2, \ldots, s_n)$. If the goal state appeared in these states, it means that we have got the plan of the planning task. If not, estimate the cost of changing state from current state to goal state by the heuristic function, and take the state which has the minimal cost as the next current state.

Step 4. Repeat the process, until the goal state appeared or there is no more reachable state or actions could get. The definition of additive heuristic is

$$h(s) = \sum_{v \in G} cost(s(v), s_G(v)) \tag{4}$$

where $cost(s(v), s_G(v))$ represents the heuristic cost of changing a variable v from a value in state s to another value in the goal G. When one compute the heuristic cost of v, it is necessary to combine the domain transition graph and causal graph which will be described in the next section.

3. Architecture of Fast Downward

The Fast Downward planning system solves a planning task in three phases: translation, knowledge compilation and search. We will review the three components in this section.

3.1. Translation

The main purpose of translation is to transform classical STRIPS planning tasks to multi-valued planning tasks (MPT). This form is based on the SAS$^+$ planning model[10], in which state variables are allowed to have non-binary finite domains. A multi-valued planning task is given by a 5-tuple

$$\Pi =< V, s_0, s_*, O > \tag{5}$$

with the following components.

V is a finite set of state variables, each with an associated finite domain D_v. State variables are partitioned into fluent (affected by operators) and derived variables (computed by evaluating axioms). The domains of derived variables must contain the default value $\perp$.

A partial variable assignment or partial state over V is a function s on some subset of V such that $s(v) \in Dv$ wherever $s(v)$ is defined. A partial state is called an extended state if it is defined for all variables in V and a reduced state or state if it is defined for all fluent in V. In the context of partial variable assignments, we write $v = d$ for the variable-value pairing $< v, d >$ or $v \rightarrow d$.

The tuple s_0 is a state over V called the initial state.

The tuple s_* is a partial variable assignment over V called the goal.

O is a finite set of (MPT) operators over V. An operator $< pre, eff >$ consists of a partial variable assignment pre over V called its precondition, and a finite set of effects eff. Effects are triples $< cond, v, d >$, where $cond$ is a (possibly empty) partial variable assignment called the effect condition, v is a fluent called the affected variable,

and $d \in D_v$ is called the new value for v. O include a set of (MPT) axioms over V. Axioms are triples of the form $< cond, v, d >$, where $cond$ is a partial variable assignment called the condition or body of the axiom, v is a derived variable called the affected variable, and $d \in D_v$ is called the derived value for v. The pair $< v, d >$ is called the head of the axiom and can be written as $v := d$.

We take the transportation tasks as an example. Three kinds of objects are included in the transportation domain: $locations$, $trucks$ and $cargos$. Three operators are defined: $load$ cargo at one location, $unload$ cargo at one location and $move$ car from one location to another one.

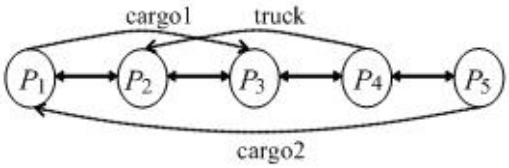

Figure 1 The planning task $Task1$.

Figure 1 shows one transportation task signed $Task1$: the nodes represent locations, where gray node is the initial location of the truck, and solid arcs represent paths between different locations. In $Task1$, there are 5 locations, one truck and two cargos. Dashed arcs show the goal: moving the $truck$ located at P_4 to P_2, moving the $cargo1$ at P_1 to P_3, and moving the $cargo2$ at P_5 to P_1.

At the translation step, $Task1$ is transformed into a MPT $\Pi =< F, s_0, s_*, A >$:

(1)State variables set: $F = (f_a, f_b, f_t)$. Three variables f_a, f_b and f_t define the state of $cargo1$, $cargo2$ and the truck $locations$ respectively. The domain of f_a and f_b have six values: $P_1, P_2, P_3, P_4, P_5, T$, and f_t has five different values: P_1, P_2, P_3, P_4, P_5.

(2)Operators set:

$$A = \{< (f_t = P_1), (f_t = P_2) >, \\ < (f_a = P_1, f_t = P_1), (f_a = T) >, \\ < (f_a = T, f_t = P_5), (f_a = P_5) >\}$$

Here, we only list three operators which are respectively: $move$, $load$ and $unload$.

(3)Initial state:

$$I = \{f_a = P_1, f_b = P_5, f_t = P_4\}$$

(4)Goal state:

$$G = \{f_a = P_3, f_b = P_1, f_t = P_2\}$$

3.2. Knowledge compilation

Knowledge compilation comprises three items. First, we compute the domain transition graph of each state variable. The domain transition graph for a state variable encodes under what circumstances that variable can change its value, i. e., from which values in the domain there are transitions to which other values, which operators or axioms are responsible for the transition, and which conditions on other state variables are associated with the transition.

Second, we compute the causal graph of the planning task. The causal graph encodes dependencies between different state variables Where domain transition graphs encode dependencies between values for a given state variable.

Third, we compute two data structures that are useful for any forward-searching algorithm for MPTs, called successor generators and axiom evaluators.

Considering $Task1$, the domain transition graphs are shown in Figure 2 and Figure 3, where the labels are transition conditions. No label in Figure 2 shows that the transition of the truck does not depend on other variables. Figure 3 shows that the transition of cargos is affected by the truck location. For example, moving $cargo1$ from $T(f_a = T)$ to $P_1(f_a = P_1)$ needs the condition that the truck location is $P_1(f_t = P_1)$, which corresponds to the operator unload the cargo at P_1. So an arc from variable f_a to f_t and f_b to f_t is included in the causal graph shown in Figure 4. We say that f_a and f_b depends on f_t, or f_t affects f_a and f_b, where f_a and f_b is high-level variable and f_t is low-level variable.

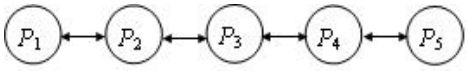

Figure 2 Domain transition graph of f_a and f_b.

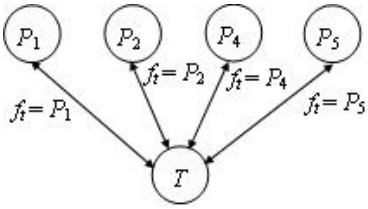

Figure 3 Domain transition graph of f_t.

3.3. Search

Unlike the translation and knowledge compilation components, for which there is only a single mode of execution, the search component of Fast Downward can perform its work in various alternative ways. There are three basic search algorithms to choose from:

(1)Greedy best-first search. This is the standard textbook algorithm, modified with a technique called deferred

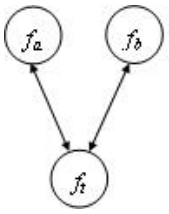

Figure 4 Causal graph of $Task1$.

heuristic evaluation to mitigate the negative influence of wide branching. Fast Downward extended the algorithm to deal with preferred operators, similar to FF's helpful actions. Fast Downward uses this algorithm together with the causal graph heuristic.

(2)Multi-heuristic best-first search. This is a variation of greedy best-first search which evaluates search states using multiple heuristic estimators, maintaining separate open lists for each. Like a variant of greedy best-first search, it supports the use of preferred operators. Fast Downward uses this algorithm together with the causal graph and FF heuristics.

(3)Focused iterative-broadening search. This is a simple search algorithm that does not use heuristic estimators, and instead reduces the vast set of search possibilities by focusing on a limited operator set derived from the causal graph. It is an experimental algorithm; in the future, a further develops the basic idea of this algorithm into a more robust method.

Fast Downward uses causal graph heuristic. The main idea of causal graph heuristic is to compute $h^a_{cg}(s)$, which is the transition cost form state s to the goal s_*, and if s is the initial state, $h^a_{cg}(s)$ is the causal graph heuristic distance of the overall task. The function $h^a_{cg}(s)$ is define as

$$h^a_{cg}(s) = \sum_{v \in s_*} cost_v(s(v), s_*(v)) \tag{6}$$

In Equation (6), $cost_v(s(v), s_*(v))$ represents the estimated cost of changing variable v from a value in state s to another value in goal s_*. Equation (6) shows that the causal graph heuristic of a state is the sum of heuristic costs of all the variables in the goal, so it is an additive heuristic. We call $h^a_{cg}(s)$ as causal graph additive heuristic.

For a variable v, $cost_v(d, d')$ is computed mainly based on the slightly modified Dijkstra's algorithm by using causal graph and domain transition graphs as follows.

(1) If v has no predecessors in the causal graph, the value of $cost_v(d, d')$ is the length of the shortest path from d to d' in the domain transition graph of variable v, or ∞ if the path doesn't exist. This can be computed by Dijkstra's algorithm.

(2) Let V' be the set of the predecessors of variable v in the causal graph. If the transition of v from d to d' has a condition of assignment about a variable v' in V', then transition cost of v' from current state is computed and added into the transition cost of variable v.

(3) All high-level transitions have the basic cost 1.

4. Problems in the heuristic of Fast Downward

We take the transportation tasks shown in Fig.1 as an example. Four kinds of objects are included in the transportation domain: $locations$, $truck$, $cargo1$ and $cargo2$. The paths between different locations are two-way. Three operators are defined: $load$ cargo at one location, $unload$ cargo at one location and $move$ car from one location to another one.

In the planning task, obviously, f_t is a precondition of the action (load) which can move f_a or f_b to goal state. Therefore, even f_t at the goal state, but f_a or f_b is not at the goal state, the state of f_t at goal will be deleted when moving f_a or f_b to the goal state. Guaranteeing that f_a and f_b at the goal state first before moving f_t to goal state is a reasonable method. But the additive heuristic takes the sum of costs of all the variables in the goal as heuristic costs. There is a problem that the heuristic cost of f_t get smaller while the heuristic cost of f_a and f_b are unchanged after the action $< (f_t = P_4), (f_t = P_3) >$ being executed, and the heuristic will lead the search complete f_t first. As mentioned above, when moving f_a or f_b to goal state, the state of f_t at goal state will be deleted. As a result, if we only take the cost of f_a and f_b as the heuristic cost, the heuristic will lead f_a and f_b complete first. Then we use the cost of f_t as heuristic cost to take f_t to get the goal state. We will get a better plan of the planning task at this rate.

The heuristic of Fast Downward is the sum of heuristic cost of all the variables in the goal. These variables are regarded as independent each other. But in fact, there are helpful relations[8] between these variables, so the heuristic cost computed by Fast Downward is often bigger than the reality cost. We still take $Task1$ as an example. If we just want to get the heuristic cost of f_a and f_b , the additive heuristic cost from initial state to a goal is computed according to the following equation.

$$h(s) = cost_{fa}(P_1, P_3) + cost_{fb}(P_5, P_1) \tag{7}$$

The transition cost of f_a from P_1 to P_3 is 7, computed by Dijkstra algorithm in the domain transition graph of f_a, and the corresponding plan is:

$$\{move(P_4, P_3), move(P3, P2), \ldots, move(P2, P3), unload\}$$

Then we can compute the transition cost of f_a from P_5 to P_1. The cost of this step is 7, and the corresponding plan is:

$$\{move(P_4, P_5), load, \ldots, move(P_2, P_1), unload\}$$

Finally, we can get the heuristic cost of f_a and f_b which is $7 + 7 = 14$. But the optimal plan is:

$$\{move(P_4, P_5), load, \ldots, move(P_2, P_3), unload\}$$

The heuristic cost of optimal plan is 11 rather than 14, and the reason for the error of the heuristic: f_t come to P_1 with f_b having arrived at goal state. At this time, the precondition $(f_t = P_1)$ of the action which can take f_a to goal state becomes true. Therefore, there is less heuristic cost for f_a to go to goal state. It means that f_b has helpful relation on f_a.

The definitions and propositions of dependency relations are listed as follows.

DEFINITION 1. (dependency relation) Given a planning task $Q = (F, A, I, G)$, variables $x, z \in G$, action $a \in A$, if x is an effective variable of a, and z is a precondition variable of a. We call x has dependency relation on z. Signed as $x \to z$.

PROPOSITION 1.

$$\forall a \in A, \forall x, z \in G,$$
$$(x \in add(a))\&(z \in pre(a)) \Rightarrow x \to z$$

DEFINITION 2. (helpful relation) Given a planning task $Q = (F, A, I, G)$, variable $x, z \in G, c \in F$, actions $a, b \in A$, if the action a makes the variable z come to goal state, and action b makes the variable x come to goal state, and x is an effective variable of a, z is a precondition variable of a, and variable c is an effective variable of action a, as well as a precondition variable of b, and c has the same value in the two states. We call variable x has helpful relation on z. Signed as $x \to z$.

PROPOSITION 2.

$$\begin{aligned}\forall a, b \in A, \forall x, z \in G, c \in F,\\ (c \in add(a) = c \in pre(b))\&\\ (z \in add(a))\&\\ (x \in add(b)) \Rightarrow x \to z\end{aligned}$$

DEFINITION 3. (helpful value) According to definition 2, the helpful value of $x \to z$ is the cost of variable c transition from the state which the action a not be executed to the state which action a executed.

DEFINITION 4. (deep relation) Given a planning task $Q = (F, A, I, G)$, variables $x, z \in G, y \in F$, actions $a, b \in A$. And x is an effective variable of a, z is a precondition variable of b, and y is a precondition variable of a, as well as an effective variable of b, and y has the same value in the two state. We call variable x has deep relation of z, Signed as $x \to z$.

PROPOSITION 3.

$$\begin{aligned}\forall a, b \in A, \forall x, z \in G, y \in F,\\ (y \in pre(a) = y \in add(b)))\&\\ (x \in add(a))\&\\ (z \in pre(b)) \Rightarrow x \to z\end{aligned}$$

PROPOSITION 4.

$$\forall x, y, z \in G,$$
$$(x \to y)\&(y \to z) \Rightarrow x \to z$$

Obviously, one could get the proposition due to the definition 4.

DEFINITION 5. (dependency relation tree) Given a planning task $Q = (F, A, I, G)$, the tree of dependency relations of Q is defined as one or more trees $(t_1, t_2, \ldots, t_n)$. The set of vertex of the tree is the set of all the variables of goal. For example, if there has $x \to z$, take x as the child node, z as parent node. Constitute all the dependency relations and deep relations of goal, we will get $(t_1, t_2, \ldots, t_n)$.

DEFINITION 6. (helpful relations graph) Given a planning task $Q = (F, A, I, G)$, the helpful relations graph of Q is defined as a digraph whose vertex are the variables of a goal. There will be an arc from x to z, when $x \to z$, and the value of the arc is the helpful value of $x \to z$.

5. Heuristic based on subgoal ordering and helpful value

According to the analyses of the existing problems on the heuristic function of Fast Downward as well as the definition of dependency relations between the variables of goal and helpful value, We improve the heuristic function of Fast Downward from two aspects: first, ordering the variables of goal in order to get a reasonable sequence of variables to complete. Second, extracting the helpful value in the goal to get a more accurate heuristic cost.

5.1. Subgoal ordering

First, find all the variables of a goal which do not depend on any other variables as definition 1 and definition 4 form a subgoal. Then delete the subgoal from the goal and finish the process as mentioned above to get the next subgoal, until the goal is empty. The result of the algorithm is a subgoal sequence $(SG_1, SG_2, \ldots, SG_n)$. The algorithm is described as follows.

Step 1. Find all the dependency relations and deep relations in the planning task. Then do the work as definition 5, we will get one or more tree $(t_1, t_2, \ldots, t_n)$.

Step 2. Obviously, the leaves of the tree are the variables which should be completed first. So take all the leaves to form a subgoal (SG).

Step 3. Delete all the leaves from the tree. There will be many new leaves. After the work mentioned above, we will get the second subgoal (SG).

Step 4. Repeat the three steps mentioned above, until the tree is empty. Finally, we will get the subgoal sequence $(SG_1, SG_2, \ldots, SG_n)$.

The pseudocode of subgoal ordering algorithm is shown in Algorithm 1.

Algorithm 1 Subgoal ordering (Q).

InputA planning problem $Q = (F, A, I, G)$.
OutputSubgoals sequence $(SG_1, SG_2, \ldots, SG_n)$.
1. vector$< pair < int, int >>$ pair;
2. while $G \neq \phi$ do

3. choose the first element $e \in G$;
4. find $a \in A$ and $e \in add(a)$;
5. for $i \leftarrow 1$ to $pre(a).size()$ do;
6. if $(pre(a)[i] \in G$ or $pre(a)[i] \in pair.first)$ then
7. pair$\leftarrow (e \rightarrow pre(a)[i])$ or $(e \rightarrow pair.second)$;
8. else goto 4;
9. move e from G;
10. class node int n;vector$< node* > next$; $tree$;
11. $N \leftarrow 1$;
12. while $pair \neq \phi$ do
13. New $tree\ t_N$;
14. $N \leftarrow N + 1$;
15. while(Find $(e \leftarrow i) \in pair \cap e \notin t_N$)
16. $t_N \leftarrow (e \rightarrow i)$;
17. goto 15;
18. else goto 13;
19. while$(t_1, t_2, \ldots, t_n \neq \phi)$ do
20. $SG_i \leftarrow$all the leafs of the tree;
21. Delete the leafs from the tree;
22. Retun $(SG_1, SG_2, \ldots, SG_n)$.

5.2. Computing helpful value

We extract the helpful value in the goal to get a more accurate heuristic cost. There is a problem we have to concern that one variable may has helpful relation on several variables. However, only one of the variables can use the helpful relation. For example, let us assume that there is another variable $cargo3$ in P_1 to P_2. In this case, $cargo2$ has helpful relations $(f_t = P_1)$ on $cargo1$ and $cargo3$, but only one of them can use the helpful relation. If $cargo1$ comes to goal state, the helpful relation $(f_t = P_1)$ will be vanished, and the $cargo3$ cant use the helpful relation in this time. We use the longest path to compute helpful value in order to avoid this problem. The algorithm is described as follows.

Step 1. Find all the helpful relations of the planning task as definition 2, and compute the helpful value of each helpful relation as definition 3 to get the helpful relations graph as the definition 6.

Step 2. Compute the longest path in the helpful relation graph, and its length is the helpful value of all the helpful relations in the longest path.

Step 3. Delete the longest path from the helpful relation graph. Repeat the work above, until there is no arc in the helpful relation graph. Add all the helpful value. Finally, we will get sum of the helpful value of the goal (HV).

The pseudocode of helpful value algorithm is shown in Algorithm 2.

Algorithm 2 Helpful Value(Q, SG).

Input:A planning problem $Q = (F, A, I, G)$ and SG.
Output:helpful value.
1. $HV \leftarrow 0$;
2. graph GH;
3. for $i \leftarrow 1$ to $SG.size()$ do
4. if $sg[i] \neq G[i]$
5. find $a \in A$ and $sg[i] \in add(a)$ and $add(a)[i] = G[i]$;
6. for $j \leftarrow 1$ to $SG.size()$ do
7. if$(\exists b, f, (sg[j] \in add(b) \& f \in add(a) \& f \in pre(b)))$
8. $GH \leftarrow (sg[i], sg[j])$ and the length of the arc is DTG$(f, add(a)[f])$;
9. while$(GH \neq \phi)$
10. $HV+=$the length of longest path in GH;
11. Delete the longest path from GH;
12. return HV.

5.3. System architecture

Reading the planning task, we will get the subgoal sequence $(SG_1, SG_2, \ldots, SG_n)$ by the algorithm of ordering the variables of goal. Take the heuristic function to compute the heuristic cost of the first subgoal. If the heuristic cost is equal to 0, take the next subgoal to compute heuristic cost. If not, then compute the helpful value of the subgoal by the algorithm of computing helpful value. Extract the helpful value from heuristic cost to get a more precise heuristic cost, and return the heuristic cost to planner. Repeat the work above, until the subgoal sequence is empty. The system architecture of HBSH planner is shown in Fig.5.

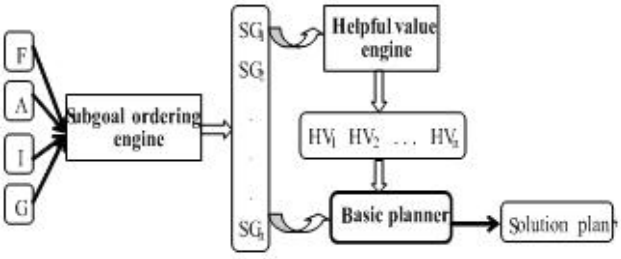

Figure 5 System architecture of HBSH planner.

6. Experimental Results

To compare with the Fast Downward planner, the algorithm proposed in this paper is complemented by C++. We compare the performance of the two algorithms using questions from the IPC 2008. The experimental platform is: Ubuntu-10(Linux kernel 2.6.32), the compiler is g++4.1.2, the memory is 3GB, the CPU is Intel Pentium Dual T3400 2.16GHz. By the rule of the IPC the schedule time is 30 minutes (1800s) for every domain problem. For convenience, our algorithm singed as HBSH (Heuristic Based on Subgoal ordering and Helpful value), and the Fast Downward singed as F-D.

Table.1 shows the length of planning answers for HBSH and F-D in Woodworking and Pegsol domains form IPC 2008.

Table 1 Plan length between HBSH and F-D

Woodworking			Pegsol		
Problem	F-D	HBSH	Problem	F-D	HBSH
1	5	5	1	6	5
2	6	6	2	10	9
3	8	8	3	13	11
4	13	13	4	15	11
5	15	13	5	14	12
6	8	8	6	18	12
7	50	48	7	16	16
8	25	25	8	16	15
9	24	22	9	21	19
10	34	32	10	24	20
11	36	36	11	25	20
12	45	42	12	23	21
13	63	57	13	28	24
14	72	72	14	27	22
15	116	107	15	30	26
16	147	153	16	35	25
17	89	76	17	62	67
18	129	118	18	86	82
19	168	173	19	122	107
20	153	147	20	75	77

In Table 1, it shows that the average length of plan for Fast Downward in the Woodworking domain is 60.3, while the LSH is 56.45. In addition, the average length of plan for Fast Downward in the Pegsol domain is 33.3, the LSH is 31.05. The experimental results proved that the algorithm proposed improves the performance in shorting the length of planning answers and obtains better plan.

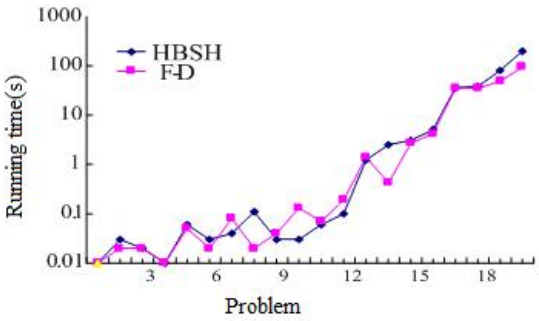

Figure 6 Running time of woodworking.

Fig.3 to Fig.6 shows that the running time of HBSH and Fast Downward in Woodworking domain is closed and the same in the former 10 problems in Pegsol domain. However, the running time of HBSH is longer than Fast

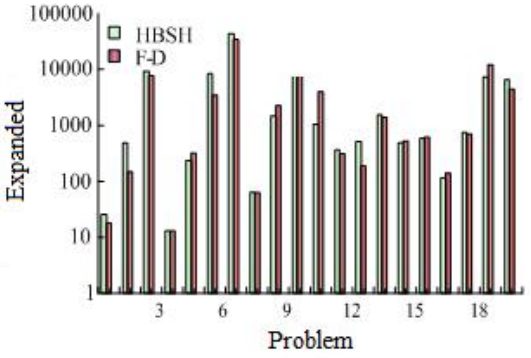

Figure 7 Expanded nodes of woodworking.

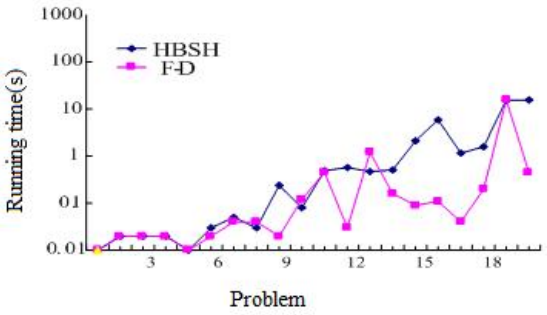

Figure 8 Running time of Pegsol.

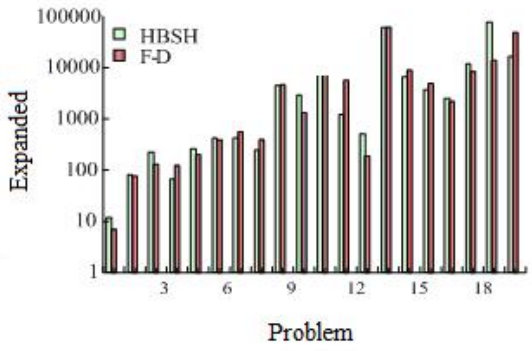

Figure 9 Expanded nodes of Pegsol.

Downward in the last 10 problems, which might because the calculate amount grows greatly when the complexity of our algorithm has being to a significant extent. How to improve the algorithm efficiency and accelerate the searching time is the future work. The performances of the two algorithms are similar from the angle of expanded nodes.

7. Conclusions

In recent years, more and more researchers begin to pay attention to the heuristic search. The accuracy of the heuristic function in the heuristic searching method will determine the performance of a planner, as a result, after analyzing the heuristic function of the Fast Downward planner we propose a heuristic search method based on subgoal ordering and helpful value. The experiments results prove that the proposed algorithm is effective. In the actual planning problems, the dependency relations in the whole planning task are more than the above two points. How to extract the dependency relations in the planning task, and then use these dependency relations to design a more efficient heuristic function to improve our efficiency in solving planning task, is a problem worth us to continue to study. Research of dependency relations in planning task and to design a more efficient heuristic function are our future work.

Acknowledgement

The authors acknowledge the financial support of NSFC under project No. 61142011, the Key Project of Chinese Ministry of Education under project No. 210183, and the Program for New Century Excellent Talents in University of China under project No. NCET-11-1085.

References

[1] B. Bonet, H. Geffner, Planning as heuristic search, Artif. Intell., **129** (2001).
[2] J. Hoffmann and B. Nebel, The FF planning system: Fast plan generation through heuristic search, J. Artif. Intell. Res., **14** (2001).
[3] M. Helmert, The Fast Downward planning system, J. Artif. Intell. Res., **26** (2006).
[4] M. Helmert, A planning heuristic based on causal graph analysis, Proc. the International Conference on Automated Planning and Scheduling, 161 (2004).
[5] E. Keyder, H. Geffner, Set-Additive and TSP Heuristic for Planning with Action Costs and Soft Goals, Proc. the International Conference on Automated Planning and Scheduling, 140 (2007).
[6] R. S. Liang, Y. F. Jiang, R. Bian, Admissible Subgoal Ordering for Automated Planning, J. Software, **22** (2011).
[7] F. Maria, L. Derek, PDDL: An extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res., **20** (2003).
[8] X. J. Wu, Y. F. Jiang, Y. B. Ling, Research on Relations of Effect of Action for STRIPS Domain, J. Software, **18** (2007).

Weisheng Li graduated from School of Electronics & Mechanical Engineering at Xidian University of China in July 1997. He received his M.S. degree and Ph.D. degree from School of Electronics & Mechanical Engineering and School of Computer Science & Technology at Xidian University of China in July 2000 and July 2004, respectively. Currently he is a professor of Chongqing University of Posts and Telecommunications of China. His research focuses on intelligent information processing and pattern recognition.

Peng Tu is a postgraduate of Chongqing University of Posts and Telecommunications. His research focuses on intelligent planning.

Junqing Liu is a associate professor of China Three Gorges University. His research focuses on multimedia information processing.