CS182: Assignment 5
Planning

Electronic Submission due Sunday, December 9 at 11:59 pm.
Printouts due at the beginning of lecture, Monday, December 10 at 2:30 pm

1 Introduction

In this assignment, you will compare the behavior of two different types of planning systems, namely SGP (a Graphplan-type planner) and UCPOP (a partial-order or causal-link planner), on problems in two provided domains as well as two new domains. You will also learn how to encode planning problems in the representation language accepted by UCPOP as well as a standard STRIPS-like language called PDDL (the Planning Domain Definition Language), which is the input representation language accepted by SGP. We provide you with the definitions of the "blocks-world" and "tire-world" domains in a language that UCPOP understands as well as in PDDL format. You will have to define two new domains and problems in the languages accepted by UCPOP and PDDL. UCPOP must perform better than SGP on one domain; and on the other domain, SGP must outperform UCPOP.

- *Tasks that you must perform for assignment 5 are bulleted and in italics, like this, throughout this document.*

2 Provided Code

Within `~lib182/asst5` you will find some files and two subdirectories containing the two planning systems. Copy all this code to your own directory. To do this, type the following commands: `cd ~lib182/asst5`. Then, type `tar cf - * | (cd ~/<insert your directory name here>; tar xvf -)`. Finally, use `cd` to change back to your directory.

The files you need to modify for this assignment are all in the `asst5/` directory. They are called `sgp-better.lisp`, `sgp-better.pddl`, `ucpop-better.lisp`, `ucpop-better.pddl`, and `answers.txt`.

2.1 Running Sensory Graphplan (SGP)

Sensory Graphplan is a Lisp implementation of the Graphplan algorithm covered in class. It also incorporates some extensions to the basic capabilities of Graphplan. We have provided you with a stripped-down version of Sensory Graphplan. Not all the features are available, but everything necessary for this assignment is. To load the planner, start Lisp from your `asst5` directory and type `(load ‘‘asst5.system’’)` and then `(mk)`. Change to the "gp" package by typing `(in-package domains)`. In SGP, you must load the domains manually. To load a PDDL file, for example `blocks-world.pddl`, type `(load-domains ‘‘blocks-world’’)`. To solve a planning problem, you can type, for example, `(gp:plan 'sussman-anomaly)`. Note that the files used by SGP have a `.pddl` extension.

2.2 Running UCPOP

If you are within the Lisp environment and have already loaded `asst5.system` and run `(mk)`, you can change to the "ucpop" package by typing `(in-package ucpop)`. In UCPOP, the domains are loaded automatically by the `(mk)` command. To solve a planning problem type, for example, `(bf-control 'fixit)`. Note that the files used by UCPOP have a `.lisp` extension.

3 The Syntax of the Domain Encoding Languages

3.1 Domain Encoding in PDDL—the Representation Language Accepted by SGP

The `sgp` subdirectory contains a file called `pddl.ps` which provides complete documentation for PDDL. Here are a few tips on using PDDL to get you started. The process of defining a domain is best illustrated with an example:

```
(define (domain blocks-world-domain)
  (:requirements :strips :equality :conditional-effects)

  (:constants Table)
  (:predicates (on ?x ?y)
               (clear ?x)
               (block ?b))
  ;; Define step for placing one block on another.
  (:action puton
```

```
:parameters (?X ?Y ?Z)
:precondition (and (on ?X ?Z) (clear ?X) (clear ?Y)
                   (not (= ?Y ?Z)) (not (= ?X ?Z))
                   (not (= ?X ?Y)) (not (= ?X Table)))
:effect
(and (on ?X ?Y) (not (on ?X ?Z))
      (when (not (= ?Z Table)) (clear ?Z))
      (when (not (= ?Y Table)) (not (clear ?Y))))))
```

There are some important things to note about this domain definition. All variable names must begin with a question mark (?). This example uses conditional effects, which you will hear about in lecture, but won't need for this assignment. While the semantics of each predicate are not explicitly given in this example, which is supplied along with the PDDL documentation, you must give us proper documentation explaining the semantics of each predicate you use.

Defining a problem is similar. Here is an example:

```
(define (problem sussman-anomaly)
  (:domain blocks-world-domain)
  (:objects A B C)
  (:init (block A) (block B) (block C) (block Table)
         (on C A) (on A Table) (on B Table) (clear C)
         (clear B) (clear Table))
  (:goal (and (on B C) (on A B))))
```

3.2 Domain Encoding in the Representation Language Accepted by UCPOP

The representation language accepted by UCPOP is similar to PDDL. Again, the process of defining a domain is best illustrated with an example. We provide the same example as above:

```
(in-package "UCPOP")

(define (domain blocks-world)
    ;; Define step for placing one block on another.
    (:operator puton
      :parameters (?X ?Y ?Z)
      :precondition (and (on ?X ?Z) (clear ?X) (clear ?Y)
 (neq ?Y ?Z) (neq ?X ?Z)
```

```
 (neq ?X ?Y) (neq ?X Table))
      :effect
      (and (on ?X ?Y) (not (on ?X ?Z))
   (when (neq ?Z Table) (clear ?Z))
   (when (neq ?Y Table) (not (clear ?Y))))))
```

There are a few ways in which this representation differs from PDDL. The language accepted by UCPOP doesn't have requirements, constants, or predicates fields. Also, actions are called operators and `neq` is used to mean "not-equal." Here is how we define the Sussman anomaly problem in this language:

```
(define (problem sussman-anomaly)
  :domain 'blocks-world
  :inits ((block A) (block B) (block C) (block Table)
  (on C A) (on A Table) (on B Table)
  (clear C) (clear B) (clear Table))
  :goal (and (on B C) (on A B)))
```

Unlike PDDL, problem descriptions in this language do not include objects, and the domain name must be quoted. Also, the placement of parentheses is different, and `init` becomes `inits` before the list of predicates that are true initially.

4 UCPOP vs. Graphplan on the Provided Domains

For the first part of the assignment, you will compare the performance of UCPOP to SGP on one problem each in the blocks-world and tire-world domains. First, read the files `blocks-world.*` and `tire-world.*`, understanding the domain and problem definitions. Specifically, focus on the problems called `tower-invert6` in the blocks-world domain and `fixit` in the tire-world domain. Now, run SGP and UCPOP on both problems and observe the performance of both planning systems on each problem. Time the planning systems on the problems by using the Lisp `time` function. To use this command for SGP, for example, type `(time (gp:plan 'sussman-anomaly))`. The time to be concerned with is CPU time (total)—user. (Note: if you don't see Graphplan performing significantly better on the tire-world problem and UCPOP performing significantly better on the blocks-world problem, you probably have a mistake somewhere). Record

the time taken by each planner and the other required statistics in the file `answers.txt`. Then, answer the analysis question, which asks you to formulate a hypothesis regarding the characteristics of each domain that make it better suited for a particular planning algorithm.

For this part of the assignment, you must:

- *Run UCPOP and SGP on the two provided problems and measure the times taken by the two planners to find plans.*
- *In the file* `answers.txt`*, record the required statistics and formulate a hypothesis characterizing the features of each domain that make it better suited for a particular planning algorithm.*

5 UCPOP vs. Graphlan on Your Domains

Based on your hypothesis from the previous part of this assignment, you will now encode two new domains: UCPOP must outperform SGP on one domain, and on the second domain, SGP must outperform UCPOP. The two new domains that you come up with must have a combined total of at least 12 operators and objects. For example, one domain can have 3 operators and 3 objects, and the other domain can have 3 operators and 3 objects. Also, your domains must be substantially different from the blocks-world and tire-world domains. If you are having trouble coming up with domains, think about activities in your daily life such as going to class or going on vacation.

You must define each domain and problem definition in both PDDL and the input representation language accepted by UCPOP. Place the domain and problem definitions in the following files:

- `sgp-better.pddl`: PDDL definiton for domain in which SGP outperforms UCPOP.
- `sgp-better.lisp`: UCPOP input represenation for domain in which SGP outperforms UCPOP.
- `ucpop-better.pddl`: PDDL definition for domain in which UCPOP outperforms SGP.
- `ucpop-better.lisp`: UCPOP input represenation for domain in which UCPOP outperforms SGP.

Note that the initial skeleton code in the `.lisp` files has been commented out—you will need to uncomment it. Be sure to document your two new domains appropriately. State clearly the meaning of the domain operators you use and the initial and goal states of the problems.

Time both of the planning systems on your two domains using the Lisp `time` function. You should interactively test your new domains just like you testested the provided domains. Start Lisp from your `asst5` directory and type `(load ‘‘asst5.system’’)` and then `(mk)`.

Change to the "gp" package by typing `(in-package domains)`. To load a PDDL file, for example `sgp-better.pddl`, type `(load-domains ‘‘sgp-better’’)`. To solve a planning problem, you can type, for example, `(gp:plan ’<insert your problem name here>)`.

Change to the "ucpop" package by typing `(in-package ucpop)`. To solve a planning problem type, for example, `(bf-control ’<insert your problem name here>)`.

You should time both of the planners on your new domains using the Lisp `time` function. The time to be concerned with is CPU time (total)—user. Record the times and other required statistics in the file `answers.txt`.

If the data that you collect does not support your hypothesis from Section 4, revise your hypothesis after analyzing the data more carefully. Then, use your new hypothsis to create two new domains. *Please, only submit your final hypotheis and two domains, not the intermediate ones you aren't satisfied with.*

To summarize, you must:

- *Write your two domain and problem definitions in the files* `sgp-better.pddl`, `sgp-better.lisp`, `ucpop-better.pddl`, *and* `ucpop-better.lisp`. *Run UCPOP and SGP on the two problems and measure the times taken by the two planners to find plans.*
- *Record your timing results and the other required statistics in the file* `answers.txt`.
- *Continue to revise your hypothesis and encode two domains until your data supports the hypothesis you drew in Section 4.*

6 Finishing Up

After you have tested both of your new domains interactively, you should generate output files automatically using `make`. You must first place your

problem names in the `Makefiles` located in the `asst5/` directory. Insert the names of your problems where you see the text "`<insert problem name here>`." In order to generate the output files, type `make sgp-better.out` and `make ucpop-better.out`. Before printing and submitting, you should examine the output generated in the `.out` files.

When you are done, print out and submit your solutions using `make print` and `make submit` as usual. Run these commands from the `asst5/` directory.

7 Question for Thought

Only spend extra time thinking about this question if you have time. You will not receive any extra credit for this question—it's just for fun.

> In class, we talked about encoding planning as a SAT problem. How would you encode SAT as a planning problem?