A Beginner’s Introduction to Heuristic Search Planning

3. A Simple Heuristic

Malte Helmert    Gabriele Röger

AAAI 2015 Tutorial

January 25, 2015
Hamming Distance Heuristic
Hamming Distance Heuristic

- **Heuristic** $h : S \rightarrow \mathbb{R}^+_0$
  
  estimates cost to reach a goal from a state

- Fast Downward can easily be extended with new heuristics
Hamming Distance Heuristic

- **Heuristic** $h : S \rightarrow \mathbb{R}^+_0$
  estimates cost to reach a goal from a state
- Fast Downward can easily be extended with new heuristics
- Simple example: **Hamming distance heuristic** counts number of variables that do not have required goal value
  - $h^H(\{A \mapsto a, B \mapsto b, C \mapsto c\}) = 1$
  for task with goal $\{A \mapsto a, B \mapsto b'\}$

▶ hands-on
Hands-On

$ cd hands-on/heuristic-implementation
$ ls
$ ./build_all.sh
make: Nothing to be done for ‘default’.
make: Nothing to be done for ‘default’.
make: ‘validate’ is up to date. $ cd search ...

Hamming Distance Heuristic in Fast Downward
Hamming Distance Heuristic in Fast Downward

Hands-On

$ cd hands-on/heuristic-implementation
$ ls
$ ./build_all.sh
make: Nothing to be done for ‘default’.
make: Nothing to be done for ‘default’.
make: ‘validate’ is up to date. $ cd search
...

Derive from class Heuristic

Hands-On

$ less heuristic.h
...

Implement **constructor and destructor**

```cpp
HammingHeuristic::HammingHeuristic(const Options &opts) : Heuristic(opts) {}

HammingHeuristic::~HammingHeuristic() {}
Possibly implement `initialize`

```cpp
void HammingHeuristic::initialize() {
    cout << "Initializing Hamming distance "
    << "heuristic..." << endl;
}
```

→ executed once before the first call of a heuristic evaluation
Implement `compute_heuristic`

```cpp
int HammingHeuristic::compute_heuristic(
    const GlobalState &state) {
    return 0; // TODO
}
```
Accessing the task

- Variables and variable values:
  - represented as ints
  - `vector<int> g_variable_domain` in `globals.h`
  - variable $i$ has domain $\{0, \ldots, g_{variable\_domain}[i] - 1\}$. 

Example:

```cpp
int var = g_goal[0].first;  
int value = g_goal[0].second;  
int value = state[var];  
```
Accessing the task

- **Variables and variable values:**
  - represented as `ints`
  - `vector<int> g_variable_domain` in `globals.h`
  - variable $i$ has domain $\{0, \ldots, g_{variable\_domain}[i]-1\}$.

- **Goal:** `vector<pair<int, int>> g_goal` in `globals.h`
  - e.g. `int var = g_goal[0].first;`
  - `int value = g_goal[0].second;`
Accessing the task

- **Variables and variable values:**
  - represented as ints
  - `vector<int> g_variable_domain` in `globals.h`
  - variable $i$ has domain $\{0, \ldots, g_{variable\_domain}[i] - 1\}$.
- **Goal:** `vector<pair<int, int>> g_goal` in `globals.h`
  - e.g. `int var = g_goal[0].first;`
  - `int value = g_goal[0].second;`
- **State:** `class GlobalState`
  - internal representation complicated and optimized for memory-efficiency
  - variable values can easily be accessed with `operator[]`, e.g. `int value = state[var];`
Accessing the task

- **Variables and variable values:**
  - represented as ints
  - `vector<int> g_variable_domain` in `globals.h`
  - variable $i$ has domain $\{0, \ldots, g_{\text{variable\_domain}}[i] - 1\}$.

- **Goal:** `vector<pair<int, int>> g_goal` in `globals.h`
  - e.g. `int var = g_goal[0].first;`
  - `int value = g_goal[0].second;`

- **State:** `class GlobalState`
  - internal representation complicated and optimized for memory-efficiency
  - variable values can easily be accessed with `operator[]`,
  - e.g. `int value = state[var];`

- **Operators:** `class GlobalOperator`
  - `vector<GlobalOperator> g_operators` in `globals.h`
  - cf. `global_operator.h`
Integration of Heuristic into Planner

- Make the heuristic known to the search command parser by extending the heuristic implementation file:

```cpp
static Heuristic * _parse(OptionParser & parser) {
    Heuristic::add_options_to_parser(parser);
    Options opts = parser.parse();
    return new HammingHeuristic(opts);
}

static Plugin<Heuristic> _plugin("hamming", _parse);
```
Integration of Heuristic into Planner

- Make the heuristic known to the search command parser by extending the heuristic implementation file:

  ```cpp
  static Heuristic * _parse(OptionParser & parser) {
    Heuristic::add_options_to_parser(parser);
    Options opts = parser.parse();
    return new HammingHeuristic(opts);
  }

  static Plugin<Heuristic> _plugin("hamming", _parse);
  ```

- Add entry with header filename to Makefile

  ```bash
  Hands-On
  $ less Makefile
  ...  ```
Files `hamming_distance_heuristic.h` and `hamming_distance_heuristic.cc` contain stub for Hamming distance heuristic.

1. Integrate the heuristic into the planner
2. Finish the heuristic implementation

Test your implementation with

```
$ make
$ cd ..
$ ./fast-downward.py \
  ../ipc/blocks/probBLOCKS-8-0.pddl \ 
  --search "astar(hamming())"
$ cd -
```
Outlook
After the break:

- Five Families of Heuristics
- Abstraction Heuristics and Pattern Databases
- Delete Relaxation and Landmarks
- Going Further
  - What else happens in heuristic planning?
  - What else happens in classical planning?
  - What else happens in planning?