Heuristic Search in Fluid Construction Grammar

Version of June 11, 2021

Insight Laboratories vof
Avenue du Daim 51, 1170 Brussels, Belgium
BTW BE 0755 544 975 – RPR Brussel
www.insight-labs.eu – info@insight-labs.eu

Introduction

This report documents the use of heuristic search strategies in Fluid Construction Grammar. It accompanies the new construction inventory processor `:heuristic-search` and its associated methods in the file `heuristic-search.lisp`.

Installation and use

In order to use the heuristic search architecture, the file `heuristic-search.lisp` should be moved to Babel's folder `systems/fcg/construction-inventory-processor/`, and a reference to the file should be added to `systems/fcg/fcg.asd`. In the asd file, `(:file "heuristic-search")` should be added under the module `construction-inventory-processor`, just after the reference to `(:file "construction-inventory-processor")`.

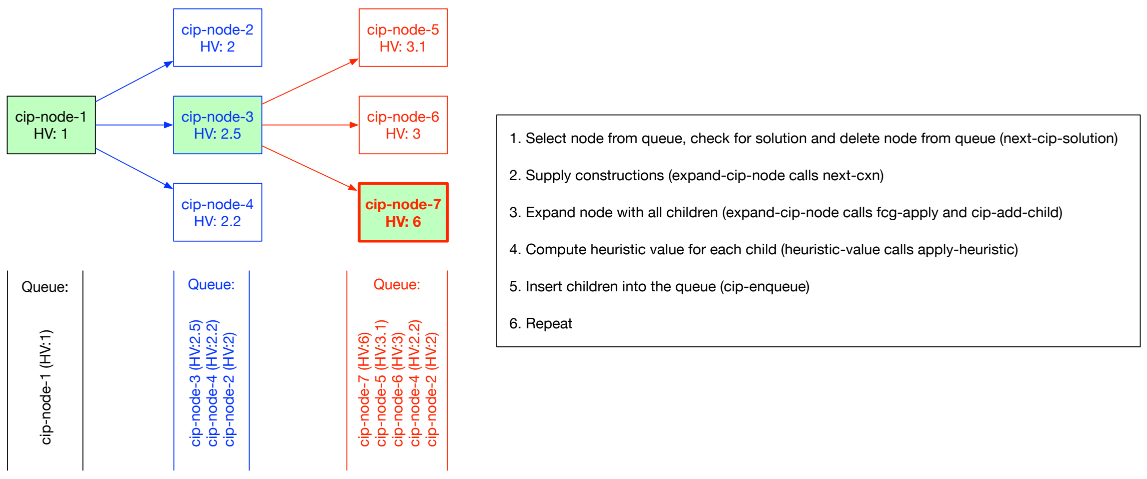
The code was designed for compatibility with the current version of babel-core (`https://gitlab.ai.vub.ac.be/ehai/babel-core` – commit `b38617597388730ce2c4dd8b128f14a7225fe9e2` and has been tested using Clozure Common Lisp version 1.12.

A demonstration of the possibilities is included in the file `demo.lisp`.

General architecture

The heuristic search method works as follows:

1. We start in the method `next-cip-solution`, specialised on the `construction-inventory-processor-heuristic-search` class . The first node in the queue of the construction-inventory-processor (cip) is selected and removed from the queue.
2. Then, the `expand-cip-node` method specialised on the mode `:full-expansion` is called. The selected node is expanded at once with all applicable constructions returned by the `next-cxn` method of the chosen construction supplier and marked as `fully-expanded`. The `next-cxn` method is thus only called once per node and the node is immediately fully expanded.
3. For each new expansion, a heuristic value is calculated. This heuristic value is computed by calling the `apply-heuristic` methods for all modes specified in the configuration of the construction inventory under the key `:heuristics`. The values returned by these methods are aggregated using the `heuristic-value` method that specialises on the mode specified in the configuration of the construction inventory under the key `:heuristic-value-mode`.
4. The children of this node are inserted into the queue by the `cip-enqueue` method according to the search algorithm specified in the configuration of the construction inventory under the key `:search-algorithm`. Uninformed search algorithms such as depth-first, breadth-first and random search insert the node at a particular position in



1. Select node from queue, check for solution and delete node from queue (next-cip-solution)
2. Supply constructions (expand-cip-node calls next-cxn)
3. Expand node with all children (expand-cip-node calls fcg-apply and cip-add-child)
4. Compute heuristic value for each child (heuristic-value calls apply-heuristic)
5. Insert children into the queue (cip-enqueue)
6. Repeat

Figure 1: A schematic representation of the heuristic search process in Fluid Construction Grammar.

the queue: at the front in the case of depth-first, at the end in the case of breadth-first, and at a random position in the case of random search. Informed search strategies such as best-first insert the node at the right place based on its heuristic value (higher is better).

A schematic representation of the heuristic search process in Fluid Construction Grammar is shown in Figure 1.

Using the heuristic search architecture

The following entries in the configuration of the construction inventory can be used to adapt the behaviour of the heuristic search architecture:

- `:construction-inventory-processor-mode` needs to be set to `:heuristic-search` to activate the new heuristic search architecture.
- `:node-expansion-mode` needs to be set to `:full-expansion` so that all child nodes are created at once for each node selected by `next-cip-solution`.
- `:cxn-supplier-mode` needs to be set to `:all-cxns`, `:cxn-sets`, `:hashed`, `:cxn-sets-hashed` or another construction supplier that returns for each node immediately all constructions that need to be considered.
- `:search-algorithm` regulates the insertion of new nodes in the queue. Possible values are `:best-first`, `:depth-first`, `:breadth-first` and `:random`.
- `:heuristics` takes as value all modes for which `apply-heuristic` needs to be called. Built-in modes are `:nr-of-applied-cxns`, `:cxn-sets`, `:nr-of-units-matched` and `:prefer-local-bindings`.
- `:heuristic-value-mode` combines the scores returned by all heuristics into a single heuristic value. By default, the mode `:sum-heuristics-and-parent` is used, which

sums the scores returned by the heuristics and adds them to the heuristic value of the parent of the node.

- `:parse-order` and `:production-order` are used to specify construction sets, just like before. A compatible cxn-supplier (e.g. `:cxn-sets` or `:cxn-sets-hashed`) and the heuristic `:cxn-sets` need to be specified as well in order to actually use the sets.
- `:parse-goal-tests` and `:production-goal-tests` are used just like before, but are even more important. They should be able to identify a solution, the heuristic search will do the rest.

Overview of classes and methods involved

The following classes and methods are involved in the heuristic search architecture:

- Class `construction-inventory-processor-heuristic-search`: subclass of `construction-inventory-processor` and hook for the heuristic search architecture. Does nothing special.
- Method `create-construction-inventory-processor` specialised on class `construction-inventory-processor-heuristic-search`: creates the construction-inventory-processor-heuristic-search.
- Method `next-cip-solution` specialised on class `construction-inventory-processor-heuristic-search`: has the same functionality as the default method, but uses configurations from the construction inventory with more intuitive names.
- Method `expand-cip-node` specialised on mode `:full-expansion`: expands a node with all possible expansions at once. Calls the `next-cxn` method of the construction supplier.
- Method `cip-enqueue` specialised on mode specified by `:search-algorithm` in the configuration of the construction-inventory implements the search algorithm by regulating where new nodes are inserted in the queue.
- Method `heuristic-value` specialised on mode specified by `:heuristic-value-mode` in the configuration of the construction-inventory regulates how the heuristic value of a new node is computed.
- Method `apply-heuristic` specialised on mode specified by `:heuristics` in the configuration of the construction-inventory implements a single heuristic. This method is called by the `heuristic-value` method for each different heuristic and should return a score.
- Classes `cxn-supplier`, `cxn-supplier-cxn-sets`, `cxn-supplier-hashed` and `cxn-supplier-cxn-sets-hashed` embody construction suppliers. Those that work with cxn-sets have an additional slot `cxn-sets`.

- Method `create-cxn-supplier` specialised on the construction supplier class initialises the construction supplier.
- Method `next-cxn` returns at once all constructions that are relevant for a node. In contrast to earlier, the `next-cxn` method is called only once for each node.

A note on construction sets

In the new architecture, construction sets are implemented somewhat differently as compared to the earlier architecture. The behaviour of construction sets is now handled by a combination of the construction supplier and the computation of the heuristic value:

- The construction supplier returns all (relevant) constructions of the current and all later sets. It does not return constructions of earlier sets. The built-in construction supplier modes `:cxn-sets` and `:cxn-sets-hashed` do this.
- A heuristic should be used that assigns a lower score to constructions that are of a later set. The score should reflect the distance (in terms of sets) between the construction that was applied to the parent of the node and the construction that is applied to the current node. The built-in `apply-heuristic` mode `:cxn-sets` does exactly this.