PA184 - Heuristic Search Methods

Lecture 9 – Software Libraries for Heuristics

- Implementation Strategies
- Software Class Libraries
- Other Software Toolkits

Learning outcomes:

- List the various strategies that exist to develop heuristic algorithms software.
- Identify advantages and disadvantages of code reuse for development of heuristic algorithms.
- Recognise the different types of frameworks that exist for the development of heuristic algorithms.
- Identify important analysis and visualisation aspects that should be considered when implementing heuristic algorithms.

Implementation Strategies

The implementation of a heuristic method involves developing software either: by writing code from scratch, by reusing software, or by employing software frameworks.

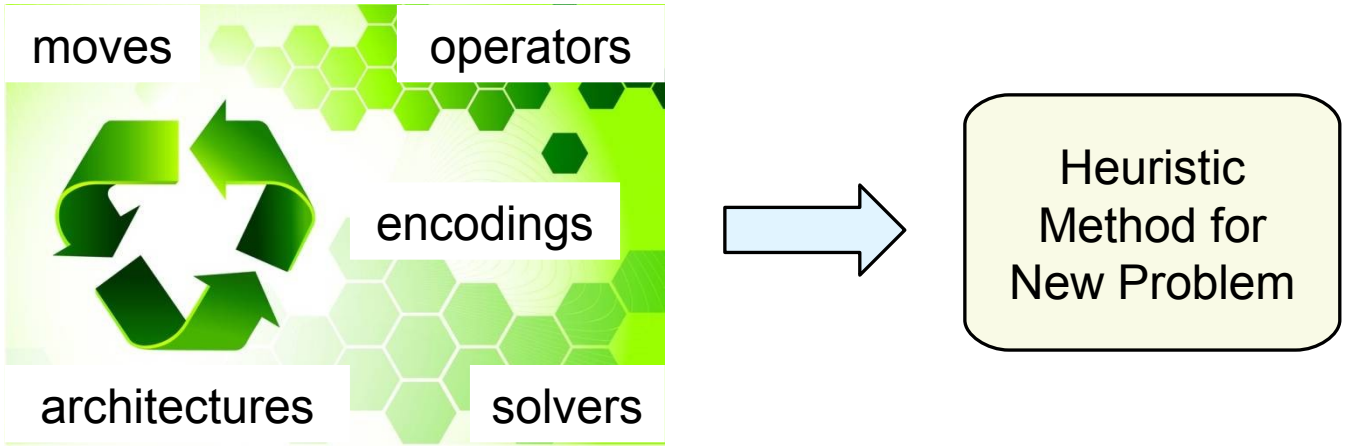
Reusing software: includes taking someone else's code and adapt it or using dedicated software libraries.

Software frameworks: provide full control on the 'invariant' parts of algorithms through the provision of algorithms components.

When selecting an implementation strategy, expertise of the developer and software engineering principles should be taken into consideration.

Talbi (2009) suggests a number of desirable criteria that software frameworks should satisfy (arguably all met in ParadisEO).

Since there is a great variety of optimisation problems, software reuse is crucial to develop heuristic methods at least as a first approach to assess the suitability of the algorithm to tackle the problem in hand.



In addition to faster development, libraries and frameworks facilitate the testing of different heuristic variants and their tuning. Software reuse often reduces the need to maintain complex data structures.

Mathematical programming frameworks are usually considered black boxes because usually no details of the algorithms are apparent.

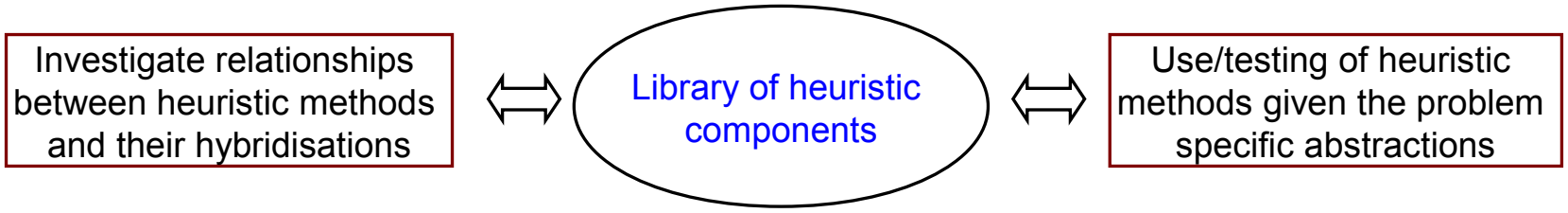
Software Class Libraries

Libraries of heuristic routines can help to rapid development of an application-specific heuristic method or solver.

There are usually 3 types of libraries (Voss and Woodruff, 2002):

- Callable routines
- Numerical libraries
- Component libraries

In general, a component library is software that permits to select, possibly adapt and combine appropriate modules from a large set of existing software components.



Localizer

Localizer was proposed by (Michel and Van Hentenryck, 1999) and (Michel and Van Hentenryck, 2000) as a modelling language for the implementation of local search methods.

It uses declarative expressions to define data structures for the problem being solved and imperative expressions to define starting and restarting states as well as transformations to those states.

It uses invariants as a tool to efficiently manage the incremental updates of iterative procedures like objective function computations.

It allows the definition of different neighbourhood exploration strategies such as best improvement, first improvement, random walk, etc.

Localizer is a predecessor of Comet, a constraint-based programming language and system. See: http://www.comet-online.org/

Example 9.1 Implementation of a *local improvement statement* in Localizer for the SAT problem.

```
Solve
Type:
    clause = record
        p : {int};
        n : {int};
        end;
Constant:
    m: int = ...;
    n: int = ...;
    cl: array[1..m] of clause = ...;
Variable:
    a: array[1..n] of boolean;
Invariant:
    nbtl : array[ i in 1..m] of int = sum(i in cl[i].p) a[j] + sum(j in cl[i].n) !a[j];
    nbClauseSat : int = sum(i in 1..m) (nbtl[i] >0);
Satisfiable:
    nbClauseSat = m;
Objective Function:
    maximize nbClauseSat;
Neighborhood:
    move  a[i] := !a[i]
    where i from {1..n}
    accept when improvement;
Start:
    forall(i in 1..n) a[i] := random({true,false});
Restart:
    forall(i in 1..n) a[i] := random({true,false});
```

move can be replaced by **first move**, **best move**, etc.
improvement can be replaced by **noDecrease**, etc.

iOpt Toolkit

iOpt was proposed by (Voudouris et al., 2001) at BTExact as a Java based environment providing declarative programming capabilities for specifying:

1. the problem modelling
2. the heuristic method
3. interactive visualisation
4. scheduling framework

Similar to Localizer, iOpt also uses invariants (one-way constraints) to support heuristic search because invariants represent an efficient way to evaluate incremental solutions changes.

The heuristic search framework (HSF) is part of the environment and provides a number of components so that the designer can build a complete algorithm.

Both single-solution and population-based heuristic methods can be specified using the HSF.

Example 9.2 Implementation of a *simple problem model* in iOpt.

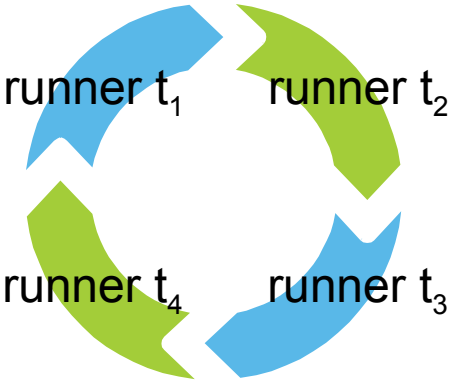
```
/*Create a simple problem model using invariants*/
Problem p = new Problem();
p.beginModelChanges();              // start changes to the problem model
RealVar x = new RealVar(10.0)       // create a real variable x and set its initial value to 10.0
p.addDecisionVar(x);                // set x to be a decision variable
p.addObjective(Inv.plus(x,5.0));    // add the term x+5.0 to the objective initially undefined
p.addConstraint(Inv.gt(x,5.0));     // set the constraint x > 5.0
p.endModelChanges();// end changes to the problem model

/*Change to the value of the decision variable*/
/*Similar operations are performed when local search is evaluating a move
   or the user modifies the solution through a GUI*/
p.beginValueChanges();              // start changes to the values of the decision variables
x.setValue(100.0);                  // set x to the new value of 100.00
p.endValueChanges();                // end changes to values of the decision variables
p.propagateValueChanges();          // the mark/sweep algorithm is updating the variables
p.saveValueChanges();               // we commit the changes, we may undo them instead
                                    // in case of constraint violations or inferior cost

/*Dynamic addition of decision variables/objective to the problem model*/
RealVar  y = new RealVar(15.0);
p.addDecisionVar(y);
p.addObjective(Inv.plus(y,10.0));   // the overall objective is now (x+5.0) + (y+10.0)
p.endModelChanges();
```

EasyLocal++

EasyLocal++ was proposed by (Di Gaspero and Schaerf, 2003) as an object-oriented framework in C++ for the development and analysis of local search algorithms. This framework is available from:

http://tabu.diegm.uniud.it/EasyLocal++/

EasyLocal++ allows the implementation of a token-ring search strategy in which a set of runners t_1, t_2,…,t_q are executed in a circular fashion with each runner starting from the best solution found by the previous one.



EasyAnalizer was proposed by (Di Gaspero, Roli and Schaerf, 2007) as an object-oriented framework for the experimental analysis of algorithms that integrates with EasyLocal++ in order to aid the understanding, study and tune stochastic local search algorithms.

EasyLocal++ incorporates 5 sets of classes:

1. **Data Classes:** input/output data, moves, search space states, etc.
2. **Helpers:** neighbourhood explorer, tabu list manager, output manager , weight handler, etc.
3. **Runners:** hill-climbing, simulated annealing, tabu search, etc.
4. **Solvers:** generate initial solutions, control the sequence of runners (tandem, multi-start, etc.)
5. **Testers:** users interface, algorithms analysis tools, etc.

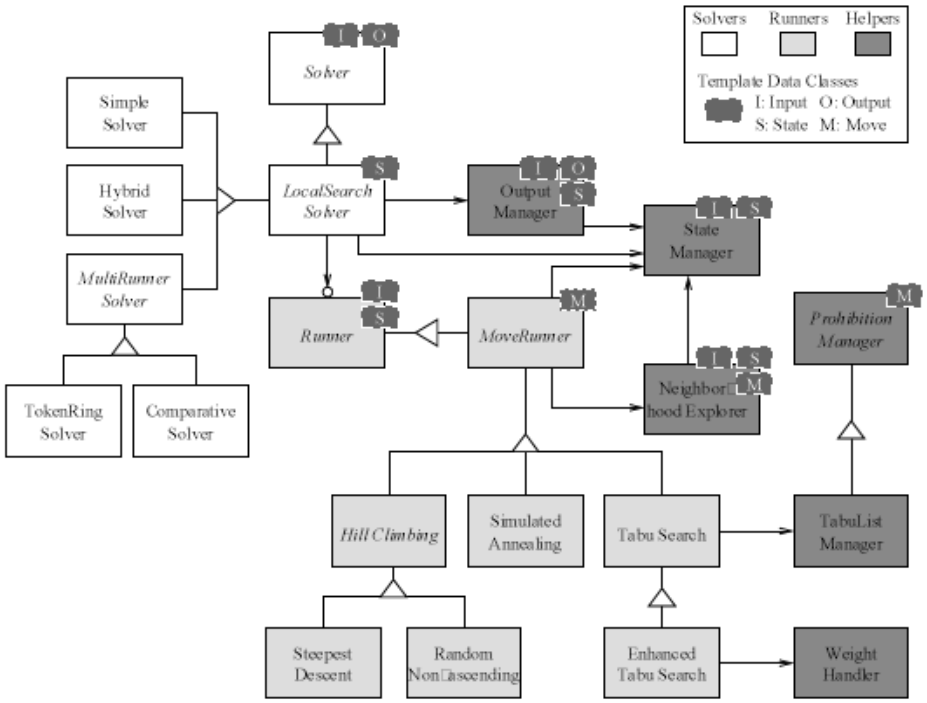


Figure 1. EASYLOCAL++ Main Classes

Other Software Toolkits

SALSA was proposed by (Laburthe and Caseau, 2007) as a language for local and global search algorithms based on the principles of constraint programming offering an environment for combining algorithm components.

VIZ was proposed by (Halim et al., 2006) as an interactive visual analysis tool for illustrating the behaviour of local search algorithms. The tool focuses on visualising the search trajectories and solution spaces visited during the execution of the local search algorithm. There are 3 types of visualisations:

1. Local search visualisations: e.g. search trajectory, objective value, etc.
2. Algorithm specific visualisation: e.g. temperature, re-heat step, etc.
3. Problem specific visualisations: e.g. tours visualisations, etc.

Other toolkits include: HotFrame, OptQuest, Templar, OPL Studio, PISA, GAMS, CODEA, etc.

Further Reading

S. Voss, D.L. Woodruff (eds). *Optimization Software Class Libraries*. Kluwer Academic Publishers, 2002.

Laurent Michel, Pascal Van Hentenryck. *Localizer: a modelling language for local search*. INFORMS Journal on computing, Vol. 11(1), pp. 1-14, 1999.

Laurent Michel, Pascal Van Hentenryck. *Localizer.* Constraints: an international journal, Vol. 5, pp. 43-84, 2000.

P. Van Hentenryck, L. Michel. *Constraint-Based Local Search*. MIT Press, 2005.

C. Voudouris, R. Dorne, D Lesaint, A. Liret. *iOpt: a Software Toolkit for Heuristic Search Methods*. In: T. Walsh (ed.): CP 2001, LNCS 2239, 716-729, 2001.

C. Voudouris, R. Dorne. *Integrating Heuristic Search and One-way Constraints in the iOpt Toolkit*. In: S. Voss, D.L. Woodruff (eds.) Optimization Software Class Libraries, Kluwer academic publishers, 177-191 , 2002.

Further Reading (cont.)

L. Di Gaspero, A. Shaerf. *EasyLocal++: an Object Oriented Framework for Flexible Design of Local Search Algorithms*. Software - Practice and Experience, 33(8), 733-765, 2003.

L. Di Gaspero, A. Roli, A. Shaerf. *EasyAnalyzer: an Object-oriented Framework for the Experimental Analysis of Stochastic Local Search Algorithms*. In: T. Stuetzle, M. Birattari, H.H. Hoos (eds.): SLS 2007, LNCS 4638, 76-90, 2007.

F. Laburthe, Y. Caseau. *SALSA: a Language for Search Algorithms*. Constraints, 7, 255-288, 2002.

S. Halim, R.H. Yap, H.C. Lau. *Viz: a Visual Analysis Suite for Explaining Local Search Behavior*. In: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST 2006), 57-66 , 2006.

Section 1.8 of (Talbi, 2009).