CSC5240: Tutorial #9

Spencer Fung
12 Nov, 2003
GENET Algorithm
GENET

Initialize the network to a random valid state

loop
    for each cluster do
        change the node with max input to “on”
        end for
    if all nodes’ output unchanged then
        if input to all “on” nodes is zero then
            terminate and return the solution
        else
            update connection weight:
            \[ W^{s+1}_{\langle i,j\rangle\langle k,l\rangle} = W^s_{\langle i,j\rangle\langle k,l\rangle} - V^s_{\langle i,j\rangle} V^s_{\langle k,l\rangle} \]
        end if
        end if
    end loop

Solution Found!
Penalization
Random CSP Generator: Gencsp

- Four parameters to specify a binary random CSP generator.
  - Number of variables
  - Domain size
  - Tightness
  - Density
Density and Tightness

- **Density (%)**
  - The percentage of the variable combinations taken by two that will be constrained.

- **Tightness (%)**
  - The tightness is the ratio between the number of value pairs which disallowed by the constraint and the total number of possible value pairs.
- GENET homepage
  - http://cswww.essex.ac.uk/CSP/genet.html

- GENET Demo
  - http://cswww.essex.ac.uk/CSP/genetdemo.html
  - Remark: the demo program can only be executed and compiled in workstation “sparc75”. And it can be compiled by using “gcc” compiler.
Ordering Heuristics in ILOG
IlcChooseIndex1(name, criterion, varType);
IlcChooseIndex2(
    name, criterion1, criterion2, varType);

- These macro define a new choice function (a criterion) in Solver for setting parameters on the search for a solution.
IlcChooseIndex1(name, criterion, varType);

- This macro defines a choice function for constrained variables of type varType.
- The name of the function will be name.
- The second argument, criterion, should be a C++ expression of type IlcInt.
  - In the expression, the constrained variable to evaluate must be denoted by var.
  - The index of the variable in the array is varIndex.
The function named *name* returns the index of the constrained variable of type *varType* that minimizes the expression criterion.

If all the constrained variables have already been bound, then this function returns -1.
Example

IlcChooseIndex1(
    IlcChooseMinSizeInt, var.getSize(), IlcIntVar);

- This function returns the index of the unbound constrained variable with the smallest domain from among the constrained variables in the array of constrained variables.

IlcChooseIndex1(
    IlcChooseMaxSizeInt, -var.getSize(), IlcIntVar);

- This function returns the index of the unbound constrained variable with the greatest domain from among the constrained variables in the array of constrained variables.
How about having two criteria?

IlcChooseIndex2(
    name, criterion1, criterion2, varType);

- You can use this macro if you have *two* integer criteria for the choice function.

- If more than one constrained variable minimizes the first criterion, then the second criterion will be used to distinguish between them.
IlcChooseIndex2(IlCChooseMinSizeMin, var.getSize(), var.getMin(), IlcIntVar)

int main(int argc, char** argv){
    IlcManager m(IlcNoEdit);
    IlcInt nqueen = (argc > 1) ? atoi(argv[1]) : 100;
    IlcIntVarArray x(m, nqueen, 0, nqueen-1),
        x1(m, nqueen),
        x2(m, nqueen);
    x.setName("Solution : ");

    IlcInt i;
    for (i = 0; i < nqueen; i++) {
        x1[i] = x[i]+i;
        x2[i] = x[i]-i;
    }
    // ...
}
/...  
m.add(IlcAllDiff(x));  
m.add(IlcAllDiff(x1));  
m.add(IlcAllDiff(x2));  
**m.add(IlcGenerate(x, IlcChooseMinSizeMin));**  
if (m.nextSolution()) {  
    for (i=0; i<nqueen && i<=100; i++)  
        m.out() << x[i].getValue() << " ";  
    if (nqueen>100) m.out() << "...";  
    m.out() << endl;  
}  
else m.out() << "No solution" << endl;  
m.printInformation();  
m.end();  
return 0;
Value Ordering in ILOG

- Solver offers you predefined objects to choose the next value to try. Those objects are instances of \texttt{IlcIntSelect} (Handle Class).

- You must define the virtual member function of the \textit{implementation} class, \texttt{IlcIntSelectI::select()} to specify the value selection strategies.
Let's assume that you want to control the order in which constrained integer variables are instantiated, and in particular, you want the order of instantiation to be 0, -1, +1, -2, +2, and so forth.

We make use of these two member functions `IlcIntExp::getNextLower` and `IlcIntExp::getNextHigher`. 
**CSC5240 Topics in Constraint Processing**

### getNextHigher and getNextLower

- **IlcInt getNextHigher(IlcInt threshold) const;**
  - If \( \text{threshold} \geq \text{max. of the domain of the invoking constrained integer expression} \), then returns \( \text{threshold} \).
  - Otherwise, it returns the first element that is strictly greater than \( \text{threshold} \).

- **IlcInt getNextLower(IlcInt threshold) const;**
  - If \( \text{threshold} \leq \text{min. of the domain of the invoking constrained integer expression} \), then returns \( \text{threshold} \).
  - Otherwise, it returns the first element that is strictly less than \( \text{threshold} \).
class mySelectI: public IlcIntSelectI {
public:
    mySelectI(){};
    virtual IlcInt select(IlcIntVar var);
};

IlcInt mySelectI::select(IlcIntVar var){
    IlcInt vn = var.getNextLower(1);
    if (vn == 0) return 0;
    IlcInt vp = var.getNextHigher(0);
    if (vp == 0) return vn;
    if (vn == 1) return vp;
    if (-vn <= vp) return vn;
    return vp;
}

IlcIntSelect mySelect(IlcManager m){
    return new(m.getHeap()) mySelectI();
}
int main() {
    IlcManager m(IlCNoEdit);
    IlcIntVar x(m, -3, 5);

    m.add(IlcInstantiate(x, mySelect(m)));

    while (m.nextSolution()) {
        m.out() << x << endl;
    }

    m.end();
    return 0;
}
You should see results like this:

```
[0]
[-1]
[1]
[-2]
[2]
[-3]
[3]
[4]
[5]
```
Any Questions...