Formalize neighborhoods for local search using predicate logic

San Tu Pham

KU Leuven - CODeS

June 5th, 2017

Overview

1. Motivation: The need of a declarative language for local search
2. IDP and FO(.)
3. Modeling neighborhoods for local search using FO(.)

Outline

Model-and-run systems

- Wide variety of solving mechanisms for solving combinatorial optimization problems: MP, CP, Metaheuristics, Satisfiability checking of propositional logic, ASP.. $\rightarrow$ requires expert knowledge of the solving methods.
  - Large number of combinatorial optimization problems.
  - Frequent change of requirements/constraints.
- *Model-and-run*: high-level declarative language + back-end solver.

  - IDP (CP-SAT based), AMPL (MP based), OPL (MP-CP based), Clingo(ASP based), Zinc, Comet (CP based)...
  - Solving mechanisms: mostly exact approaches.

Local search and metaheuristics

- Moving from solution to solution in the search space in order to improve the objective value.
- Are successful in dealing with large-scale problems: vehicle routing, frequency allocation, scheduling problems...
- Drawback: the lack of expresiveness and abstraction.
  - Reuse? Modularity?
  - Change of requirements?

The need of a formal declarative language for local search and metaheuristics

Local search in *model-and-run* system

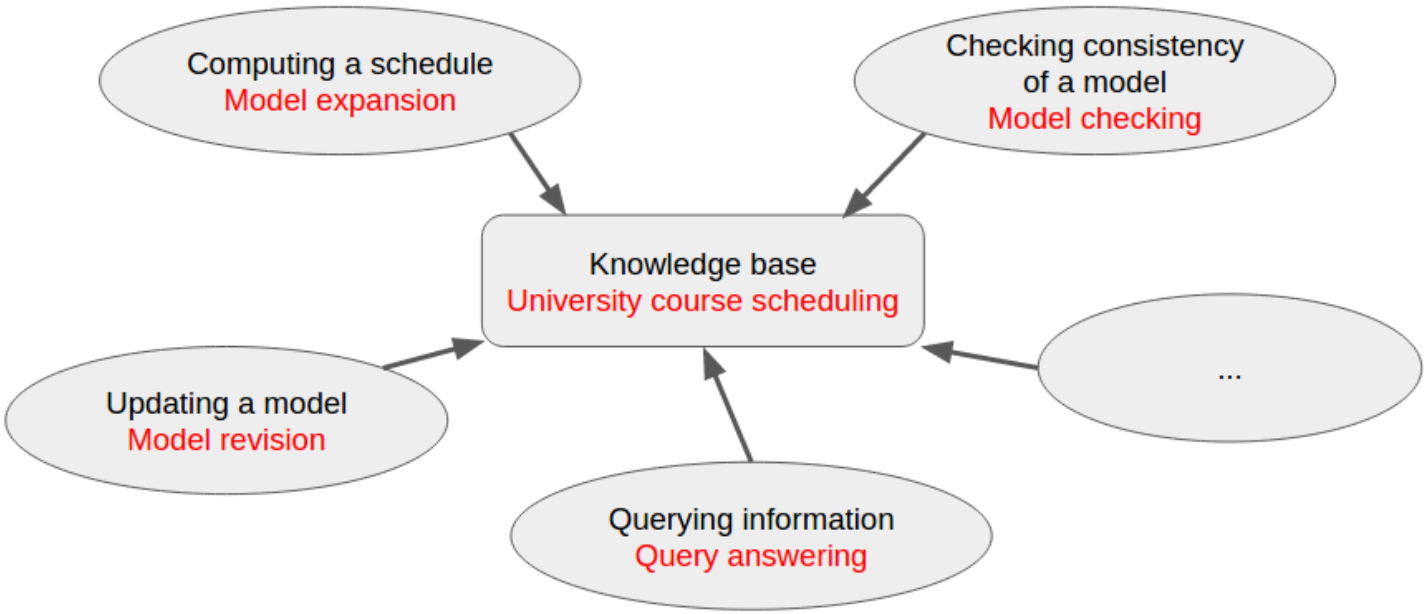
- *Model-and-run* system for local search.
  - Local search is formalized using a high-level declarative language.
  - Back-end solver generates the algorithm based on the declarative description.
- Benefits:
  - Enables modularity and reusability.
  - Allows automatic checking of certain properties.
  - Formalizing complex neighborhoods and metaheuristics is a necessary step to be taken before deriving them automatically.

Formalize local search using **FO(.)** and solve using the **IDP**.

Outline

IDP

- A Knowledge Base System (KBS): separates knowledge from problem solving to allow the reuse of the same knowledge for solving different problems.
  - An expressive high-level modeling language.
  - Many inferences to solve different tasks.



IDP's language: FO(.)

- FO(.) = FO(Types + Aggregates + Arithmetic + Inductive Definitions)
  - FO uses **quantified variables** over non-logical objects and allows the use of sentences that contain variables.
  - $\forall x \in N : \exists y \in N : 2 * x = y$

Modeling the Traveling Salesman Problem in FO(.)

FO(.) specification

```
vocabulary V{
    type Node
    Distance(Node, Node): int
    Path(Node, Node)
    Reachable_From_Depot(Node)
}
structure S:V{
    Distance = { A,A,0; A,B,1; A,C
          ,15; A,D,8;
     B,A,1; B,B,0; B,C,9;  B,D,7;
     C,A,6; C,B,9; C,C,0;  C,D,10;
     D,A,9; D,B,7; D,C,10; D,D,0;}
}
```

```
theory T:V{
    !x: ?1y: Path(x,y).
    !x: ?1y: Path(y,x).
    !x: ~Path(x,x).
    !x: Reachable_From_Depot(x).
    {
        Reachable_From_Depot(MIN[:Node]).
        Reachable_From_Depot(x) <- ?y:
            Reachable_From_Depot(y) & Path(y,x).
    }
}
```

Procedural interface

```
procedure main(){
result = modelexpand(T, S)[1]
    print(result)
}
```

Result

```
structure  : V {
  Node = { "A"; "B"; "C"; "D" }
  Path = { "A","C"; "B","A"; "C","D"; "D","B" }
  Reachable_From_Depot = { "A"; "B"; "C"; "D" }
  Distance = { "A","A"->0; ...}
}
```

model expansion and *minimization*

Model expansion

- Search for assignments to free symbols that satisfy a set of constraints.
- Backend: MINISAT(ID)- a constraint programming solver built around the SAT solver MiniSat.

```
procedure main(){
     result = modelexpand(T, S)[1]
     print(result)
}
```

```
structure  : V {
  Node = { "A"; "B"; "C"; "D" }
  Path = { "A","C"; "B","A"; "C","D"; "D","B" }
  Reachable_From_Depot = { "A"; "B"; "C"; "D" }
  Distance = { "A","A"->0; ...}
}
```

Minimization

- Repeatedly call model expansion with additional constraint on upper bound of the objective function until no solution can be found.

```
term Obj:V{
    sum{x y: Path(x,y): Distance(x,y)}
}
procedure main(){
     result = minimize(T, S , Obj)[1]
     print(result)
}
```

```
structure  : V {
  Distance = { 0; 1; 6; 7; 8; 9; 10; 15 }
  Node = { "A"; "B"; "C"; "D" }
  Path = { "A","B"; "B","D"; "C","A"; "D","C" }
  Reachable_From_Depot = { "A"; "B"; "C"; "D" }
  Distance = { "A","A"->0; ... }
}
```

Outline

Local search

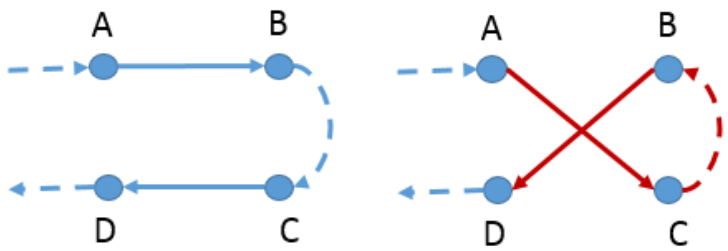
- Local search: moving from solution to solution in the search space in order to improve the objective value.
- Key component: **neighborhood move**.
- The first step towards modeling local search: modeling **neighborhood move** using FO(.).

Modeling 2-opt move for the TSP in FO(.)

```
vocabulary Vproblem{
    type Node
    Distance(Node, Node): int
    dec_Path(Node, Node) //decision
         variables
    Reachable(Node)
    Reach(Node, Node)
}
```

```
theory Tproblem:Vproblem{
    !x: ?1y: dec_Path(x,y).
    !x: ?1y: dec_Path(y,x).
    !x: ~dec_Path(x,x).
    !x: Reachable(x).
    {
        Reachable(MIN[:Node]).
        Reachable(x) <- ?y: Reachable(y) &
             dec_Path(y,x).
    }
    {
        !x: Reach(x,x) <- true.
        !x, y: Reach(x,y) <- dec_Path(x,y) & y
             ~= MIN[:Node].
        !x, y: Reach(x,y) <- ?z: dec_Path(x,z) &
              Reach(z,y) & z ~= MIN[:Node].
    }
}
```

Modeling 2-opt move for the TSP in FO(.)(cont.)




```
vocabulary Vmove{
    extern type Vproblem::Node
    A: Node
    B: Node
    C: Node
    D: Node
}
vocabulary Vnext{
    extern vocabulary Vproblem
    extern vocabulary Vmove
    next_dec_Path(Node,Node)
}
```

```
theory Tnext:Vnext{
{
        //...A -> B ..... C -> D ....
        //...A -> C ..(r)..B -> D ...
        next_dec_Path(A,C) <- true.
        next_dec_Path(B,D) <- true.
        next_dec_Path(x,y) <- dec_Path(x,y) & (Reach(y, A)) & y ~= MIN[:Node].
        next_dec_Path(x,y) <- dec_Path(x,y)  & (Reach(D, x)) & D ~= MIN[:Node].
        next_dec_Path(y,x) <- dec_Path(x,y) & Reach(B,x) & Reach(y,C) & y ~= MIN[:Node].}
}
```

From a solution (a full interpretation of $Vproblem$) and a move, apply *model expansion* on $Tnext$ to get the neighbor solution

Supporting modeling

```
query getvalidMoves:Vproblem{
    {a b c d: dec_Path(a,b) & dec_Path(c,d) & b ~= c & a ~= c & Reach(b,c) & b ~= MIN[:
        Node]}
}

query queryGetDelObj:VdelObj{
    {delObj :  delObj = Distance(A,C) + Distance(B,D) - Distance(A,B) - Distance(C,D) +
        sum{x y: dec_Path(x,y) & Reach(B,x) & Reach(y,C): Distance(y,x) - Distance(x,y)
        }}
}
```

Procedural interface

```
procedure main(){
    result = hillclimbingSearch(T, S , Obj)[1]
     print(result)
}
```

→ *hillclimbingSearch* is a blackbox procedure.

Hill climbing procedure

Algorithm 1: hill climbing search

input : structure S
output: structure *bestSol*

```
1  iniSol ← model expansion on Tproblem;
2  curSol ← iniSol;
3  bestSol ← iniSol;
4  bestDelobj ← 1;
5  repeat
6      moves ← query validmoves from curSol;
7      foreach move ∈ moves do
8          neighbor ← create new solution from move and curSol using model expansion;
9          delObj ← query delObj from neighbor;
10         if delObj < then
11             update bestDelobj, bestSol;
12         end
13     end
14     curSol ← bestSol;
15 until no improvement found or timeout;
```

Preliminary results

Instance	IDP minimization		Hill climbing		
	Obj val	Time	Obj val	Time	Improvement
br17.atsp	39	301.88	39	47.13	0.00%
bays29.tsp	2905	301.83	2066	45.61	28.88%
bier127.tsp	575623	302.09	426110	301.13	25.97%
eil51.tsp	995	301.98	441	67.48	55.68%
ft53.atsp	18615	301.90	11707	93.15	37.11%
ft70.atsp	64305	302.12	52077	164.84	19.02%
ftv33.atsp	2433	301.97	1535	49.35	36.91%
ftv64.atsp	6852	302.04	3367	79.13	50.86%
gr24.tsp	1533	301.68	1279	39.29	16.57%
pr76.tsp	516157	301.93	137134	300.69	73.43%
rbg323.atsp	-	301	5939	304.60	-
rbg358.atsp	-	301	6485	303.24	-
rbg403.atsp	-	301	7363	301.56	-
rbg443.atsp	-	301	8152	320.44	-
ry48p.atsp	40038	301.74	15344	82.29	61.68%
swiss42.tsp	2420	301.74	1378	55.67	43.06%
tsp225.tsp	-	301	25607	302.45	-

Summery and future work

- Summery
  - Formalize neighborhood move in FO(.).
  - Build hill climing search procedure from a declarative description of the neighborhood move.
- Future work
  - More complicated problems and neighborhood moves?
  - Different local search/metaheuristics: late-acceptance hill-climing, tabu search, simulated annealing..?

Thank you for listening!