Formalize neighborhoods for local search using predicate logic

San Tu Pham
Computer Science
KU Leuven, campus Kulak
san.pham@kuleuven.be

Jo Devriendt
Computer Sience
KU Leuven
jo.devriendt@kuleuven.be

Patrick De Causmaecker
Computer Science
KU Leuven, campus Kulak
patrick.decausmaecker@kuleuven.be

***Abstract*—Local search and metaheuristics are powerful approaches to solve difficult combinatorial optimization problems and are particularly suitable for large size problems. However, local search and metaheuristic approaches are often expressed in low-level concepts, which are not suitably formal to allow for modularity and reusability. In this paper, we describe the formalization of neighborhood moves for local search and metaheuristics in predicate logic as a first step towards adressing this problem. This formalization is rooted in FO(·), a first-order logic specification language, which is interpreted by the *knowledge base system* IDP.**

I. Introduction

The number of combinatorial optimization problems arising from practical applications is rising dramatically in the last decade. Along with it is a wide variety of solving mechanisms, including Mathematical Programming (MP), Constraint Programming (CP), Metaheuristics, Satisfiability checking of propositional logic (SAT), Answer Set Programming (ASP) etc. Regardless of the mechanism used, solving a combinatorial problem is a difficult task which requires expert knowledge of the solving method. However, the number of optimization problems is outnumbering the number of solving experts. That is the motivation for a recent trend of *model-and-run*, which allows users to model a problem using a high-level declarative language while a general solving mechanism is provided by some back-end solvers. Some representative systems adopting this idea are IDP [3] (CP-SAT based), AMPL [5] (MP based), OPL [11] (MP-CP based), Clingo [6] (ASP based), Zinc [8], Comet [9] (CP based). However, as a drawback, the solving mechanisms of such systems are mostly exact approaches like CP or MP, which have difficulties solving large or even moderately sized instances of problems.

In the field of optimization, heuristics and metaheuristics have shown their ability to deal with large instances. They are currently the best approaches to many important problems such as vehicle routing or scheduling problems. However, as Van Hentenryck and Michel mention in [12], most local search algorithms are often presented in terms of low-level (implementation) concepts which offers little opportunity for reuse, separation of concerns, and modularity. Related to this issue, Swan et. al. [10] emphasize the need of a pure-functional description of local search and metaheuristics.

The above issues inspire our research question of formalizing local search. To be more specific, we are working on a declarative specification of local search and metaheuristics. The advantages of such formalization are threefold. Firstly, formalization will enable modularity and reusability in designing local search/metaheuristics. A metaheuristic specifications consists of 4 components: hard constraint specification, weak constraint specification, neighborhood and metaheuristic. With some neighborhood specifications at hand, several metaheuristics can be created for a single problem. Small changes in problem requirement are also easy to be taken into account without affecting most of neighborhoods and general algorithms. The second advantage of formalizing local search and metaheuristics is to allow automatic checking of certain properties. In literature, there probably is a significant difference between neighborhood descriptions in papers and their implementations. If neigborhood can be formally described, we can experimentally verify whether the implementation satisfies its formal description by simply generating and testing some examples. In other words, our formalization will allow an automatic simulation of local search and metaheuristics by declaratively specifying the algorithm components. Finally, formalizing complex neighborhoods and metaheuristics is a necessary step to be taken before deriving them automatically.

This paper presents the first step towards local search/metaheuristics formalization: formalizing a *neighborhood* – the basic element of local search and metaheuristics – using predicate logic. As our goal is to automatically reason over neighborhoods, or easily run simulations of local search algorithms, it is useful to use a formal language interpreted by an automated reasoning system. For this, we employ FO(·) as the formal language, as FO(·) is rooted in well-known first-order logic. Besides, the IDP system can interpret FO(·) specifications, and perform various reasoning tasks over these specifications [1].

The rest of the paper is organized as follows. Section II gives a brief introduction about IDP and FO(·). Section III shows how a neighborhood is modeled using FO(·) and how we simulate a local search procedure using this neighborhood specification. Conclusion and future work are given in section IV.

II. IDP and FO(.)

The IDP system [1] is a Knowledge Base System (KBS) that aims at separating knowledge from problem solving to

```
1  vocabulary V{
2      type row isa int
3      type col isa int
4      type intrange isa int
5      queen(row): col
6      violation1(row): intrange
7      violation2(row): intrange
8      violations(): intrange
9  }
10
11 theory T:V{
12     ∀x: violation1(x) = #{y: y ≠ x ∧ queen( x) = queen(y)}.
13     ∀x: violation2(x) =
14         #{y: y ≠x ∧ abs(x - y) = abs(queen(x) - queen(y))}.
15     violations() =
16         sum{x: ∃y: queen(x)=y: violation1(x) + violation2(x)}.
17 }
18
19 term O: V{
20     violations
21 }
22
23 structure S: V{
24     row = {1..4}
25     col = {1..3}
26     intrange = {-100..100}
27 }
28
29 procedure main(){
30     result = minimize(T, S, O)[1]
31     print(result)
32 }
```

Fig. 1. Modeling nqueens problem in FO(·)

allow the reuse of the same knowledge for solving different problems. IDP allows users to describe knowledge of a problem domain using an expressive high-level modeling language. This specification is called the *knowledge base* (KB). On this KB, IDP can perform *inferences* to solve different tasks. Many of the inferences are fundamentally different (e.g. theorem proving and querying), so the IDP system applies a specific solving approach for most of the inference methods. In the context of this paper, the inference of interest is *model expansion*, which searches for assignments to free symbols that have to satisfy a set of constraints. Model expansion can be utilized to solve classic search problems such as Nqueens or more complex planning problems where a sequence of actions has to be timed to obtain some desired state. As backend, IDP's current model expansion uses MINISAT(ID) [2], [7], a constraint programming solver built around the conflict driven clause learning SAT solver MiniSat [4].

The input language of IDP is FO(·), a logic that extends first-order logic with types, aggregates, arithmetic and inductive definitions. FO(·) allows users to specify domain knowledge of the problem as a logical theory consisting of predicate logic formulas.

Here, we briefly illustrate modeling in FO(·) through a modified version of the Nqueens problem. The original Nqueens problem is a decision problem where n queens are placed on an n by n chessboard so that no queen threatens another. In this paper, we are interested in optimization problems only so we modify the problem as follows: we try to place n queens on a n by m chessboard, each queen on one row. Since the number of columns m can be smaller than the number of rows n (also the number of queens), there will be some queens threatened by being placed on the same column or diagonal with another queen. We call it a violation. The problem is to find positions of the queens so that the total number of violations is minimized. We know that for problems with $n = m$ and $n > 4$, the optimal solution has 0 total violations. The problem is modeled in FO(·) as in figure 1.

An IDP specification (or modeling) consists of several components. The first component is the *vocabulary* V that introduces the types, constants, functions and predicates used in the specification. For the modified Nqueens, two types row, col are declared. isa stands for a subset relation (*is a*). $queen(row) : col$ is a function which maps a queen at a row to a column. An interpretation to $queen(row) : col$ represents a solution for the Nqueens problem. For example, $queen(2) = 3$ means the queen at row 2 is placed at column 3. $violation1(r)$ represents the number of queens on the same column as the queen on row r. $violation2(r)$ represents the number of queens on the same diagonal as the queen on row r. $violations$ keeps track of the total violations of all queens. The data type of violations are $intrange$, which is a finite subset of int.

The second component is the *theory* T which specifies the constraints of the problem. Line 12 defines $violation1$ as the total violation of queens being on the same column. $\#$ is a cardinality aggregate symbol. $\#\{x : \phi(x)\}$ counts the number of x that makes $\phi(x)$ true. Line 12 can be interpreted roughly as: $violation1$ is the number of tuples (x, y) so that $queen(x) = queen(y)$. Similarly, line $13, 14$ define the violation of queens being on the same diagonal while line $15, 16$ sum over both types of violation.

The third component specifies the *term* O which is the total number of violations and will be used as objective function.

The fourth component is the *structure* S that describes type domains and input values for the instance of interest. Structure S in figure 1 specifies an instance with 4 rows and 3 columns.

The fifth and final component is a piece of imperative code of which the most important line is $result = minimize(T, S, O)[1])$ instructing to execute the minimization inference method with theory T, input structure S, objective term O, and to search for the minimal solution. The IDP specification along with the executable code can be found at goo.gl/vKpcol.

III. MODELING NEIGHBORHOODS FOR LOCAL SEARCH USING FO(.)

Local search in a heuristic method to solve a minimize/-maximize optimization problems by moving from solution to solution in the search space in order to find a local optimum. Local search algorithms move from one solution to another (neighbor solution) in the neighborhood space by applying local changes, which is called neighborhood relation or neighborhood move. In this section, we present how a neighborhood move is modeled using FO(·). The idea is to create a duplicated vocabulary to represent a neighbor solution. Given a move (which can be specified as an FO(·) predicate or function), the relation between current solution and neighbor solution is specified in an FO(·) theory using logic formulas and/or inductive definitions. Given a solution (which is an interpretation of the original vocabulary) and a move, model expansion is applied to retrieve the neighbor solution. The difference in objective value between two neighbor solutions, referred to as the *delta objective value*, can also be obtained by

```
15  vocabulary VLS{
16      extern vocabulary V
17      next_queen(row): col
18      next_violation1(row): intrange
19      next_violation2(row): intrange
20      next_violations(): intrange
21
22      delObj(): intrange
23      move(row, col)  //move(r, c) puts queen at row r to col c
24  }
25
26  theory T:VLS{
27      ∀x: violation1(x) = #{y: y ≠ x ∧ queen( x) = queen(y)}.
28      ∀x: violation2(x)
29          = #{y: y ≠x ∧ abs(x - y) = abs(queen(x) - queen(y))}.
30      violations()
31          = sum{x: ∃y: queen(x)=y: violation1(x) + violation2(x)}.
32      {
33          next_queen(i) = j ← move( i, j).
34          next_queen(i) = queen(i) ← ¬∃j: move(i,j).
35      }
36      ∀x: next_violation1(x) =
37          #{y: y ≠ x ∧ next_queen( x) = next_queen(y)}.
38      ∀x: next_violation2(x) =
39          #{y: y ≠x ∧ abs(x - y) = abs(next_queen(x) - next_queen(y))}.
40      next_violations =
41          sum{x:∃y: next_queen(x)=y:next_violation1(x) + next_violation2(x)}.
42
43      delObj = next_violations - violations.
44      ∃1q v: move(q,v).
45  }
46
47
48  procedure main(){
49      result = localsearch(VLS,T,S)
50      print(result)
51  }
```

Fig. 2. Modeling Nqueens problem in FO(·)

model expansion given its value is specified in the theory. This delta objective can be defined as an approximate evaluation to speed up the metaheuristics or local search using them. Using these neighborhoods and delta objective values provided by IDP, many local search or metaheuristic approaches can be simulated.

The idea is illustrated using the running example of the modified Nqueens. We consider a simple move which relocates a queen (uniquely associated with a row) to a different column. To represent a neighbor solution, we declare a duplicated vocabulary V_{LS} which takes V as a subvocabulary. The neighbor solution is represented by the function $next\_queens$. Violation information is represented by $next\_violation1$, $next\_violation2$ and $next\_violations$. Predicate $move(row, col)$ represents a neighborhood move. $move(r, c)$ means the queen at row r is moved to column c in the next solution. $delObj$ gives the delta between the total violations of the current solution and that of the neighbor solution.

In theory T, besides the constraints from the original modeling, we have an *inductive definition* (line 32 to 35) to define the neighbor solution based on the current solution (encoded in $queen$) and the predicate $move$. Roughly, the definition states: "If there exists a $move(i, j)$, then in the next solution, the queen at row i will be placed at column j. The rest of the queens have the same positions as in the current solution." The next six lines define the violation information in the next solution. Line 43 calculates $delObj$ as the difference between total violations of the next solution and the current one. The last line in the theory states that there exists exactly one move, disallowing multiple move applications in a single local search step.

In the empirical code, procedure $localsearch$ is called to perform a simple hill climbing local search. Firstly, an initial solution is obtained by applying model expansion. Then the procedure iteratively generates a move, applies model expansion to obtain a neighbor solution from the current one and get the delta objective to evaluate the new solution. If the delta objective value is negative, the objective function is improved, and the obtained neighbor solution now functions as the current solution whose neighborhood is further investigated. The pseudo code for $localsearch$ procedure is presented in algorithm 1.

Algorithm 1: local search

```
input : Vocabulary V_LS, theory T, structure S
cur_sol ← apply model expansion to construct initial solution;
repeat
    m ← number of rows;
    n ← number of columns;
    foreach r ∈ 1..m do
        foreach c ∈ 1..n do
            move(r, c) ← create a move from (r, c);
            queen(r): c ← get the queens interpretation from cur_sol;
            next_sol ← create new solution from queen(r): c and move(r, c) by applying model expansion;
            delObj ← get delObj from next_sol;
            if delObj < 0 then
                cur_sol ← next_sol;
            end
        end
    end
until stopping condition is met;
```

IV. Conclusion and Future Work

In this paper, we present how to formalize neighborhoods using predicate logic through the example of the Nqueens problem. This is the very first step towards formalizing meta-heuristics/local search and lots of work still need to be done to reify the idea. In the near future, we plan to investigate other problems and different types of neighborhood moves. Further steps include formalizing meta-heuristics and investigating other techniques to increase the procedure's performance.

References

[1] Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker. Predicate logic as a modelling language: The IDP system. *CoRR*, abs/1401.6312v2, 2016.

[2] Broes De Cat, Bart Bogaerts, Jo Devriendt, and Marc Denecker. Model expansion in the presence of function symbols using constraint programming. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pages 1068–1075. IEEE Computer Society, 2013.

[3] Marc Denecker and Joost Vennekens. Building a knowledge base system for an integration of logic programming and classical logic. In María García de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *LNCS*, pages 71–76. Springer, 2008.

[4] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[5] Robert Fourer, David M Gay, and Brian W Kernighan. *AMPL: A mathematical programming language*. AT&T Bell Laboratories Murray Hill, NJ 07974, 1987.

[6] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo= asp+ control: Preliminary report. *arXiv preprint arXiv:1405.3694*, 2014.

[7] Maarten Mariën, Johan Wittocx, Marc Denecker, and Maurice Bruynooghe. SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In Hans Kleine Büning and Xishun Zhao, editors, *SAT*, volume 4996 of *LNCS*, pages 211–224. Springer, 2008.

[8] Kim Marriott, Nicholas Nethercote, Reza Rafeh, Peter J. Stuckey, Maria Garcia de la Banda, and Mark Wallace. The design of the Zinc modelling language. *Constraints*, 13(3):229–267, 2008.

[9] Laurent Michel and Pascal Van Hentenryck. The Comet programming language and system. In Peter van Beek, editor, *CP*, volume 3709 of *LNCS*, pages 881–881. Springer, 2005.

[10] Jerry Swan, Steven Adriaensen, Mohamed Bishr, Edmund K Burke, John A Clark, Patrick De Causmaecker, Juanjo Durillo, Kevin Hammond, Emma Hart, Colin G Johnson, et al. A research agenda for metaheuristic standardization. In *Proceedings of the XI Metaheuristics International Conference*, 2015.

[11] Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, 1999.

[12] Pascal Van Hentenryck and Laurent Michel. *Constraint-Based Local Search*. MIT Press, 2005.