Modelling Local Search in a Knowledge Base System

Tu-San Pham, Jo Devriendt and Patrick De Causmaecker

Abstract In this paper we present how the basic building blocks of local search approaches—problem constraints, neighbourhood moves, objective function, move evaluations—can be modelled declaratively using FO (·), an extension of first order logic. We extend the Knowledge Base System IDP with three built-in local search heuristics, namely first improvement, best improvement and tabu search, which take those building block specifications as input and execute local search accordingly. To demonstrate the framework, three neighbourhood moves for three different problems are modelled and tested.

Keywords Local search · Metaheuristics · Knowledge base system

1 Introduction

A Knowledge Base System (KBS) is a computer program that reasons and uses a general knowledge base to solve different complex problems. The IDP system [4] is an experiment on building such a KBS using the rich formal language FO (·) [7] for expression of the knowledge. FO (·) is an extension of first order logic with types, cardinality, arithmetics, and recursiveness to elegantly model of a wide range of concepts. The knowledge specified in FO (·) is called a *knowledge base* (KB), which IDP uses as input to a set of builtin *inference methods* to solve a large variety of tasks.

For example, given a KB on professors teaching courses, students studying, resources available and rules to be respected, the IDP system can generate schedules at the start of the year using the *model expansion* inference; verify correctness of hand-made and revised schedules using the *model checking* inference; generate a reason of why a certain schedule is not consistent using the *explain unsat* inference; proving invariants on the knowledge using the *theorem proving* inference—all using the same KB. The ability to solve many tasks distinguishes IDP from other program-

T.-S. Pham (✉) · J. Devriendt · P. De Causmaecker
KU Leuven, Leuven, Belgium
e-mail: san.pham@kuleuven.be

P. Daniele and L. Scrimali (eds.), *New Trends in Emerging Complex Real Life Problems*, AIRO Springer Series 1,
https://doi.org/10.1007/978-3-030-00473-6_44

ming languages or modelling systems, which are suited for solving only one task. IDP has been used in application domains such as interactive configuration [14, 15], machine learning and data mining [2] or access control systems [3].

Important classes of problems appearing in many applications are constraint satisfaction and combinatorial optimization, which also are the focus of this paper. IDP's inferences to solve such problems are *model expansion*, and its variant, *optimization*. The current back-end engine for these two inferences is MINISAT(ID) [5]. As a CP-SAT-based solver, it shows limited performance on typical operations research problems, such as the assignment problem [8], or on large real-world problems such as routing and scheduling. Local search on the other hand, is a well-studied combinatorial optimization technique, which has been succesfully applied to solve such problems.

In this paper, we specify local search neighbourhoods in FO (·) and extend the IDP system with a local search back-end. Doing so, we hope to arrive at a declarative, modular and reusable formalization of local search heuristics. Given a problem, as long as the problem's specifications and its necessary components (how to get valid moves, how to evaluate delta objective, how to create the neighbour solution given a current solution and a move) can be modelled in FO (·), the local search can be simulated in IDP.

The contribution of this work is two-fold. Firstly, formalizing local search neighbourhoods allows us to benefit from the existing functionality of the underlying system (in this case, IDP). E.g., we can automatically generate an initial feasible solution using the *model expansion* inference; we can debug neighbourhoods by using the *explainunsat* inference (which gives the reason why a solution is not feasible). Those benefits from the underlying system also distinguish this work from related work such as constraint-based local search systems (Comet [11], Oscar [12]) and Localsolver [1], which are particularly dedicated to solving optimization problems and are based on a set of built-in libraries of neighbourhood moves and combinators to construct local search heuristics for a set of problems. Secondly, the ultimate goal of a knowledge base system is to tackle all types of problems efficiently—including those for which the current state-of-the-art employs local search. Our work is a step towards introducing local search in a knowledge base environment.

The rest of the paper is organized as follows. Section 2 gives a brief introduction of IDP and FO (·). Section 3 shows how neighbourhood moves and move evaluations are formalized in FO (·), and how three local search heuristics are simulated using these formalizations. Experimental results are reported in Sect. 4. Conclusions and future work close the paper in Sect. 5.

2 IDP and FO (·)

The IDP system [4] is a Knowledge Base System (KBS) equipped with (1) FO (·) , a high level language which allows users to specify knowledge of a problem domain, and (2) a set of *inferences* to solve a wide range of problems around this knowledge.

A thorough introduction to IDP and FO (·), including examples and demos, can be found at https://dtai.cs.kuleuven.be/software/idp. Many of IDP's inferences are fundamentally different (e.g. theorem proving and model checking), therefore a specific solving approach is applied for each inference method. For *model expansion* and its variant *optimization*, IDP employs a two-phase *ground-and-solve* strategy where a FO (·) specification is grounded to a set of constraints in Extended Conjunctive Normal Form (ECNF) [6], which is then handled by MINISAT(ID) [5], a CP-SAT-based solver.

A specification (or modelling) in IDP consists of different components. The three most important components are *vocabularies* specifying the types and variables used, *theories* specifying problem constraints in FO (·), and *structures* representing both input data and solutions.

As a running example we employ the travelling salesman problem (TSP). Given a set of nodes and a distance function between them, the TSP consists of finding the shortest Hamiltonian cycle—the *tour*—visiting all cities. Here, we give a model for the TSP in IDP which we also made available online at http://goo.gl/TTv85c.

Example 2.1 (TSP)

The four components for the TSP modelling in IDP are as follows:

1. The vocabulary $Vproblem$ specifying the parameters ($Node$, $Distance$) and the variables ($Path$, $Reachable$) of the problem.
2. The theory $Tproblem$ over $Vproblem$, which is a conjunction of FO (·) formulas specifying the constraints of the problem:

$$\begin{aligned}
&\forall x : \exists! y : Path(x, y).\\
&\forall x : \exists! y : Path(y, x).\\
&\{\ Reachable(Depot).\\
&\quad Reachable(x) \leftarrow \exists y : Reachable(y) \wedge Path(y, x).\}\\
&\forall x : Reachable(x).
\end{aligned}$$

   The first two lines of the theory in the example represent the flow constraints. Line three and four inductively define the $Reachable$ predicate, starting from the depot, and inductively adding neighbouring nodes according to the links present in $Path$. The last line then states that all nodes must belong to $Reachable$, forming a subtour elimination constraint—if a node is not reachable from the depot, it belongs to a subtour different from the one containing the depot.
3. The term Obj over $Vproblem$ representing the total travelling distance and serving as the objective function.
4. The (partial) structure S over $Vproblem$ describing type and parameter values. In S, the assignment to problem parameters (in this case, $Node$ and $Distance$) represents an instance of the problem, while assignments to variable symbols (in this case, $Path$ and $Reachable$) could form a solution for that instance. IDP distinguishes between parameters and variables simply by the fact that variables are unassigned in the initial structure.

Given these components, the inference *model expansion* can be applied to expand a (partial) structure S (an input instance) into a (complete) structure respecting the constraints in theory $Tproblem$ (a feasible solution); the inference *minimization* can be applied to obtain the minimal solution according to the objective function Obj. Besides, users can specify a *query* and apply *querying* inference to retrieve information from the model.

3 Modelling Local Search

Local search is a heuristic method to solve an optimization problem by moving from solution to solution in the search space using *neighbourhood moves*. A neighbourhood move is a function which maps a solution to another. Given a set of neighbourhood moves and some initial feasible solution, a local search algorithm explores the search space by enumerating the neighbours. When a neighbour satisfies an acceptance criterion, it becomes the starting point of a new iteration. This loop ends when a stop condition is met, and the solution with the best objective value is returned. Metaheuristics are local search-based heuristics, which guide a subordinate heuristic to escape from local optima.

In this section, the formalization of local search heuristics' building blocks in FO $(\cdot)$ is presented. We further describe how IDP is extended with three local search heuristics—first improvement, best improvement and tabu search.

3.1 Modelling Local Search Neighbourhood Moves

To model local search neighbourhood moves for a problem, several components are added to the problem specification. The essential part of the modelling is the function mapping a current solution to the corresponding neighbour solution given a neighbourhood move. This function is modelled in a theory *Tnext* which is built over two vocabularies *Vmove* representing a move and *Vnext* representing a neighbour solution. A query *queryGetDeltaObj* is added to allow a user-defined move evaluation. Valid moves are obtained by a query *queryGetValidMoves*. The above components are illustrated using the running example of the TSP as follows.

Example 3.1 In this example, we model the *2-opt* move for the TSP. In order to model the move, the problem specification is extended by introducing the predicate $Reach$ to represent the order of nodes in the solution. For example, $Reach(A, C)$ holds means that on the path starting from the depot, node A appears before C. To represent moves and neighbour solutions, two vocabularies $Vmove$ and $Vnext$ are added to the modelling. $Vmove$ consists of 4 constants A, B, C and D representing the four nodes involved in the 2-opt move—(A, B) and (C, D) are two edges in the solution in this specific order. $Vnext$ consists of the predicate $next\_dec\_Path(Node, Node)$, which represents a neighbour solution. The function mapping a current solution to its neighbour is defined in theory *Tnext* as below:

$$
\begin{aligned}
&\{next\_dec\_Path(A, C).\\
&next\_dec\_Path(B, D).\\
&next\_dec\_Path(x, y) \leftarrow dec\_Path(x, y) \wedge Reach(y, A) \wedge y = minimum(Node).\\
&next\_dec\_Path(x, y) \leftarrow dec\_Path(x, y) \wedge Reach(D, x) \wedge D = minimum(Node).\\
&next\_dec\_Path(y, x) \leftarrow dec\_Path(x, y) \wedge Reach(B, x) \wedge Reach(y, C)\\
&\qquad\qquad \wedge y \neq minimum(Node).\}
\end{aligned}
$$

Theory *Tnext* consists of a definition describing how to create a neighbour solution given a solution and a move. Roughly, it states that the new solution consists of 2 edges (A, C) and (B, D), the segment from the depot to A, the segment from D to the depot and the reversed segment from B to D. By applying *model expansion* on *Tnext* and a structure with assignments to dec_Path, $Reach$ and A, B, C, D, an interpretation of $next\_dec\_Path$, which represents a neighbour solution.

To complete the modelling, two queries are declared. Query $queryGetDelObj$ evaluates a move by calculating the delta between the total travelling time of the current solution and its neighbour: $\Delta = d_{AC} + d_{BC} - (d_{AB} + d_{CD})$. Query $queryGetValidMoves$ gets all valid moves from a given solution.

3.2 Metaheuristics Framework

In this section, we describe how IDP takes formal neighbourhood move specifications as input and combines them with the built-in heuristics to perform local search. The common mechanism is as follow. Firstly, an initial feasible solution is obtained using IDP's model expansion on *Tproblem*. Next, a set of valid moves is computed by solving the query *queryGetValidMoves* on the current solution. For each valid move, the query *queryGetDelObj* is applied to evaluate the move. If the move is accepted, the corresponding neighbour is computed by performing model expansion on $Tnext$. The obtained neighbour solution now functions as the current solution whose neighbourhood is further investigated in the next iterations.

As already mentioned, three local search heuristics are provided: first improvement, best improvement and tabu search. First improvement search and best improvement search are two simple heuristics which iterate through all valid moves and proceed with a move once the first improvement or the best improvement is found. Tabu search [9] is a more advanced metaheuristic which keeps a list of forbidden moves to avoid revisiting a recently visited solution. These heuristics take the neighbourhood move specifications described in Sect. 3.1 as input and run the corresponding local search heuristic accordingly. By way of illustration, we describe the first improvement local search procedure in Algorithm 1.

Given a problem, as long as the problem along with the necessary components can be modelled in FO $(\cdot)$, a local search can be simulated using the framework.

Algorithm 1: First improvement search

```
   input  : instance S
   output: solution bestSol
 1 iniS ← model expansion on (S, Tproblem);
 2 curSol ← iniS;
 3 bestSol ← iniS;
 4 repeat
 5     moves ← evaluate queryGetValidMoves on curSol;
 6     foreach move ∈ moves do
 7         delObj ← evaluate queryGetDelObj on move and curSol;
 8         if delObj < 0 then
 9             neighbour ← create new solution from move and curSol by applying model
               expansion on Tnext;
10             bestSol ← neighbour;
11             curSol ← neighbour;
12             break;
13         end
14     end
15 until no improvement found or timeout;
```

4 Experiments

The main purpose of the experiment is to demonstrate how the framework is utilized to model and solve different problems. We model and test three different neighbourhood moves for three different problems as follows: (1) The TSP and the 2-opt move neighbourhood as presented in Sect. 3; (2) *The assignment problem* which consists of finding a bijection between a set of agents and a set of tasks that minimizes the assignment cost. A simple neighbourhood with moves that swap the assignment of two agents is modelled; (3) *The colouring violations problem* which consists of finding a graph colouring which minimizes the number of adjacent nodes sharing the same colour. The neighbourhood consists of moves that assign a different colour to a node. The modelling and executable code for all problems in this section can be found at http://github.com/tusanpham/LocalSearchInIDP.git.

We modelled each of these problems and neighbourhoods in FO $(\cdot)$. Those specifications are then taken as input by IDP to perform the three built-in local search heuristics (first improvement, best improvement and tabu search). Additionally, for each problem, we compare the results with IDP's builtin minimization inference.

Instances for the TSP were obtained from the TSPLIB [13], instances for the assignment problem were taken from [8], and instances for the colouring violations problem were taken from Michael Trick's operations research page [10]. A time limit of 300 seconds were imposed on all experiments. The experiment was conducted on an Intel Core i7-5600 cpu with 8GiB of RAM, running Ubuntu 14.04 64-bit. The IDP system we extended was IDP 3.6.

In Table 1, results of some representative instances from the three problems are reported. The first two columns show the best objective value and running time

Table 1 Results of the three test problems

	Instance	IDP minimization		Best improvement		First improvement		Tabu search	
		Best sol	Time	Best sol	Time	Best sol	Time	Best sol	Time
TSP	br17.atsp	39	301.88	39	3.83	39	4.37	39	3.66
	eil51.tsp	995	301.98	**441**	46.94	461	40.83	**441**	51.39
	ft70.atsp	64,305	302.12	47,078	300.11	61,173	352.98	**46,155**	300.77
	ftv33.atsp	2433	301.97	1658	300.02	2047	301.38	**1577**	300
	pr76.tsp	516,157	301.93	**113,187**	195.61	114,353	124.28	**113,187**	200.45
	rbg443.atsp	–	–	**8169**	399.09	8126	373.1	**8169**	403.96
	swiss42.tsp	2420	301.74	**1378**	35.48	1418	40.34	**1378**	35.56
Assignment	21	**52**	300	65	40.19	63	39.6	60	301.92
	25	67	300	76	83.96	**60**	57.3	**60**	304.29
	29	142	300	95	97.93	89	81.76	**74**	303.96
	33	127	300	**90**	176.47	95	148.89	**90**	301.02
	37	337	300	111	250.48	141	168.34	**103**	305.07
Colouring	anna.col	**2**	300	342	301.71	118	102.96	304	333.27
	homer.col	**641**	29.62	999	300	815	302.42	999	300
	le450_25a.col	**533**	300	1000	300	694	308.341	1000	300
	miles1500.col	8344	300	8967	300	**3032**	305.971	8868	300
	mulsol.i.4.col	**246**	300	917	308.01	384	303.383	917	300.06
	queen7_7.col	**0**	118.59	6	154.73	6	115.004	**0**	303.03
	zeroin.i.3.col	537	300	854	307.96	**286**	306.847	854	306.21

of IDP's minimization while the next columns contain results obtained by the best improvement, first improvement and tabu search respectively.

We firstly discuss the results on the TSP problem. It is apparent from the table that all three local search heuristics outperform IDP's minimization routine, especially in big instances. This suggests that at least for the TSP, a general solver can benefit from domain knowledge which can be modelled easily using a descriptive language, as we did with the 2-opt neighbourhood.

The results of the assignment problem show a clear difference between the three heuristics. Tabu search outperforms the remaining two heuristics on most instances, which is easily explained by the strong mechanism of tabu search to prevent repeating moves. This result highlights the benefit of modularity offered by our framework. Given a single neighbourhood formulation, it is easy to experiment with different heuristics before committing to the best one.

Finally, the results from the colouring violations problem, in contrast to the two above problems, show a win for the IDP's minimization routine over the three heuristics. This can be explained by the CP-SAT backend behind IDP's minimization being more suitable for this particular problem compared to local search with a simple swap move. This shows the benefit of having several back-ends in a single engine.

In the TSP, besides performing local search, we also experiment the *explain unsat* reference to reason why a solution is not valid. The inference works well on instances up to 78 nodes and gives a readable explanation on which variable assignments lead to constraints violation. The usage of *explain unsat* is available online at http://goo.gl/U4hvkf.

5 Conclusions and Future Work

In this paper, we describe our work on a local search framework within the Knowledge Base System IDP using the formal language FO (·). Three local search heuristics are provided, taking a formal description of problem's constraints, neighbourhood moves, objective function, and move evaluations as the input. The framework is demonstrated through three different problems in the experiment section. The experiment is a proof-of-concept on modelling and automatically exploiting neighbourhoods of different problems, and gauges the performance of our proof-of-concept. It is also interesting that we have improved the IDP's performance on two problems (TSP and assignment problems) using very simple local search modelling. It shows that we can improve a general solver by supplying it with neighbourhood specifications.

Regarding future work, we plan to propose a complete framework which allows specifying multiple neighbourhoods and combining them to create more sophisticated local search heuristics. In addition, the use of a formal language in our work enables the potential of applying formal methods to local search. For example, an automatic theorem prover can be applied to prove the correctness of neighbourhoods.

References

1. Benoist, T., Estellon, B., Gardi, F., Megel, R., Nouioua, K.: Localsolver 1. x: a black-box local-search solver for 0–1 programming. 4OR: Q. J. Op. Res. **9**(3), 299–316 (2011)
2. Bruynooghe, M., Blockeel, H., Bogaerts, B., De Cat, B., De Pooter, S., Jansen, J., Labarre, A., Ramon, J., Denecker, M., Verwer, S.: Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with idp3. Theory. Pract. L. Program. **15**(6), 783–817 (2015)
3. Cramer, M., Van Hertum, P., Ambrossio, D.A., Denecker, M.: Modelling delegation and revocation schemes in IDP. arXiv:1405.1584 (2014)
4. De Cat, B., Bogaerts, B., Bruynooghe, M., Janssens, G., Denecker, M.: Predicate logic as a modelling language: the IDP system (2016). http://arxiv.org/abs/1401.6312v2
5. De Cat, B., Bogaerts, B., Devriendt, J., Denecker, M.: 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, 4–6 November 2013, pp. 1068–1075. IEEE Computer Society (2013). http://dx.doi.org/10.1109/ICTAI.2013.159
6. De Cat, B.: Separating knowledge from computation: An FO(·) knowledge base system and its model expansion inference. Ph.D. thesis. KU Leuven, Leuven, Belgium (2014)
7. Denecker, M., Vennekens, J.: Building a knowledge base system for an integration of logic programming and classical logic. In: M. García de la Banda, E. Pontelli (eds.) ICLP, LNCS, vol. 5366, pp. 71–76. Springer (2008). http://dx.doi.org/10.1007/978-3-540-89982-2_12
8. Devriendt, J.: Exploiting symmetry in model expansion for predicate and propositional logic. Ph.D. thesis, Informatics Section, Department of Computer Science, Faculty of Engineering Science (2017). https://lirias.kuleuven.be/handle/123456789/564687. Denecker, Marc (supervisor)
9. Glover, F., Laguna, M.: Tabu search. In: Handbook of Combinatorial Optimization, pp. 3261–3362. Springer (2013)
10. Michael trick's operations research page. https://mat.gsia.cmu.edu/COLOR/instances.html (2017)
11. Michel, L., Van Hentenryck, P.: The Comet programming language and system. In: P. van Beek (ed.) CP, LNCS, vol. 3709, pp. 881–881. Springer (2005)
12. OscaR Team: OscaR: Scala in OR (2012). https://bitbucket.org/oscarlib/oscar
13. TSP instance library. www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (2017)
14. Van Hertum, P., Dasseville, I., Janssens, G., Denecker, M.: The KB paradigm and its application to interactive configuration. Theory. Pract. L. Program. **17**(1), 91–117 (2017)
15. Vlaeminck, H., Vennekens, J., Denecker, M.: A logical framework for configuration software. In: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming, pp. 141–148. ACM (2009)