Knowledge-Driven Decision Optimization for Non-Experts

Yishai A. Feldman, Aviad Sela, Segev Wasserkrug

IBM Research – Haifa
Mount Carmel, 3498825 Haifa, Israel
{yishai, sela, segevw}@il.ibm.com

Abstract

Decision optimization is a pressing need of small and large enterprises, who are trying to succeed in a competitive and rapidly changing marketplace. The creation of decision optimization models typically requires modeling the problem in a manner that can be solved by optimization engines such as IBM CPLEX®. While high-level modeling languages exist for many solvers, their use requires significant optimization expertise, due to the need to model the problem in terms of constraining variable values, and to create models that can be solved efficiently. This severely limits the widespread use of optimization. In this demo, we present an alternative approach, which enables users who are not optimization experts to write programs that verify that a given solution satisfies the problem constraints and compute the value of the objective function. These programs are written in popular languages such as Python; our technology, demonstrated here, analyzes such specifications and produces models for an optimization solver. We believe that this approach will enable the wider community of data scientists and developers to model optimization problems, enabling a much more widespread use of optimization.

Knowledge-Driven Optimization

Decision optimization can provide significant value to many enterprises, providing benefits typically measured in millions, and sometimes billions, of dollars. However, realizing this value often requires modeling the problem in a way that can be solved by optimization engines such as IBM CPLEX®. This is complex, as such modeling requires specification in terms of constraining variables rather than computing their value.

As a running example, consider the problem of optimally assigning people to offices so that several constraints are met. Examples of such constraints include that each employee is assigned to the same floor as his/her team lead unless the team lead is a third-level manager or above (in which case the team is too large to fit on one floor), and that the number of seats of a given type required by the solution does not exceed the number of available seats. The quality of the solution is measured in terms of objectives such as the total monthly rental cost, and the number of floors occupied by employees all belonging to the same area (both of which should be minimized).

There are two major steps in the creation of a specification of an optimization problem suitable for a solver such as IBM CPLEX®. First, it is necessary to define the appropriate decision variables. For example, the solution of the room-allocation problem can be formalized in at least two ways: (1) one binary variable for each employee and floor, denoting whether the employee is assigned to that floor or not; (2) one variable per employee, whose value is the floor to which that employee is assigned in the solution. The second step is formalizing the constraints and objective function in a way that satisfies the limitations of the particular solver engine. In the case of a linear-programming (LP) solver, the formalization would have to consist of non-strict linear (in)equalities. Often, the natural specification of the requirements of the optimization problem does not conform to these limitations, and would have to be transformed in various ways in order to be acceptable to the solver. For example, taking the integer part of a real variable is not a linear operation; however, it can be expressed through two inequalities. One of these would be strict, and would have to be massaged (by adding an appropriate epsilon value) to a non-strict inequality. In addition, the formulation of the problem will be different for different choices of the representation of the variables. Needless to say, carrying out such modeling in order to achieve a correct specification that can be solved efficiently requires very specific expertise.

To alleviate this problem, and make optimization technology available to a wider audience,we have developed an alternative technology, based on a common computational paradigm. Users start by writing a program in a conventional language such as functional subset of Python, to *verify* a given solution instead of specifying an optimization model for finding the optimal solution. Developers and data scientists who are not optimization experts should find the process of developing such a program familiar and convenient. Moreover, this program checks that the solution satisfies the problem constraints, and computes the value of the objective function achieved by the given solution. Such a program is already useful, because it can be used to evaluate and compare potential solutions, whether these are created manually based on experience, or by some heuristic approach. It is also helpful when trying to ensure that the formulation of

```
def assigned_offices(self, o1: OfficeType,
                     f1: Floor) -> int:
    return (math.ceil(self.occupancy(f1, o1)
            / self.get_office_info(o1)
              .max_occupancy))

@minimize
def cost_objective(self) -> float:
    return sum(self.assigned_offices(o1, f1)
               * self.get_office_info(o1).cost
               for o1 in self.all_office_types()
               for f1 in self.all_floors())
```

Figure 1: Python code that computes the rental cost in the room-allocation problem.

```
minimize sum (o1 in all_office_types,
              f1 in all_floors)
  assigned_offices[o1][f1]
  * get_office_info[o1].cost;

subject to {
  forall (o1 in set_of_all_office_types)
    forall (f1 in set_of_building_floors)
      assigned_offices[o1][f1]
        >= occupancy[f1][o1]
           / get_office_info[o1].max_occupancy
      && assigned_offices[o1][f1]
      <= occupancy[f1][o1]
        / get_office_info[o1].max_occupancy
        + 1 - 1e-10;
}
```

Figure 2: Generated specification for optimization solver.

the problem actually conforms to the business requirements, using well-known testing techniques.

Given this specification, our technology is able to automatically generate a model for a mathematical optimization engine. We therefore expect this to enable the much more widespread application and use of optimization in enterprises.

Example

An example of a verification program specified using our approach is given in Figure 1. This is an excerpt from a program that checks a solution to a room-allocation problem, in which employees are assigned to offices in different floors of a building under constraints related to the type of office each type of employee should be assigned to and the groups they work with. The figure shows how to compute the total rental cost, based on the number of occupied offices; this cost should be minimized, as indicated by the `@minimize` decorator.

The resulting automatic translation of this part of the program into a specification suitable for the IBM CPLEX® solver is shown in Figure 2. Several transformations had to be performed on the verification program in order to get this form, which is acceptable to the solver. One example of such a transformation is the `ceil` function, which cannot be used on decision variables, and needed to be transformed into two inequalities. The second of these is a strict inequality (less than), and needed to be converted into a non-strict inequality (less than or equal) by using an appropriate epsilon. Many more such transformations are needed in order to transform a full verification program into an optimization specification.

These transformation are implemented on top of an expressive mathematical representation, which contains mathematical expressions as well as logical operators, including quantifiers. This can support a variety of formalisms, including functional Python programs and optimization specifications. Converting an input formalism into this representation, and generating the output formalism from it, are relatively straightforward. However, the representation needs to be analyzed and transformed according to the requirements of the output formalism. This is done by two major technologies, which are built over the mathematical representation: (1) rewrite rules with a pattern language; and (2) constraint propagation over the AST and data-flow graphs. The transformations shown in the example above, and many others, are implemented as rewrite rules; the correct application of these rules requires, among other things, an analysis of the types and precise domains of the variables involved.

Demonstration

In our demonstration, we will begin by showing how a subset of the room allocation problem can be modelled using functional Python. We will then show how this code can be directly run to verify a given solution. Following this, we show the automatic translation of this specification into an optimization model in IBM's CPLEX® OPL optimization modeling language, and solve this model. Finally, we will take the output produced by the optimization model, and show how it can be verified in the functional Python specification.

Related Work

The Decision Guidance Query Language (DGQL) (Egge 2014) is a language for describing optimization problems from data descriptions; it is an extension of SQL. A research prototype converts problems written in DGQL into programs for IBM CPLEX®.

CVXPY (Agrawal et al. 2018) translates optimization specifications written in a domain-specific language (DSL) into specifications for a number of optimization engines. It analyzes the problem to find the best class of solver for it, and transforms the problem description into a form suitable for that solver.

Both DGQL and CVXPY use rewriting rules, in a way similar to our technology. However, the inputs to these systems cannot be used to independently verify a given solution, the way that the functional Python input to our system can, since both use non-standard languages (an extension of SQL for DGQL, and a DSL for CVXPY).

References

Agrawal, A.; Verschueren, R.; Diamond, S.; and Boyd, S. 2018. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1): 42–60.

Egge, N. E. 2014. *Decision Guidance Query Language (DGQL), Algorithms and System*. Ph.D. thesis, George Mason University, Fairfax, VA.