Generating Explanations for Mathematical Optimisation: Solution Framework and Case Study

Christina Burt and **Katerina Klimova**
Satalia, London
{christina,katerina}@satalia.com

Bernhard Primas
School of Computing, University of Leeds
scbjp@leeds.ac.uk

Abstract

In this paper, we address the problem of generating explanations automatically for mathematical optimisation. Explanations can improve the way users interact with optimisation tools and help them trust the solutions. One of the challenges of generating explanations for mathematical optimisation is to reconstruct meaning from abstract mathematical expressions. We present a general framework in which we exploit problem diversity exploration in order to infer meaning from algorithm results, and present an automatic sentence generator that works with this framework. Finally, we describe an industrial project where we applied these algorithms.

Introduction

In the field of mathematical optimisation, and particularly in academia, we seek simplified expressions of a problem class so that we can study its structure in the purest sense. However, real-world problems are often best expressed using both hard and soft constraints—meaning that some restrictions on the solution space are simply *preferences*. Additionally, it is common to find that real-world problems have multiple, typically competing, objective functions.

Finding a high-quality or even optimal solution is not the most important aspect for every real-world optimisation application. For some applications it is desirable to obtain several diverse solutions that come with explanations of their advantages and drawbacks such that they are understandable for a human user. It is in this spirit that we define the following general optimisation problem:

Given a set of problem inputs, determine the solution(s) that optimise a set of objectives such that a set of hard and soft constraints is satisfied.

In the context of explainability, we can extend the above problem as follows:

The output is a set of solutions with explanations.

In this paper, we study the problem of solving an optimisation problem where diverse solutions are useful, and exploit the diversity in order to obtain human-readable explanations for the user.

State-of-the-art mathematical optimisation algorithms fall short when it comes to providing adequate explanations to users. These algorithms have traditionally focused on finding a single solution with an optimal objective function value. Once such a solution is found, the algorithm returns the solution and terminates. The users are not provided with information to help them understand why the returned solution was considered 'best', or even if it is the *only* 'best' solution. Therefore, the optimisation problems we consider in this paper require extensions to existing technology.

The development of an algorithm with explanations is challenging for a number of reasons. First, explanations can be highly domain-specific. Mathematical optimisation algorithms, through the manner of encoding, tend to abstract away domain-specific information. This leaves a difficult task to reconstruct the relationship between constraints, variables and objectives in a meaningful way. Secondly, providing an explanation requires supportive and easily understandable sentence construction so that human users can understand them. This incorporates various difficulties including understanding what an explanation should and should not contain, and the fact that even solutions obtained by entirely theoretically well-understood algorithms may be hard to explain. As a consequence, a structural approach for finding diverse solutions is required to enable our framework to provide sensible and supportive explanations.

The contributions of this paper are as follows:

- *General explanation algorithm development*. We present a framework for eliciting explanation information from a diverse search space for mathematical optimisation methodologies. The algorithm aims towards providing rich explanation to users.
- *Implementation and illustration as a case study*. We illustrate the use of our framework for an industrial project as an anonymised case study.
- *Bridging the gap between mathematical optimisation and explainable AI*. Explainability is not well explored in the field of mathematical optimisation. This paper brings explainable AI to this field.

Background

For every application in the context of explainable AI, there is a set of functional abilities an autonomous agent should be able to provide. Some functional abilities may differ for different types of applications, whereas other abilities should be provided for an entire class of applications (Langley et al. 2017). For our application, we have identified three functional abilities that should be provided:

- *Rich explanation*. The framework should provide explanations for (a) the solutions chosen, (b) why the solutions were selected, (c) how many constraints and user preferences are satisfied or violated and (d) how good each solution is, i.e. what the objective function values are. If no solution is found, this should also be explained.
- *Understandable explanation*. Explanations provided should come in the form of full sentences understandable for a non-scientist that was not involved in the development process of the framework.
- *Interactivity*. The framework should be designed to allow the user to interact with the exploration of diverse solutions. This should enable the prioritisation of certain restrictions or objectives in a natural way.

(Fox, Long, and Magazzeni 2017) explain the need for explainable AI. Three key reasons have been identified that drive the motivation for explainable AI: trust, interaction and transparency. We explain their importance for our application as follows. If a scheduler considers using our framework in order to create a timetable, the scheduler needs rich explanation for every timetable suggested in order to *trust* the quality and meaningfulness of the suggested timetable. Creating a timetable is a step-by-step process, in which the user typically wants to fix the position of certain items first as the user considers them to be of high priority. This requires our framework to be designed such that it enables *interaction* with users. Finally, *transparency* is required for situations in which our framework makes a decision which is wrong or unexpected for the user so that the user can react accordingly.

(Smith 2012) discusses difficulties standing in the way of a more automated approach for which only minimal or no work at all is performed by a human. Examples include *oversubscription* (not all hard and soft constraints are satisfiable simultaneously), preferences (some goals are prioritised over others) or uncertainty (the input data is only partially known at the time of scheduling).

In related work, (Horiguchi, Tomoto, and Hirashima 2015) present a classification approach to components of a model in order to help students learn the appropriate application of the models. They create *model fragments* and use laws of physics to connect them. These connections lead naturally to explanations, which in turn can help students understand the processes better.

In our work, we will focus on reconstructing the connections between components of an optimisation model. The model itself can be formulated by

$$\max \{c^T x | Ax \leq b, x \geq 0, x \in \Omega\},$$

where $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ are vectors, $A \in \mathbb{R}^{m \times n}$ is the coefficient matrix and $x \in \mathbb{R}^n$ is the vector containing the variables. Additionally to constraints $Ax \leq b$ and $x \geq 0$, the set Ω is used to impose further restrictions on x. If no further restrictions are required, i.e. $\Omega = \mathbb{R}^n$, the problem is called a *linear program*. For the special case in which only integer values are allowed, i.e. $\Omega = \mathbb{Z}^n$, the problem is called an *integer linear program*. If only some (but not all) variables are required to be integer, the problem is called a *mixed-integer linear program*.

In the context of linear programming, sensitivity analysis is a methodology that can be used to generate rich explanations. Using this methodology, one can automatically generate information such as (Winston and Goldberg 2004):

- whether the optimal solution is unique or not;
- how much the coefficient vector c in the objective function can vary before a different solution becomes optimal;
- how much the right-hand-side vector b can vary before a different solution becomes optimal.

The theory of duality provides these results for linear programming. However, for mixed-integer programming, no such duality theory, and explanation pathway, exists. The closest concept available for understanding infeasibility is the one of Irreducible Infeasible Sets (Chinneck 2008). An IIS represents a smallest set of constraints that cannot be satisfied simultaneously, i.e. removing any of these constraints would make the remaining set of constraints satisfiable. Only for linear programming, there exist exact methods for determining the IIS. However, for mixed-integer programming heuristics can be used such as, simply, removing constraints one at a time in order to discover a feasible set.

General Approach

In order to solve the problem presented in the previous section, we follow an intuitive and effective heuristic in Algorithm 1. We provide an illustration of the algorithm in flow chart form in Figure 1.

Algorithm 1 Detecting diverse solutions.

```
 1: $\mathcal{H}$ ... set of hard constraints
 2: $\mathcal{S}$ ... list of soft constraints
 3: $\mathcal{O}$ ... set of objective functions
 4: $\mathcal{C} \leftarrow \mathcal{S} \cup \mathcal{H}$
 5: for $i = 0 : (|S| - 1)$ do
 6:     Check if constraints in $\mathcal{C}$ are satisfiable
 7:     if $\mathcal{C}$ is not satisfiable then
 8:         Record explanation (1)
 9:     else if $\mathcal{C}$ is satisfiable then
10:         for $o \in \mathcal{O}$ do
11:             Solve, using constraints $\mathcal{C}$ and objective $o$
12:             Record explanation (2)
13:     Remove $S[i]$ from $\mathcal{C}$
```

Explanation (1), in its simplest form, is *"No solution exists, constraint set $\mathcal{C}$ is too restrictive"*. Explanation (2) is of the form *"An optimal solution exists for objective o, all constraints are satisfied from constraint set $\mathcal{C}$"*.

Removing constraints one at a time allows us to logically infer causality. Of course, it is possible to take this further. In the decision hierarchy described, only the last constraint that created infeasibility provably causes the infeasibility. Therefore, we could allow those constraints eliminated earlier in the process to re-enter the constraint set. The usefulness of such an exhaustive analysis depends on the application of the explanation generator. The richness of the 'infeasible' explanation can be enhanced by starting with the most relaxed constraint set and adding constraints one at a time, in-

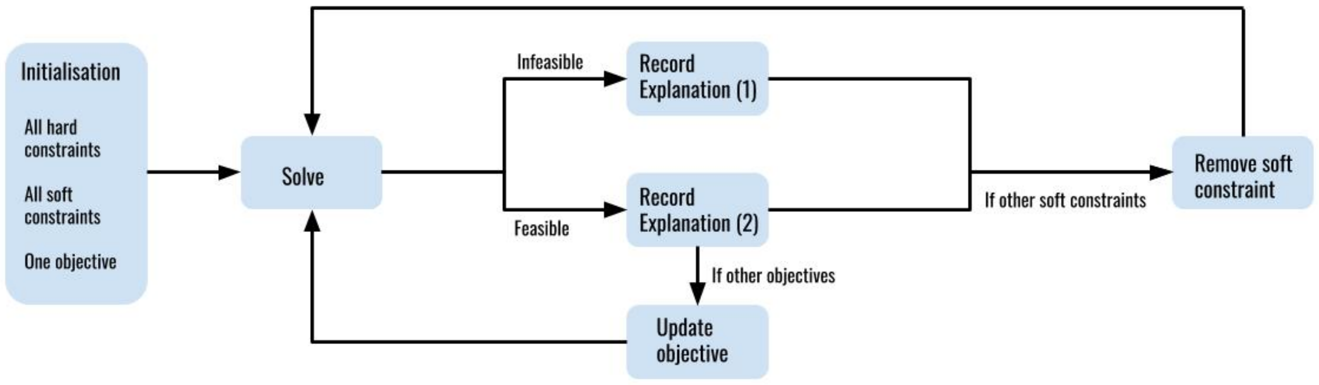

Figure 1: A flow diagram illustrating the search strategy in Algorithm 1

stead, as it allows us to pinpoint the precise constraints that created the present infeasibility in the system.

The incorporation of soft constraints can be used to achieve particular goals. To this end, an order is assigned to the soft constraints. For example, starting with soft constraints we expect to be most restrictive allows us to find infeasible solutions sooner, and therefore eliminate those constraints that lead to infeasibility. On the other hand, starting with soft constraints we expect to be least restrictive allows us to potentially satisfy more soft constraints and therefore achieve more preferences for the user. The implication of ordering soft constraints can lead to richer explanations if the context of the constraints is well understood. In mixed-integer programming, this cannot be automatically derived from a model, but can be achieved with domain knowledge.

Since there is no sufficient sensitivity analysis theory for mixed-integer programs, it is challenging to obtain relationships between constraints, variables and the solution space. What we achieve in Algorithm 1 is a mimicking of sensitivity analysis. We consider the set of hard constraints and start by adding all soft constraints. In most cases, it is not feasible to satisfy all constraints simultaneously. Armed with this information, we explore the ways in which soft constraints interact with the feasible region, and seek diversity in our solutions by trialling different combinations of objectives.

Achieving diversity in this controlled manner is an important part of deriving explanations. Since we can usually measure the quality of solutions and compare them against one another, it is useful to not only have diverse solutions but also to have the precise reasons for this diversity. This construction of diversity leads directly to richer explanations.

Allowing the user to select preferences, i.e. soft constraints, is also important. Often, the user is the expert and will pre-filter the constraints to a more interesting sub-set within the context. In this way, the unsophisticated and most uninteresting part of the exhaustive search is replaced by inference from the user.

Automatically Generating Sentences

Algorithm 2 presents an illustration of how we achieved explanations in sentence form. The main arguments are:

- P is a `boolean` that denotes whether there exists a solution without any constraint relaxation.
- If P is `true`, each constraint in the constraint set $\mathcal{C}$ has assigned a boolean indicating if it was applied in the solving round or not.
- *objective* is a `string` stating the objective function currently under consideration.
- Finally *item* contains information about the type of the object we are creating the solution for.

The algorithm starts with a check whether the perfect solution has been found. In that case, the explanation message states that the solution fulfils all constraints and is the best among others. In the case that there is no solution for the full set of constraints the algorithm starts the explanation message based on the type of objective function. It then continues with the information that this solution does not satisfy some of the constraints, and names these constraints in a user-friendly way. The list of unsatisfied constraints is then added and the explanation message is returned.

Case study

In this section we describe a pilot client project, for which we created an interactive scheduling tool. To protect our client's identity, we describe the project using an analogue. The essential features of the problem and solution, with respect to the topic of this paper, are intact.

Our study case is based on the problem of scheduling different free time activities in a leisure centre. The centre offers a range of different activities for all age categories, starting with preschool kids and ending with courses for adults. Each of the age categories has a specific time range during the day and also different activities. In our project, we focused on scheduling activities for the two youngest age groups: preschool kids and primary-school kids.

The preschool group has lessons planned between 1 pm and 5 pm and a primary-school group from 1 pm to 7 pm,

Algorithm 2 Automatically generating explanations

```
Require: $\mathcal{C}$, $P$, $item$, $objective$
 1: if not $P$ then
 2:     explanation ← "There are no < item.type > that account for < all constraints from $\mathcal{C}$ >."
 3: else
 4:     explanation ← "This <item.type> is the most recommendable in terms of < objective > and <full set of hard constraints>"
 5:     for constraint ∈ $\mathcal{C}$ do
 6:         if not constraint.applied then
 7:             nonActiveList ← constraint
 8:     for constraint ∈ nonActiveList do
 9:         explanation ← ", but may not satisfy"
10:         if first constraint then
11:             explanation ← " <constraint>"
12:         else if last constraint then
13:             explanation ← " and <constraint>"
14:         else
15:             explanation ← ", <constraint>"
16:         explanation ← " restrictions."
    return explanation
```

from Monday to Friday. These hours have been set by the leisure centre with regards to kids age and school attendance.

The leisure centre offers three types of lessons. The first category accommodates all physical activities such as swimming, dancing and yoga. The second category contains all the creative activities starting with drawing classes and ceramics and ending with musical education. The third category is formed by educative courses strongly represented by language classes of different levels. Some of these activities are accessible for kids of all ages, but some, such as playing the accordion, are only for 6 years and older, and vice versa.

Some of the classes are fixed in the schedule from the beginning due to the specific requirements. The rest of the classes are then planned with respect to many constraints. The constraints include activity continuity for the instructors when the leisure centre aims to plan the activities with same requirements for different levels in a block. This continuity feature is one of our soft constraints. Another soft constraint is then the aspiration for balanced schedules and balanced offers from all activity types. Hard constraints were mainly driven by the target age and facility availability.

Our project was split into three types of scenarios. The first scenario is to recommend the activity for a chosen time slot. The idea behind this scenario is to help the scheduler choose the right activity with respect to all or as many as possible fulfilled constraints. The second scenario was then to recommend the best slot for given activity. This describes the situation when some of the lectures are for example sponsored by a council, therefore, the scheduler needs to schedule these activities and our tool recommends best slots and explains why these sets are the most preferable. The last scenario is then focused on the themed weeks. These weeks are not necessarily only pre-Christmas week, pre-Easter week and other holiday-specific weeks, but also weeks with a general theme like 'sea life'. In these weeks the leisure centre plans all lessons which are related to the theme such as drawing sharks or saying "fish", "boat" or "sand" in other languages. In this scenario, the tool recommends the whole sequence of classes related to the theme.

Implementation

The implementation of the problem is composed of two parts. The first is a wrapper which we built around an optimisation solver to get results efficiently and with the explanation. This part follows Algorithm 1 and is described further below. The second consists of Satalia's SolveEngine solver and supporting functions. After solving is completed, the results are passed back to the wrapping function which evaluates the results and takes one of the following actions:

- If solver did not find any solution, one of the active constraints is deactivated and the problem is passed again to the solver.
- If the solver found a solution but more variants of the solution are needed, the chosen activity or time slot is removed and the problem is passed back to the solver.
- If there are enough diverse solutions or no new solution can be found and no further constraint relaxation is possible, the search is terminated.

After we have all solutions which we need or can obtain, we collect all explanations created for each solution using Algorithm 2 and return them as a list of tuples containing recommended activity or time slot and explanation. In the project, we were given different *item.types* so that we extended the algorithm accordingly in order to keep the explanation meaningful.

We added additional functionality to the tool, such as automatically generation of tweets to announce new activities. The users interacted with the pilot recommendation tool via an interface. Our feedback was very positive for the use of real sentences and explanations, as these were features the users could readily understand. In fact, they appreciated these elements far more than the optimisation algorithm under the hood, which highlights the importance of explanations of abstract algorithmic results for users.

Conclusion

We have presented a framework for automatically generating explanations about the search space for the user to interpret. In a case study, we communicated this information to the users by automatically generated sentences. Our approach relies on the user to interpret the relationship between model components from the information we present as an explanation. By utilising this domain knowledge of the user, interpretations of the model components can be automatically encoded on an application basis. We continue to explore this aspect in our ongoing projects.

Acknowledgement

The tool for the case study was created by a larger project team at Satalia including Harry Edmonds, Yohann Pitrey, Alex Lilburn, Vito Tumas and Daniel Hulme.

References

Chinneck, J. W. 2008. *Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods*. Springer US.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. In *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*, 24–30.

Horiguchi, T.; Tomoto, T.; and Hirashima, T. 2015. A framework of generating explanation for conceptual understanding based on "semantics of constraints". *Research and Practice in Technology Enhanced Learning* 10(1):1–21.

Langley, P.; Meadows, B.; Sridharan, M.; and Choi, D. 2017. Explainable Agency for Intelligent Autonomous Systems. In *Twenty-Ninth AAAI Conference on Innovative Applications (IAAI-17)*, 4762–4763.

Smith, D. E. 2012. Planning as an Iterative Process. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2180–2185.

Winston, W. L., and Goldberg, J. B. 2004. *Operations Research: Applications and Algorithms*. Thomson Brooks/Cole.