A Spreadsheet-like User Interface for Combinatorial Multi-Objective Optimization

Derek Rayside
MIT CSAIL
drayside@csail.mit.edu

H.-Christian Estler
University of Paderborn
estler@upb.de

Abstract

We present Moolloy, a general purpose, spreadsheet-like user interface for combinatorial multi-objective optimization. Current user interfaces for multi-objective optimization tend to either require some programming experience to use, or are narrowly focused on specific problems. Consequently, multi-objective optimization is usually only used by sophisticated technical workers, such as aerospace engineers. Moolloy makes multi-objective optimization accessible to a larger set of potential users.

1 Introduction

In a single-objective optimization problem, a set of decision variables are assigned values from a given domain; a solution is an assignment for which the specified constraints hold. The optimal solution is the solution with the best value, computed by applying an objective function to the assignment.

A *multi-objective optimization problem* (MOOP) is an optimization problem with several, oftentimes conflicting, objectives. In the design of a bicycle, for example, cost and performance conflict. The decision variable *frame material* might take one of the values *Aluminum* (for high performance but high cost) or *Steel* (for lower performance but lower cost).

In most cases, a MOOP does not have a single optimal solution, but a set of optimal solutions. For the bicycle problem, a range of solutions that balance cost and performance in different ways might be obtained. These solutions are optimal in the sense that the value of one objective can be raised only by lowering the value of another. Furthermore, no solution in this set is *dominated* by any other solution in the set, meaning that for each pair of solutions $\langle s_1, s_2 \rangle$, s_1 has at least one objective-value that is better than s_2's corresponding objective-value. Named after the economist Vilfredo Pareto, the set of non-dominated solutions is often called the *Pareto front*.

Single-objective optimization is supported by common end-user tools (such as spreadsheets) and is widely used as a decision-making aide. The same cannot be said for multi-objective optimization: tools that support it tend to require programming expertise, and their use is most often constrained to highly

Copyright © 2009 Derek Rayside and H.-Christian Estler. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

skilled professionals in specialised technical domains, such as the aerospace industry.

Scaffidi et al. [16] estimate, based on data from the US Bureau of Labor Statistics, that by 2012 over 90 million Americans will use computers at work. Of these, 55 million will be spreadsheet and database users; 13 million will do some programming, and fewer than 3 million will be professional programmers. We would like to expand the potential user-base for multi-objective optimization from its current position of those with some programming experience to all spreadsheet users.

We imagine that multi-objective optimization will become more important for regular business people as concerns for the environment, social responsibility, life-cycle costs, safety, *etc.*, compete with regard for the short-term bottom line.

A similar transformation occurred in the American aerospace industry in the 1970s [17]: in the 1960s the aerospace industry pursued performance at any cost; by the 1970s they realized that each marginal increase in performance was coming at larger and larger costs, both to the short-term bottom line, as well as to maintainability and life-cycle costs. Hence the aerospace industry has been interested in multi-objective optimization for some time now. However, the tools they use tend to be restricted to those with some programming experience (which is reasonably common amongst the highly trained engineers who design aircraft).

We have been collaborating with a group of aerospace engineers here at MIT, and have studied the kinds of multi-objective optimization problems that they are most commonly interested in solving. In previous work [15] we developed a new algorithm for multi-objective optimization and applied it to solving some of their complex challenge problems. This algorithm uses the Kodkod [20] relational model-finder as its underlying constraint solver.

In this work we report on a novel user interface that we have designed and implemented based on (a) our study of the kinds of models that they write with their existing tools, and (b) interviews and design reviews with them. From this collaboration and study we posit that:

1. the idea of multi-objective optimization is comprehensible to regular spreadsheet users;

2. a spreadsheet-like user interface can be designed for multi-objective optimization;

3. such a user interface will make multi-objective optimization more accessible.

In this paper we concretely substantiate the second proposition by describing the design of a spreadsheet-like user interface for discrete multi-objective optimization. We provide arguments supporting the first and third propositions but leave their empirical exploration for future work.

The prototype of our user interface and solver may be downloaded from `http://sdg.csail.mit.edu/moolloy/`

2 Problem Statement

This section gives a formal description of combinatorial multi-objective optimzation, Pareto dominance, Pareto optimality, *etc.*: the concepts that are needed to understand what our user interface is intended to do.

2.1 Problem Input

In a multi-objective optimization problem, a vector of *decision variables* $\vec{X} = [x_1, \ldots, x_z]$ is assigned a vector of *values*, called an *assignment*. Each value is drawn from a given domain, thus we sometimes refer to it as *domain value*. An assignment is *feasible* if it respects all the constraints represented by a vector $\vec{C} = [c_1(\vec{X}), \ldots, c_p(\vec{X})]$. A *feasible assignment* is also called a *solution*. A vector of metric (or objective) functions $\vec{M} = [m_1, \ldots, m_q]$ is applied to a solution to obtain a *point* (or metric values or objective values) $[m_1(\vec{X}), \ldots, m_q(\vec{X})]$.

2.2 Pareto dominance and Pareto optimality

Two solutions can be compared based on their metric values. We make the following distinctions:

Definition 1 *Let $\dot{a}$ and $\hat{a}$ be solutions, q the number of metric functions, and let u, v be metric function indices in $\{1, \ldots, q\}$. We say*

- $\hat{a}$ (Pareto) dominates $\dot{a}$ *with respect to the metric M:* $\Leftrightarrow \forall u : m_u(\hat{a}) \geq m_u(\dot{a})$ *and* $\exists v : m_v(\hat{a}) > m_v(\dot{a})$
- $\hat{a}$ (Pareto) equals $\hat{a}$*:* $\Leftrightarrow \forall u : m_u(\hat{a}) = m_u(\dot{a})$

Given a set of solutions S, we are interested in finding *maximal* or *optimal* solutions (minimization problems can be expressed as maximization problems). Optimality is defined in terms of metric values:

Definition 2 *Let $\dot{a}$, $\hat{a}$ be solutions.*
We call $\hat{a}$ maximal *or* (Pareto) optimal *iff no $\dot{a}$ exists such that $\dot{a}$ dominates $\hat{a}$. The set containing all optimal solutions is called the* Pareto front*.*

2.3 Solving a Multi-Objective Optimization Problem

The result of solving a multi-objective optimization problem is its Pareto Front. Conventionally, the Pareto front is regarded as a set. In practice, however, an algorithm produces one solution at a time. The Pareto front may be so large that the user cannot wait for all optimal solutions to be generated; or the user may simply wish to start assessing and exploring solutions as they are generated.

It is therefore important to specify a solver in terms of the sequence of solutions that it produces, and not the set (which may never be obtained).

A MOOP solver has three essentual properties:

Definition 3 ***Specification of a MOOP solver***

Given decision variables $\vec{X}$, metric functions $\vec{M}$, and constraints $\vec{C}$. Let S be the set of all solutions for $\langle \vec{X}, \vec{M}, \vec{C} \rangle$. A solver should produce a sequence of assignments O such that:

i) Soundness*: Every generated assignment satisfies the constraints (i.e. is a solution):*
$\forall a \in O \mid \bigwedge c_i(a),\ 1 \leq i \leq p$

ii) Optimality*: Every generated assignment is optimal:*
$\forall a \in O \mid a$ *is Pareto optimal*

iii) Completeness*: Every optimal assignment is generated:*
$\forall a \in S \mid a$ *is Pareto optimal* $\Rightarrow a \in O$

It is up to the MOOP solver to satisfy the 3 criteria given above and the design of a user interface can be seen as separate task. Nevertheless, only with a solver which computes correct solutions and yields them as early as possible during the computation, we are able to build interfaces with high usability which fulfill today's criteria for human-computer interactions, e.g. "provide feedback" [12].

2.4 Problem domains of MOOPs

In different research communities, the vectors $\vec{X}$, $\vec{M}$ and $\vec{C}$ take different forms. In *Operations Research*, for instance, the decision variables are usually assigned numerical values, either drawn from a continuous domain like $\mathbb{R}$, or from a discrete domain like $\mathbb{Z}$. Likewise, constraints are given as equations or inequations over $\vec{X}$ (e.g. as linear functions).

For so called *pseudo boolean* problems, $\vec{M}$ and $\vec{C}$ are used in the same manner as just described, but decision variables are restricted to take the values 0 or 1.

The solver underlying our interface is different. Addressing the solver directly over its API, we can specify domain values which can themselves be complex structures (though finite and discrete). Thus the topology of a network or the shape of a directory hierarchy, for example, might be the value assigned to a single variable. For the graphical user interface, however, we tried to identify the subset of optimization problems which are most relevant in practice. This subset is described in more detail in the next section.

3 Moolloy - an Interface for MOOP

We introduce the interface of our tool, called Moolloy, by further pursuing the bicycle example from the introduction. A number of vari-

ables with their associated domain values describe the characteristics of a bike. Our goal is to find all Pareto optimal solutions: *i.e.*, those on the maximal performance and minimal cost trade-off curve.

3.1 Declaring Metrics

Figure 1 shows the default window directly after the start of the Moolloy tool. On the left hand side, a tree with with 4 nodes is provided. Using context menus, we add new nodes to this tree; each node representing parts of the optimization model. Therefore, we also call the tree a *model-tree*.

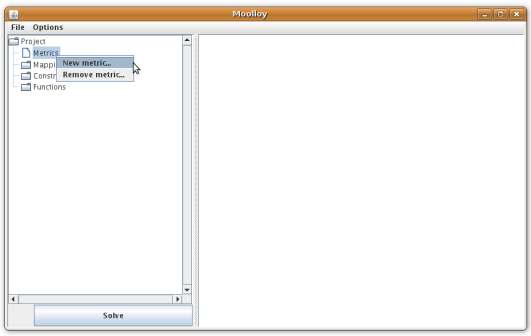

Figure 1: Moolloy's main window with an empty model-tree.

Usually, the first step during the development of an optimization model is to state the metrics of the problem. For our example, we define the two metrics *Cost* and *Performance* as shown in Figure 2. We set the preference of metric *Cost* to "min" and the preference of metric performance to "max". The operator determines if metric values are combined with addition or multiplication. We discuss the columns "Min" and "Max" in the next section.

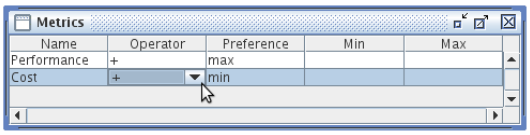

Figure 2: Declaring metrics of the model.

3.2 Declaring Domain Values

In a next step, we declare domains and their values (shown in Figure 3). Note that the ordering in which domain values are listed is meaningful. Internally, we define a total ordering over the values of each domain; *i.e.*, in our example $Steel < Aluminum < Carbon\_fiber$ holds. Such an ordering is useful once we add constraints to the model.

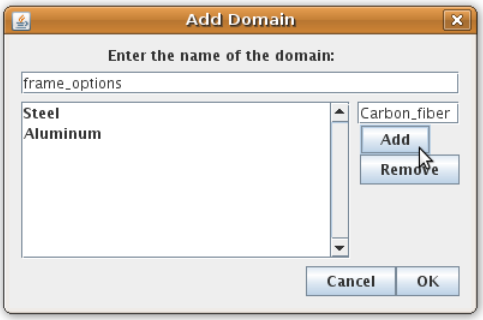

Figure 3: Adding a domain and its values. The ordering of the values describes a total ordering over the values of the domain.

As different decision variables, e.g. *Frame* and *Fork*, will take different domain values, we can declare multiple domains. Later on, we assign the decision variables to their corresponding domains. Each declaration of a domain adds a new node (a child of the mapping node) in the model-tree.

3.3 Declaring Variables

We can now add decision variables to each domain of our model. This is, again, done by using the context menu of the model-tree. Each variable creates a new node in the tree. Opening such a node, *e.g.*, for the variable *Frame*, opens a table as shown in Figure 5. Using this table we specify how a particular domain value changes a metric value (in case the variable gets assigned to the domain value).

It might happen that some variables should not be able to get assigned to particular domain value. This can be achieved by simply leaving the row of that value blank. In the example: if we don't enter at least one metric value in the row of *Carbon_fiber*, the variable

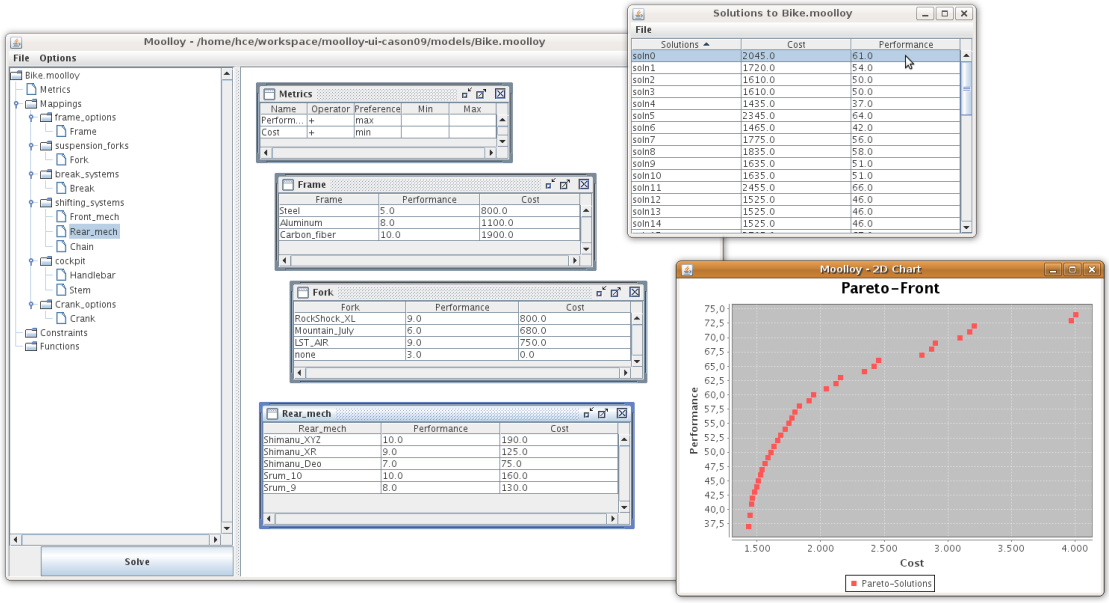

Figure 4: The entire model-tree, the solution window and the plot of the Pareto front.

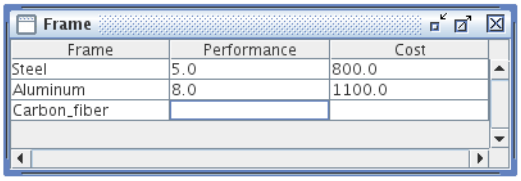

Figure 5: For each variable we declare the influence of a domain value on the metric value.

Frame will never be assigned to this domain value. Rows which are only partially filled with metric values, are automatically completed by adding a neutral element (0 for addition, 1 for multiplication) in the empty fields.

3.4 Solving the Model

Once we have completed our model, we solve it in order to get the Pareto optimal solutions (the Pareto front). Figure 4 shows the main window with the complete model-tree for our bicycle example.

A separate window displays a table with all the Pareto optimal solutions, showing their metric values (see Figure 6). Clicking on one of these rows will open up a new window displaying the details of the the solution, i.e. the assignments of variables to domain values (see Figure 7).

To visually inspect the Pareto front of our model, we can plot two metrics in a diagram. For our bicycle example, we can observe that there is high density of solutions with low costs and low to medium performance but only a few solutions with excellent performance. This might indicate that we have not enough medium-price, medium-performance options in our model.

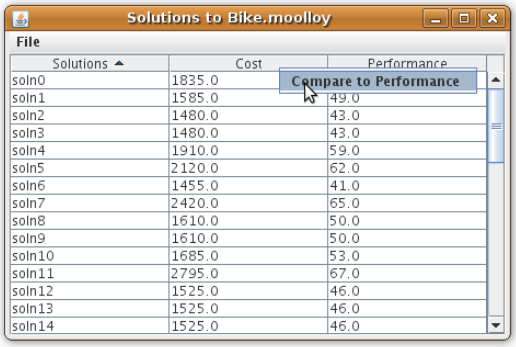

Figure 6: Pareto optimal solutions to the model. The Pareto front can be plotted using the context menu.

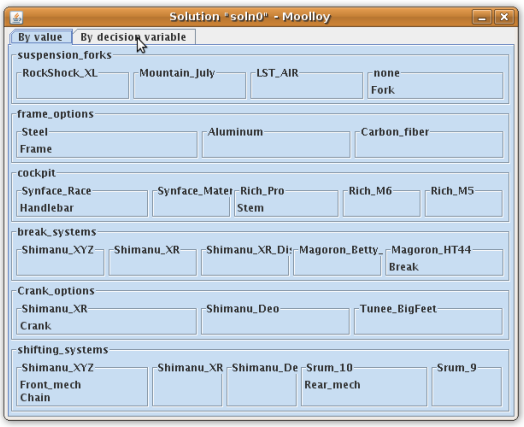

Figure 7: The assignment window displays the detail of a particular solution.

4 Expressing Constraints

Once the decision variables, domain values and metrics of an optimization model are defined, a user might want to add constraints (also called restrictions) to the model. Constraints ensure that every solution respects certain properties. In our bicycle example, for instance, we might want to ensure only a certain fork model is used, given that the frame material is aluminum. Thus, a constraint might be: *if frame = aluminum then fork = RockShock_XL.*

4.1 Classification of Constraints

We identify three major types of constraints:

1. *Assignment constraints*: this type of constraint relates to the example given above. The assignment of a decision variable to a certain value influences the assignment of other decision variables.

2. *Metric constraints*: a metric constraint ensures that a solution respects certain metric properties. For example, we could formulate a constraint *"overall performance of a solution must be > 40"* for our bicycle model. At a more fine-grained level, a metric constraint might not just restrict the overall value of a metric but instead a specific part of the model: e.g. *"cost of handlebar + stem must be ≤ 250"*.

3. *Functional constraints*: given a decision variable which is assigned to a domain value, a metric value for this assignment might depend on other decision variables. Again, we give an example from our bicycle model: the performance of the *rear mech* (the rear gear shifter) depends on the type of chain chosen. A constraint could be *"if chain = Shimanu_XYZ then the performance of the Shimanu_XYZ rear mech equals 10, else it is 8"*.

We do not claim that every constraint of an arbitrary combinatorial MOOP can be matched to one of the three categories. However, our work with real world case studies and a variety of (sometimes artificial) benchmark problems has shown that most constraints fit within this classification.

During the development of the interface, our cooperation with aero-space engineers revealed that *assignment constraints* and *metric constraints* are essential to model (even simplified versions of) real-word optimization problems. In contrast, *functional constraints* are of greater importance for the fine adjustment of the optimization model.

4.2 Implicit and Explicit Expression of Constraints

The Moolloy user interface is structured so as to make the most common kinds of constraints that users wish to express as simple as entering a value in to a cell in a table. We refer to such constraints as *implicit* because they do not require the user to write an explicit logical formula.

We gave an example for such an implicit constraint in Section 3.3: by not defining any metric values for a particular domain value, we implicitly stated a constraint which prohibits any variable from being assigned to this domain value.

Another possibility to state an implicit constraint is shown in Figure 2. Entering a value in the "Min" field ensures that all solutions will have a value for this metric that is greater than or equal to the specified value. Entering a value in the "Max" field ensures that all solutions will have a value for this metric that is less than or equal to the specified value.

```
constraint ::= '(' constraint ')'
   | '@' classId
   | negationOp constraint
   | constraint logicOp constraint
   | formula
   | logicConst

negationOp ::= 'not' | '!'

logicOp ::= 'and' | '&'
   | 'implies' | '=>'
   | 'iff' | '<=>'
   | 'or' | '|'

logicConst ::= 'true' | 'false'

formula ::= expr formulaOp expr
   | varId formulaOp (varId | valueId)
   | valueId formulaOp varId
```

```
formulaOp := '=' | '<' | '>' | '<=' | '>='

expr ::= '(' expr ')'
   | expr exprOp expr
   | sumExpr
   | multExpr
   | intConst

exprOp ::= '*' | '/' | '%' | '+' | '-'

sumExpr ::=
   '$sum(' metricId ',' domainId ','
           valueId ',' number ')'

multExpr ::=
   '$mult(' metricId ',' domainId ','
            valueId ',' number ')'

intConst ::= '$'number
```

Figure 8: Grammar of Moolloy's expression language: every name ending with *Id* represents an identifier, and is to be treated as a terminal. The terminal *number* represents a positive integer value. Precedence of logical operators (tightes first): *not*, *and*, *implies*, *iff*, *or*. Expression operators * (multiplication), / (division) and % (modulo) have higher precedence than + (addition) and - (substraction). All operators associate to the left.

In a similar fashion, we can assure that a minimal or maximal number of variables get assigned to a particular domain value. Figure 9 shows the fields "MinVars" and "MaxVars" which are used for this purpose.

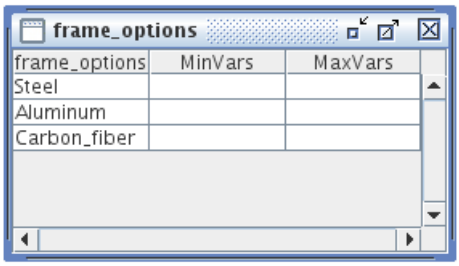

Figure 9: "MinVars" and "MaxVars" allow a implicit definition of a constraint.

Sometimes, however, the user wishes to express constraints that do not correspond to a single table cell in the user interface. In this uncommon case the user may write an explicit logical formula, such as the example shown in Figure 10. In the next section we describe the grammar for these explicit constraints.

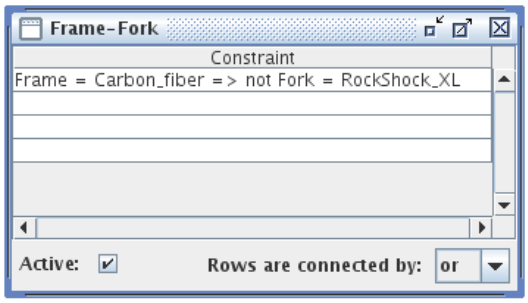

Figure 10: Explicit definition of a constraint using the expression language.

4.3 Explicit Constraints

Figure 8 shows the grammar of the expression language currently implemented in Moolloy. As an example, we show how to express the constraints from the beginning of this section in Moolloy's expression language:

- the constraint *if frame = aluminum then fork = RockShock_XL* gets almost directly translated to: `frame = aluminum => fork = RockShock_XL` (where `frame`

is a `varId` and `aluminum` is a `valueId`).

- the metric constraint *"cost of handlebar + stem must be ≤ 250"* can be translated to: `$sum(Cost, cockpit, Syntace_Race, 4) <= $250`. The expression specifies that the sum of `Cost` for the variables in the domain `cockpit` (these variables are handlebar and stem) has to be less than 250. We specified that every domain value from the cockpit should be considered in the calculation because `Syntace_Race` is the first value in the domain `cockpit` which has 4 additional domain values. The last two parameters of the `$sum()` expression define the range over which the calculation is performed. If, for instance, instead of `Syntace_Race, 4` we would write `Snytace_Race,0`, then only variables assigned to the value `Syntace_Race` would be considered in the calculation.

We already mentioned that Moolloy's underlying constraint solver [15] has a greater expressiveness than what is expressible within the interface. The same accounts for the expression language. While our experience shows that most user constraints can be handled within Moolloy itself, there might be case where a constraint can not be expressed due to the limitations of the expression language. In such a case, an experienced user can write the constraint directly as a Kodkod input (i.e. as a Java class) and refer to it in the interface using `@classId`, where `classId` represents the full name of the class. The class will then dynamically be instantiated and invoked before solving the model.

5 Related Work

There is relatively little work on end-user interfaces for combinatorial multi-objective optimization that we are aware of.

Most tools that do multi-objective optimization have interfaces like Matlab (which includes some evolutionary algorithms for multi-objective optimization). These interfaces are intended for a technically sophisticated audience, and are usually targeted for problems with continuous variables and few constraints. By contrast, we are interested in developing a user interface for spreadsheet-level users who wish to solve multi-objective optimization problems with discrete variables with many constraints.

Catalyst Development Corporation (`http://catalyst.com`) is currently running a private beta test of their forthcoming ChoiceAnalyst tool, which has similar objectives to ours: combinatorial multi-objective optimization for spreadsheet-level users. Our prototype has a richer and more general language for expressing constraints. ChoiceAnalyst has an interesting idea of having not only multiple objectives, but multiple decision makers. The tool attempts to find the solutions that are good not only on all metrics but also for all decision makers. This is a very interesting dimension that we have not yet explored: we assume a single decision maker with multiple objectives.

Blasco et al. [1] and Eskelinen et al. [7] have developed some techniques for visualizing Pareto fronts in high-dimensional spaces. These techniques are complementary to ours, and we discuss them more in the Future Work section below.

Our aerospace collaborators here at MIT have largely been developing their own user-interfaces and solving algorithms [11, 18]. Their most fully developed user-interface and associated solver is the Object-Process-Network (OPN) tool [11] (`http://opn.mit.edu`). One may think of OPN as Petri-nets annotated with short procedures written in Python. This is a very popular tool with our collaborators for a few reasons:

1. It is very easy to draw the Petri-net-like structure of the model.

2. It is relatively easy to write short procedures within a framework – as compared to writing complete programs. These users have the technical sophistication to write short procedures within a framework, but find it inconvenient to write complete programs.

3. OPN is Turing complete, and hence capable of modelling a wide variety of problems – not just multi-objective optimization. For example, they also use OPN to

model stakeholder value-flow feedback networks.

The OPN user-interface makes it relatively easy for technically sophisticated users to model a wide variety of problems. Our goal is to take one of these problems that OPN is commonly used for – combinatorial multi-objective optimization – and make it accessible to regular business people who are familiar with spreadsheets but not programming.

A good introduction to the literature on multi-objective decision making is the collection of surveys edited by Figueira et al. [8]. More focused surveys of the literature on multi-objective combinatorial optimization include those by Ehrgott and Gandibleux [3–5, 9] and by Ulungu and Teghem [21]. Additionally, the Annals of Operations Research recently ran a special issue on combinatorial multi-objective decision making [6].

6 Future Work

We are pursuing future work along a number of fronts, including enhanced user input of expressions with schematic tables [2], improved visualization of the computed Pareto fronts, and usability studies with our implemented tool.

6.1 Schematic Tables

The goal of the work reported in this paper was to move the expression of combinatorial multi-objective problems from imperative code to a functional (*i.e.*, stateless) spreadsheet-like user interface. This has been accomplished. The next step down this path is to transfer formulas from the textual expression language to a more graphical and tabular form.

Schematic tables [2] are a new form of decision tables [14, 19] developed by Edwards [2] that we could use to supplant some usages of our expression language. Schematic tables have a number of useful features:

- Canonical form: there is only one way to express each meaning.
- Proper partitioning: a decidable logic ensures that the table covers the input space both exhaustively and disjointly.
- Editability: schematic tables have a systematic way that they can be put into an inconsistent state while the user is editing the table, and the user is given feedback about how to continue editing the table to resolve the introduced inconsistencies.

Peyton Jones et al. [13] also developed a more tabular approach to user-defined spreadsheet functions. These ideas are actually more similar to the way that regular spreadsheet users work than are decision tables or schematic tables. However, the approach of Peyton Jones et al. [13] does not offer any of the discipline of schematic tables: for example, ensuring that the function covers all of its inputs exhaustively and disjointly. Such discipline is important in this domain, and so we think that schematic tables are the best choice here.

6.2 Visualizing the Pareto Front

The focus of our work reported here has been on a spreadsheet-like user interface for describing combinatorial multiobjective problems: *i.e.*, on the input to the solver. We have not emphasized here the user interface for inspecting the computed Pareto front: *i.e.*, the output of the solver.

Blasco et al. [1] and Eskelinen et al. [7] have recently proposed techniques for visualizing Pareto fronts – especially fronts in spaces with more than two dimensions. Pareto fronts in two-dimensional spaces are easily visualized in the obvious way, as we already do (*e.g.*, Figure 4). It is less obvious how to visualize Pareto fronts in high-dimensional spaces. However, since our solver [15] is designed to work in high-dimensional spaces, we are very interested to incorporate appropriate visualizations for the resulting Pareto fronts.

6.3 Usability Studies

While we have conducted user interviews and mock-up walkthroughs in the work reported here, we have not yet conducted quantitative usability studies with the working tool. Three particular issues that we are interested in exploring empirically include:

1. Whether users prefer the problem space hierarchy organized around domains (as it is currently the case), or around decisions.

2. Which kinds of functions users prefer to write formulæ for, and which they prefer to use schematic tables for. The results of previous usability studies (*e.g.*, [10, 22]) do not obviously make predictions of what users will find easier in our context.

3. Whether the problems that other classes of business and engineering users are interested in can be expressed within our user interface. We are currently collaborating with some materials scientists here at MIT who are also interested in a multi-objective approach to planning manufacturing.

7 Conclusions

Multi-objective optimization is likely to become more important to business people wishing to make rigorous decisions in complex spaces.

In the 1970s the aerospace industry evolved from a single-minded focus on performance to a richer view including maintainability, life-cycle costs, *etc.* [17] Business is now evolving from a single-minded focus on the bottom-line to a richer view that includes environmental concerns, social responsibility, *etc.*

The aerospace industry has highly technical staff that can make use of sophisticated tools such as Matlab in their decision making. Regular business users need better user interfaces to make ideas such as multi-objective optimization accessible. The idea of multi-objective optimization is not that hard to understand, but the currently available tools that implement it are insurmountably difficult to use for most spreadsheet users.

We have designed and implemented a prototype user-interface for combinatorial multi-objective optimization with a spreadsheet-like tabular design. This design is based on our study of the kinds of multi-objective optimization problems that a group of aerospace engineers works with most commonly. We believe that this user interface will also support many problems that users in other domains, such as business and other areas of engineering, may be interested in.

Acknowledgements

Many people have participated in helpful discussions of this work, and we are grateful for their time and insights; in alphabetical order: Felix Chang, Justin Colson, Ed Crawley, Greg Dennis, Arthur Guest, Wilfried Hofstetter, Andrew Yi Huang, Daniel Jackson, Eunsuk Kang, Ben Koo, Ben Kuhn, Maokai Lin, Gustavo Pinheiro, Rob Seater, Theo Seher, Bill Simmons, Dan Sturtevant, Tim Sutherland, Emina Torlak, Olivier de Weck, and Brian Williams. Ben Kuhn implemented part of an early version of this user interface.

This research was funded in part by the National Science Foundation under grant 0438897 (SoD Collaborative Research: Constraint-based Architecture Evaluation), and by the Air Force Research Laboratory (AFRL)/IF and Disruptive Technology Office (DTO) in the National Intelligence Community Information Assurance Research (NICIAR) Programme (ConfigAssure: Dynamic System Configuration Assurance for National Intelligence Community Cyber Infrastructure). A research scholarship was provided by the University of Paderborn.

About the Authors

Derek Rayside is a doctoral student in Daniel Jackson's Software Design Group at the MIT Computer Science and Artificial Intelligence Laboratory. Derek was previously a master's student with Kostas Kontogiannis at the University of Waterloo, during which time he did research in collaboration with CAS Toronto. Derek worked as an intern in the IBM Toronto Lab while studying for his undergraduate degree.

Christian Estler is a masters student at the University of Paderborn. He became involved with this project while visiting MIT. When he returned back to Germany he sold Derek his bicycle: a large Schwinn Madison in single-speed configuration with a steel frame and mountain-bike handlebars.

References

[1] X. Blasco, J.M. Herrero, J. Sanchis, and M. Martinez. Decision making graphical tool for multiobjective optimization problems. In *IWINAC'07*, volume 4527 of *Lecture Notes in Computer Science*, pages 568–577, 2007.

[2] Jonathan Edwards. No ifs, ands, or buts: Uncovering the simplicity of conditionals. In *Proceedings of the 22nd ACM/SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 639–658, Montréal, Canada, October 2007.

[3] Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22(4):425–460, 2000.

[4] Matthias Ehrgott and Xavier Gandibleux. Multiobjective combinatorial optimization: theory, methodology, and applications. In Matthias Ehrgott and Xavier Gandibleux, editors, *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Survey*, volume 52 of *International Series in Operations Research and Management Science*, pages 369–444. Kluwer Academic Publishers, Boston, MA, 2002. ISBN 1-4020-7128-0.

[5] Matthias Ehrgott and Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. In Christian Blum, Maria José Blesa Aguilera, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics: An Emerging Approach to Optimization*. Springer-Verlag, 2008. ISBN 978-3540782940.

[6] Matthias Ehrgott, José Figueira, and Xavier Gandibleux. Special issue on multiple objective discrete and combinatorial optimization. *Annals of Operations Research*, 147(1), October 2006.

[7] Petri Eskelinen, Kaisa Miettinen, Kathrin Klamroth, and Jussi Hakanen. Pareto navigator for interactive nonlinear multiobjective optimization. *OR Spectrum*, 2008. doi://10.1007/s00291-008-0151-6.

[8] José Figueira, Salvatore Greco, and Matthias Ehrgott, editors. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer-Verlag, 2005.

[9] Xavier Gandibleux and Matthias Ehrgott. 1984–2004 — 20 years of multiobjective metaheuristics. but what about the solution of combinatorial problems with multiple objectives? In Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, editors, *Proceedings of the 3rd Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science*, pages 33–46, Guanajuato, Mexico, March 2005. ISBN ISBN 3-540-24983-4.

[10] R. Halverson Jr. An Empirical Investigation Comparing IF-THEN Rules and Decision Tables for Rrogramming Rule-based Expert Systems. *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, 3, 1993.

[11] H.-Y. Benjamin Koo. *A Meta-language for Systems Architecting*. PhD thesis, Massachusetts Institute of Technology, 2005. Advised by Edward Crawley.

[12] Rolf Molich and Jakob Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33(3), 1990.

[13] Simon Peyton Jones, Alan Blackwell, and Margaret Burnett. A user-centred approach to functions in Excel. In Olin Shivers, editor, *Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 165–176, Uppsala, Sweden, August 2003. ACM Press, NYC, NY.

[14] U.W. Pooch. Translation of Decision Tables. *ACM Computing Surveys (CSUR)*, 6 (2):125–151, 1974.

[15] Derek Rayside, H.-Christian Estler, and Daniel Jackson. A Guided Improvement Algorithm for Exact, General Purpose,

Many-Objective Combinatorial Optimization. Technical Report MIT-CSAIL-TR-2009-033, MIT Computer Science and Artificial Intelligence Laboratory, 2009. URL `http://hdl.handle.net/1721.1/46322`.

[16] Christopher Scaffidi, Mary Shaw, and Brad Myers. Estimating the numbers of end users and end user programmers. In *VL/HCC'05*, 2005.

[17] Daniel Schrage, editor. *White Paper on Current State of the Art*. AIAA Technical Committee on Multidisciplinary Design Optimization (MDO), January 1991. URL `http://endo.sandia.gov/AIAA_MDOTC/sponsored/aiaa_paper.html`.

[18] Willard Simmons. *A Framework for Decision Support in Systems Architecting*. PhD thesis, Massachusetts Institute of Technology, 2008. Advised by Edward Crawley.

[19] D. Thomas. Agile Programming: Design to Accommodate Change. *IEEE Software*, 22(3):14–16, 2005.

[20] Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In Orna Grumberg and Michael Huth, editors, *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 632–647, Braga, Portugal, March 2007. Springer-Verlag.

[21] E. L. Ulungu and J. Teghem. Multi-objective combinatorial optimization problems: A survey. *Journal of Multi-Criteria Decision Analysis*, 3(2):83–104, 1993.

[22] I. Vessey and R. Weber. Structured Tools and Conditional Logic: an Empirical Investigation. *Communications of the ACM*, 29(1):48–57, 1986.