# Enriching Solutions to Combinatorial Problems via Solution Engineering

Thierry Petit, Andrew Trapp

# Enriching Solutions to Combinatorial Problems via Solution Engineering

Thierry Petit

*LS2N (TASC), IMT Atlantique (DAPI), CNRS, Nantes, France,*
*Worcester Polytechnic Institute, USA,*
*Thierry.Petit@imt-atlantique.fr*

Andrew C. Trapp

*Robert A. Foisie Business School, Worcester Polytechnic Institute, USA,*
*atrapp@wpi.edu*

**Abstract:** Existing approaches to identify multiple solutions to combinatorial problems in practice are at best limited in their ability to simultaneously incorporate both diversity among generated solutions, as well as problem-specific desires that may only be discovered or articulated by the user after further analysis of solver output. We propose a general framework for problems of a combinatorial nature that can generate a set of of multiple (near-)optimal, diverse solutions, that are further infused with desirable features. We call our approach *solution engineering*. A key novelty is that desirable solution properties need not be explicitly modeled in advance. We customize the framework to both the mathematical programming and constraint programming technologies, and subsequently demonstrate its practicality by implementing and then conducting computational experiments on existing test instances from the literature. Our computational results confirm the very real possibility of generating sets of solutions infused with features that might otherwise remain undiscovered.

**Keywords:** mathematical programming; constraint programming; diversity; quality notions; solution generation framework

## 1. Introduction

A number of analytical technologies exist to find solutions that satisfy the constraints of combinatorial problems. Arising from domains that include mathematical programming and constraint programming, among others, they can solve both satisfaction and optimization problems, for example various scheduling, assignment, routing, and configuration problems. Generally speaking, modern progress has advanced to solve reasonably complex combinatorial problems in an effective manner. Today's solvers can routinely return a solution, should one exist, to very large satisfaction and optimization problems.

A prerequisite to using such a solver is the ability to formally express fundamental and non-negotiable problem attributes via a *model*. A model implicitly characterizes the possible solutions through variables, together with constraints forbidding certain variable combinations. In the case of optimization, an objective function, commonly related to some measure of cost or time, guides the search.

In practice, the requirement of a model raises important issues. Model parameters are often based on most-likely estimates of uncertain and fluctuating data. Imperfect communications between the decision maker – whom we term the *user* – and the modeler can result in inaccuracies. Some aspects can be altogether omitted due to the difficulty in expression (Schittekat and Sörensen 2009); others may be temporarily disregarded so that the model becomes tractable (Carvajal et al. 2013). Even under the best of circumstances, models are at best abstractions of reality, not without limitations. As expressed by the renowned George Box, "The most that can be expected from any model is that it can supply a useful approximation to reality: all models are wrong; some models are useful." (Box et al. 2005).

Another challenge is that in many cases, there are subtle and nuanced model preferences that are very desirable to the user, but for which they are a priori either unable to articulate, or simply unaware. This is the case in rostering problems, with concepts such as *fairness* and *equity*. Recognizing these inherent limitations of models, and in light of advancing technology, researchers have devised clever ways to alleviate such situations.

Decision support systems and related technologies have been designed to facilitate solution exploration through "what if?" user interaction, possibly including visualization techniques (see, e.g., Shim et al. 2002). They have similarly been used to visualize and understand the frontier of Pareto optimal solutions in multiobjective optimization contexts (see, e.g., Efremov et al. 2009). Model acquisition frameworks try to learn models from user responses to questions (see, e.g., Beldiceanu and Simonis 2011, Bessiere et al. 2016) or via natural language problem descriptions (Kiziltan et al. 2016). Another approach is the multi-cycle model design, where the solver can be enriched to provide explanations about the internal decisions made when generating the solution (see, e.g., Ouis et al. 2003). These explanations may help to iteratively refine the model until a satisfactory solution is found.

A number of other studies (Petit and Trapp 2015, Trapp and Konrad 2015, Camm 2014, Eiter et al. 2013, Hebrard et al. 2011, Danna and Woodruff 2009, Schittekat and Sörensen 2009, Hentenryck et al. 2009a, Hebrard et al. 2005, Glover et al. 2000, 1998, Kuo et al. 1993) have demonstrated interest in exploration *beyond the single solution returned by the solver*. These studies discuss generating multiple solutions to combinatorial problems, and some further integrate diversity into their discussions. The overarching goal is to provide the user with greater flexibility for selecting the solution that will ultimately be implemented.

Largely absent from the aforementioned studies is a means to explore feasible solutions that, in addition to a possible objective function, incorporate various notions of quality that form subsets of solutions having common properties. While diversity is commonly measured via mathematical distance measures on solution vector values, two solutions that are distinct according to separate quality notions offer clear alternatives, even when these solutions have many variable values in common. While it is no longer prohibitive to consider multiple notions of quality on modern computational infrastructure, care must still be exercised – explicitly incorporating quality notions into the model can have an adverse side effect of overly emphasizing such quality notions. There is a delicate balance between optimizing or

constraining for a quality notion, and this is accentuated when considering multiple notions. This underscores the need for an approach that doesn't require up-front, application specific, declaration of notions of quality, *but rather to use them to explore the solution space.*

In this paper we propose to solve combinatorial problems via the *solution engineering* framework: augmenting a combinatorial model with one or more general quality notions *as a means* to generate multiple diverse (near-)optimal solutions. We do this for the purpose of infusing the solution set with ideally desirable, yet hard to articulate, features that can be further analyzed using data science technologies. We develop a general framework that can handle a wide range of quality notions, including those appearing in recent applications in the literature such as *fairness* (Bertsimas et al. 2012), *persistence* (Brown et al. 1997, Bertsimas et al. 2006, Morrison 2010), and *balanced* solutions (Schaus et al. 2009), so long as the notion can be expressed using the respective technology. In particular, we focus on notions that can be represented through inclusion of one or more variables and/or constraints, quantifying the notion via a surrogate variable that measures the performance of the notion. At the end of the solution generation process, each solution is expanded with a vector of quality scores – one score per quality notion.

A key novelty of our framework is to circumvent issues raised by requiring all of the desirable solution properties to be stated in the model a priori. Simultaneously, we provide a broad spectrum of alternatives to the user. Input data include a combinatorial model and, separately, a set of quality notions. Our framework aims at identifying solution sets that perform well with respect to each selected quality notion. From these sets, the "best" solution, from the point of view of the user, may be identified in a second phase using data science technologies such as SQL, filtering, and visualizations. Queries may be stated both on solutions and their quality scores. A first advantage is that such queries can be very intricate and precise, much more than a usual weighted objective for example, without any impact on the solving process since the queries are not part of the model. A second beneficial property is that queries are set only once the solutions have been generated, rather than blindly complicating a model without having an idea of the structure of the solutions; the queries operate *on the outcome of the model*. This provides the user insight into the optimal structure or substructure of the variables, prior to proceeding. We estimate that this is particularly valuable when queries express preferences about what the user does not want in a solution. A third benefit is that one may consider generating myriad solutions, as queries may filter many solutions before looking deeply into the most relevant ones.

Our contributions are fivefold. First, we present an overarching solution engineering framework to generate a diverse set of solutions for combinatorial problems by augmenting combinatorial models with general quality notions as means to obtain desirable solution features. Second, we summarize the state-of-the-art with respect to the generation of multiple solutions to combinatorial optimization problems, including the incorporation of diversity and quality. Third, we customize the general framework to two special cases, mathematical programming and constraint programming, which includes theory and decompositions unique to each of these technologies. Fourth, we implement the framework in these two technologies, demonstrating its practicality. Fifth, in the process of conducting computational experiments, we provide novel decompositions for the selected quality notions.

The remainder of this paper is organized as follows. We motivate our study in Section 2 via an introductory example. Section 3 covers relevant literature related to our study, while

Section 4 introduces both the general framework, as well as customizations to both mathematical programming and constraint programming. In Section 5 we present the results of our computational experiments in both technologies on a classical combinatorial optimization problem. The study concludes in Section 6 with a review of our work and a discussion of possible future directions. Appendices follow that contain theoretical results.

# 2. Motivating Example To Underscore Notions of Quality

We motivate our framework with an illustrative example. Consider a scenario where tasks are assigned to three machines A, B and C. Figure 1 shows two solutions to this problem. They are not overly diverse if one considers mathematical distances, such as the Hamming distance: 7 of the 9 tasks are assigned to the same machine in both solutions. From a human point of view, however, there is a difference between solution 1 and solution 2. Assignments to machines B and C are alternated in solution 1, while in solution 2 the two machines are used without interruption. In practice, this may have significant influence on the choice of solution. On the other hand, a common property of the two solutions is the fair task distribution on the machines: each machine has exactly three tasks. Obviously, one may consider a solution near to Solution 2 but where machine C is used only twice and B is used four times, just by changing one value. Again, although very similar, such solutions may not be considered as equivalent by the user. Despite its simplicity, this example highlights that there exist meaningful ways to differentiate solutions beyond simple mathematical norm-based measures used in most existing approaches (see, e.g., Hebrard et al. 2005, 2007, Trapp and Konrad 2015).

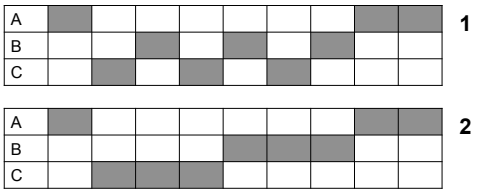


Figure 1: Two solutions of an assignment problem, A, B and C are machines and gray cells are tasks.

As a second example, consider the minimization version of the Generalized Assignment Problem (GAP), shown to be $\mathcal{NP}$-hard in Fisher et al. (1986). Given $n$ jobs $J = \{1, 2, ..., n\}$ and $m$ agents $A = \{1, 2, ..., m\}$, the goal is to determine a minimum cost scheme that assigns each job to exactly one agent while satisfying a resource constraint for each agent. Assigning job $j$ to agent $i$ has a cost of $c_{ij}$ and consumes an amount $a_{ij}$ of resource, whereas the total amount of the resource available for agent $i$ is $cap_i$.

A solution is a surjective mapping $J \to A$. Let $A(j)$ denote the agent assigned to job

$j \in J$. The problem is then formulated as follows:

$$\begin{aligned} &\text{minimize} \quad \sum_{j \in J} c_{A(j)j} \\ &\text{subject to} \quad \forall i \in A, \sum_{j \in J : A(j) = i} a_{ij} \leq cap_i. \end{aligned} \tag{1}$$

For illustrative purposes, we assume a practical context where the problem has an additional temporal component that is not explicitly stated in the model: jobs will start in sequence according to their index in $J$, so that no two jobs start at the same time. We may consider many quality notions that would likely discriminate solutions, independently from the mathematical distances that are classically used to state that two solutions are diverse. Let us consider three examples.

1. The number of jobs assigned to agents may vary more or less. We call this number the *cardinality* of an agent. As a quality notion, we consider the maximum distance between any two cardinalities of agents in $A$.

2. The sum of resources consumed by each agent also varies. We consider, over all pairs of agents, the maximum distance between total resources used.

3. As jobs are assumed to be executed in sequence, solutions can be assessed according to the variation of costs incurred by any two consecutive jobs in $J$. We count the number of differences of more than $gap = k$ units of cost among all pairs of consecutive jobs. A solution is denoted *smooth* if this number is "low", that is, having relatively smooth cost transitions between consecutive jobs.

Figure 2 shows solutions to three GAP test instances (GAPa_5_100) selected from the publicly available OR-Library (Chu and Beasley 1997). We emphasize that the scale of each quality notion, such as the difference in resources consumed, is not precisely known in advance. This supports our argument against requiring all solution criteria and features to be expressed in the original model. Conversely, once the solutions sets are generated, the scales become known, and one may state value-based queries to refine the choice of solution. It is worth noting that if there are only a few solutions to the problem that satisfy the model constraints, our framework suffers a fate similar to that of other state-of-the-art diversity approaches.

Moreover, solutions can substantially differ according to their quality notion values, and consequently, these notions can be used to classify solutions. Established distance measures such as Hamming and Manhattan distances may entirely overlook such differences. Solutions 1 and 2 are respectively optimal and very near-optimal, however they vary according to cardinality and resource distances. Solution 3 is comparable to Solution 1, but with a slightly superior objective value and an optimized smooth quality score when considering a gap of 15.

We propose a framework that permits the generation of many solutions, which, at the conclusion of the process, may be infused with any variety of quality notions. To accomplish this, we intend to keep the solution process as stable as possible, by allowing the user to

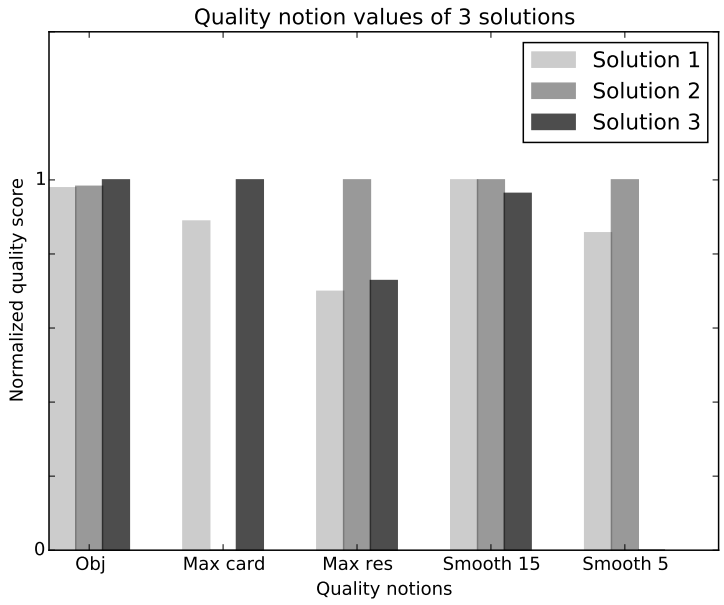


Figure 2: Three solutions of a GAP minimization problem: the scores of solutions are depicted according to the original objective value and four distinct quality notions. Note that solution 2 has a zero value for "Max card", and solution 3 has a zero value for "Smooth 5".

pass an existing combinatorial model. The complexities of solution generation are handled internally by the framework, transparent to the user.

Among the many applications for which our framework would be useful, we emphasize a few that seem particularly promising. Various assignment and scheduling problems involving human resources are often subject to subtle conditions that depend on criteria external to the problem. Among others, one may cite the examples of managing resistance to change (Armenakis and Bedeian 1999) and the negotiation of acceptable compromises. In this latter case, it is likely exceedingly difficult to state the criteria in advance. For instance, the user may discover the degree to which resource consumption may be smoothed while the objective or other solution features remain acceptable.

In computer-aided musical composition, pieces are structured into sections, phrases, and patterns. Optimization models include those involving Markovian constraints able to generate lyrics in the style of a popular author (Barbieri et al. 2012), as well as efforts to generate melodic chord sequences using a series of constraints (Schell 2002). However, anticipating all the features that would likely be pleasing from a human perspective is unrealistic. In this context, clustering sets of solutions using notions that go beyond the concepts of optimality and standard diversity can be very valuable. In still other contexts, adding some hard constraints to a model may lead to unsolvable, or even intractable, problems (Carvajal et al. 2013); instead, one may prefer to leave out such constraints, and evaluate the resulting solutions on important notions of quality. We are unaware of any other framework to obtain a variety of solutions that are likely to be desirable to the user.

# 3. Background and State-of-the-Art

In combinatorial problems, solutions are defined as vectors of variable values that satisfy problem constraints. For satisfaction problems, this is the single distinguishing criterion – whether or not a vector of variable values satisfies all constraints. For optimization problems, a key additional distinguishing criterion is the evaluation of the solution via the objective function. The objective function essentially separates optimal from suboptimal solutions, and thereby proving that no better solution exists. Hence in standard optimization problems the objective function is the de facto arbiter of quality.

## 3.1 Multiple Solutions to Combinatorial Problems

A wide variety of research using diverse technologies such as mathematical programming, constraint programming, SAT, (meta)heuristics, and others have investigated the concept of finding *multiple* solutions to combinatorial problems, the majority of which being implemented using the first two technologies. The processes used to obtain multiple solutions can be described as either offline (i.e., simultaneous) or online (i.e., sequential).

A few studies outline ways to find multiple solutions, independent of the concept of diversity. In Danna et al. (2007) the authors develop several online, tree-based algorithms to find multiple optimal and near-optimal solutions. Similarly, in Tsai et al. (2008) the authors propose an online approach to identify all multiple optima for pure integer programs. A somewhat simpler sequential approach, inspired by integer cuts (Balas and Jeroslow 1972), is outlined in Camm (2014) for the case of 0–1 integer linear optimization problems. In Voll et al. (2015) the authors also employ cuts on integer variables on mixed-integer nonlinear programs to obtain multiple optima and near-optima. Intriguingly, they suggest further investigation is warranted on the structural properties of these optima and near-optima, which in many ways is related to the present work. Some approaches put the focus on compiled representations of the solution space, e.g., through automata in Amilhastre et al. (2002) or multi-valued decision diagrams (MDDs) in Hadzic et al. (2009). Such representations can be exploited in real-time using knowledge-compilation techniques.

## 3.2 Multiple, *Diverse* Solutions

Additional works incorporate the task of *diversification* among these solutions, as it is often desirable to the user that solutions are structurally distinct from one another. Most frameworks in the literature consider diversity between solutions through mathematical distance measures, where two solutions are diverse if the pairwise distance between variable values is "high". In particular, two studies consider satisfaction problems with the aim of incorporating diversity into the solution set. In Hebrard et al. (2005) and Hebrard et al. (2007), the authors use online constraint programming techniques to generate solution sets that maximize both diversity and similarity metrics. A similar method is used in Hentenryck et al. (2009b), in the context of automatic generation of architectural tests. In Nadel (2011), a similar task is undertaken of sequentially finding a set of diverse solutions with a SAT solver.

For the task of finding a diverse set of solutions to combinatorial *optimization* problems, Kuo et al. (1993) maximize a diversity metric over a given set of elements to simultaneously obtain the subset that is maximally diverse. In Glover et al. (1998), a similar problem is considered, however the authors instead develop heuristic approaches to find diverse sets of solutions, which are empirically demonstrated to be of high quality. It should be noted that (pairwise) diversity in this context is measured a priori between set elements, whereas diversity in other contexts has to do with actual variable values in the solution vector. In the context of recommender systems, Adomavicius and Kwon (2014) develop techniques to find sets of solutions that maintain both diversity and accuracy.

The authors in Greistorfer et al. (2008) consider the question of whether online or offline approaches are more effective in identifying two solutions to binary integer linear programs that are mutually diverse. In Glover et al. (2000) the authors consider online, heuristic approaches to identify diverse and "good" solutions to mixed-integer programming problems. The same problem is tackled in Danna and Woodruff (2009), who provide both offline exact, and online heuristic approaches. In Trapp and Konrad (2015) the authors devise an online approach to iteratively identify a set of solutions to binary integer linear programs that maximize the ratio of diversity to loss in objective value. A similar ratio is also optimized in Petit and Trapp (2015), but in the context of constraint programming, to generate a set of diverse solutions that perform well with respect to the objective function. A variant of tabu search is considered in Schittekat and Sörensen (2009) to incorporate diversity into a set of competitive solutions.

Moreover, several works use techniques outside of mathematical programming and constraint programming to accomplish similar goals. In Eiter et al. (2013) the authors use answer set programming in an iterative fashion to find diverse (and similar) optima. The authors of Løkketangen and Woodruff (2005) discuss the development of diversity metrics that incorporate aspects of the psychology of decision making. They employ dynamic programming and multi-objective optimization in an online manner to locate diverse solutions on the efficient frontier of the solution space. Finally, the authors of Hadzic et al. (2009) exploit the MDD-based solution space representation to obtain a set of diverse optima.

## 3.3 Summary of Related Approaches

Table 1 synthesizes a significant collection of state-of-the-art approaches that consider the generation of multiple solutions. It indicates for each paper the technology used, type of generation (online or offline), whether it additionally considers generating diverse solutions, as well as the nature of the solution space – that is, satisfaction versus optimization; and if optimization, whether only *optimal* solutions are considered, or also *near-optima*.

## 3.4 Incorporating Quality into Solutions

Moving beyond solely the objective function, some works have considered additional quality notions in the literature. Notable examples in the operations research literature include fairness (Bertsimas et al. 2012) and persistence (Brown et al. 1997, Bertsimas et al. 2006, Morrison 2010). In constraint programming and constraint-based local search, global constraints have been designed to encode such quality notions; indeed, global constraints (Beldiceanu

| Author (Year) | Technology | Solution generation | Diversity | Solutions |
|---|---|---|---|---|
| Camm (2014) | MP | online | no | (near) optimal |
| Danna et al. (2007) | MP/HEUR | online | yes | (near) optimal |
| Danna and Woodruff (2009) | MP/HEUR | offline | yes | (near) optimal |
| Adomavicius and Kwon (2014) | MP/HEUR | online/offline | yes | (near) optimal |
| Eiter et al. (2013) | AP | online | yes | satisfaction problems |
| Glover et al. (1998) | HEUR | offline | yes | (near) optimal |
| Glover et al. (2000) | HEUR | online | yes | (near) optimal |
| Greistorfer et al. (2008) | MP | online/offline | yes | (near) optimal |
| Hadzic et al. (2009) | DD | offline | yes | satisfaction problems, (near) optimal |
| Hebrard et al. (2005) | CP | online | yes | satisfaction problems |
| Hebrard et al. (2007) | CP | online/offline | yes | satisfaction problems |
| Hentenryck et al. (2009b) | CP/HEUR | online | yes | satisfaction problems, (near) optimal |
| Kuo et al. (1993) | MP | offline | yes | (near) optimal |
| Løkketangen and Woodruff (2005) | DP | online | yes | (near) optimal |
| Nadel (2011) | SAT | online | yes | satisfaction problems |
| Petit and Trapp (2015) | CP | online | yes | (near) optimal |
| Schittekat and Sörensen (2009) | HEUR | online | yes | (near) optimal |
| Trapp and Konrad (2015) | MP | online | yes | (near) optimal |
| Tsai et al. (2008) | MP | online | no | only optimal |
| Voll et al. (2015) | MP/SA | online | no | (near) optimal |

Table 1: State of the art approaches related to multiple solution generation. AP: Answer Set Programming; CP: Constraint Programming; DD: Decision Diagrams; DP: Dynamic Programming; HEUR: Heuristics; MP: Mathematical Programming; SA: Specialized Algorithm; SAT: Constraint Satisfiability.

et al. 2007) are modeling blocks for which dedicated solution techniques are devised. For instance, the Deviation constraint introduced by Schaus et al. (2007a) is used to favor load balancing in various problems, such as nurse to patient assignment problems (Schaus et al. (2009)) and cloud-based data grid optimization (Sebbah et al. (2016)). To the best of our knowledge, all existing works consider quality notions as part of the problem statement, either in the form of hard constraints or via objective function measures.

## 3.5 Relation to Multiobjective Optimization

The literature is vast with respect to identifying good solutions to combinatorial problems by optimizing for explicit objectives, including multiobjective optimization approaches. Common methods to handle a set of explicit, competing objectives include scalar approaches and Pareto optimization. The latter is concerned with methods to identify nondominated, *Pareto efficient* solutions with respect to the various objectives – that is, those solutions for which there is no other that performs at least as well in all objectives, and strictly better according to at least one objective. While desirable, enumerating all nondominated solutions to combinatorial problems is known to be computationally demanding (Karasakal and Köksalan 2009). The former is perhaps a more straightforward approach that combines each individual objective into an aggregate function through additive weighting. While attractive in that it provides a simple mechanism to merge objective functions, a drawback is that some Pareto efficient solutions may remain undiscovered. For a more detailed treatment of these topics, we refer the interested reader to Ehrgott and Gandibleux (2000).

In this paper, we choose to combine quality notions, diversity measures, and (possibly) the original objective function into a single ratio objective. While we could use multiobjective

optimization techniques, we here restate for the sake of distinction a key difference: *the model is not initially formulated with explicit representation of all of the desired features*; indeed, such features may not even be known with clarity by the user.

# 4. Technical Developments

This section presents the core of the solution engineering framework and its specialization to mathematical programming and constraint programming.

## 4.1 Mathematical Preliminaries

Let $X$ be a set of $n$ integer variables. We consider the following problem $p$:

$$\min_{X \in \mathbb{Z}^n} \{f(X) \mid X \in \mathcal{C}\}. \tag{2}$$

When $f$ is stated to be a constant function such as $f(X) = 0$, then $p$ is considered a problem of *feasibility*, or *satisfaction*. A combinatorial model of problem $p$ can be expressed via variable set $X$, constraint set $\mathcal{C}$, and objective function $f$, or more compactly via the triplet $(X, \mathcal{C}, f)$. Each constraint $C \in \mathcal{C}$ implicitly states the permissible assignments for a given subset $Y$ of variables in $X$, its scope. An assignment $A_X$ of a superset $X$ of the variables $Y$ *satisfies* $C$ if and only if the projection on the variable scope of $C$ does not violate the constraint. In addition, we assume the value of each variable $x \in X$ belongs to a finite domain $D(x)$. This domain can be viewed as a supplementary unary constraint.

A *solution* $S$ to $p$ is a pair $S = (A_X, z)$ where $A_X$ is an assignment of $X$, $z = f(A_X)$ and $A_X$ satisfies all the constraints in $\mathcal{C}$. A solution is optimal if and only if no other feasible solution exists to $p$ with a strictly lower value for $z$.

## 4.2 A New Diversity Scheme

Given $k$ solutions of a problem on $n$ variables, diversity is calculated using some function

$$\Delta : (\mathbb{Q}^n)^k \mapsto \mathbb{Q}_+. \tag{3}$$

Some functions used in the literature include the distance to the centroid of all other solutions (Trapp and Konrad 2015), as well as a sum of mathematical distances computed with solution pairs (see, e.g., Hebrard et al. 2005, 2007).

To select $k$ maximally diverse solutions from the whole set of feasible solutions to $p$, one needs to solve the following problem $p_{div}$ on the model $(X^k, \mathcal{C}^k, f)$, obtained by making $k$ copies of the variables and constraints:

$$\begin{aligned} &\text{maximize} \quad d = \Delta(X^k) \\ &\text{subject to} \quad \mathcal{C}^k. \end{aligned} \tag{4}$$

It is possible to consider the case of diverse solutions that perform well with respect to the objective value. The problem $p_{div}$ has then two criteria:

$$\begin{array}{ll} \text{maximize} & d = \Delta(X^k) \\ \text{minimize} & z = \oplus_{i=1}^{k}(f^{loss}(X_i)) \\ \text{subject to} & \mathcal{C}^k. \end{array} \tag{5}$$

In (5), $\oplus$ is any aggregation function and $f^{loss}$ evaluates the solution with respect to the objective of problem $p$, e.g., the distance to the best objective value found so far. Due to variable set duplication, even for very small problems solving exactly $p_{div}$ is prohibitive. To remedy this issue, both offline and online approaches can be considered. In the offline case, starting from a set of solutions $\mathcal{S}_p$ generated by an oracle, the problem consists of extracting a subset that maximizes diversity. In the online case, problem $p$ is transformed into a new problem that integrates a diversity component into the objective. Solutions to $p$ are then computed incrementally (one by one), with each new solution explicitly considering the diversity from previously generated solutions. In this paper we consider an online diverse solution scheme based on the concept of *quality notion*.

**Definition 1** (Quality notion)**.** *Given the model $(X, \mathcal{C}, f)$ of a problem $p$, a quality notion $Q(Y, \mathcal{V})$ is one or more constraints defined on the variables $Y \cup \mathcal{V}$, such that the three following conditions are true: (1) $Y \subseteq X$; (2) $\mathcal{V} \cap X = \emptyset$; (3) $\mathcal{V} \supseteq \{q\}$, where $q$ is a variable.*

Variable $q$ is the *score* of the quality notion. Without loss of generality, we consider the lower the value of $q$, the better the quality of the assignment of values is to $Y$. This is apparent in the context of goal programming (Tamiz et al. 1998), where quality is interpreted as deviations from targeted levels of some quantitative notion(s) of quality, the (weighted) sum of which is to be minimized. In constraint programming, many global constraints involve a particular variable that characterizes a property on the other constrained variables, such as Among (Bessière et al. 2005), Balance (Bessiere et al. 2014), Deviation (Schaus et al. 2007b), Focus (Narodytska et al. 2013), NValue (Cambazard and Fages 2015), NVector (Chabert et al. 2009, Petit and Petit 2016), Spread (Pesant and Régin 2005), and others. Using other technologies, properly characterizing desired quality notions may require specific modeling constructs such as additional variables and constraints.

**Definition 2** (Quality enhanced solution)**.** *Let $\mathcal{Q}$ be a set of $m$ quality notions, $\mathcal{Z}_\mathcal{Q} = \{q_1, q_2, \ldots, q_\ell, \ldots, q_m\}$ be the* quality variable *representing each $Q_\ell \in \mathcal{Q}$, $S = (A_X, z)$ a solution to $p$ with original objective value $z$, and $A_{\mathcal{Z}_\mathcal{Q}}$ an assignment of values to variables in $\mathcal{Z}_\mathcal{Q}$. Then a* quality enhanced solution *to $p$ is a pair $S^\mathcal{Q} = (S, A_{\mathcal{Z}_\mathcal{Q}})$, such that $A_X \cup A_{\mathcal{Z}_\mathcal{Q}}$ satisfies all the constraints in $\mathcal{Q}$.*

Our approach aims at generating a diverse solution set of optima or near-optima, where each solution is augmented by a *vector of quality notion scores*, or quality *score vector*. The score vector captures the solution performance across all quality notions (and possibly, the original objective function). We first maintain to separately generate, for each quality notion, a set of solutions that are diverse and emphasize this specific quality notion, though nothing prohibits any pair of quality notions $(Q_i, Q_j)$ to be aggregated to form a new quality notion

$Q_k$, e.g., by stating $q_k = \alpha \times q_i + \beta \times q_j$, where $\alpha$ and $\beta$ are constants. While our framework allows the generation of solutions sets that simultaneously consider multiple quality notions, we opt to isolate a specific quality notion for each set of solutions.

Algorithm 1 generates a set $\mathcal{S}_\ell$ of $k$ diverse solutions according to quality notion $Q_\ell \in \mathcal{Q}$. It requires the following parameter list $\mathcal{L}$:

- A model $M = (X, \mathcal{C}, f)$ for $p$.
- A positive integer $k$, the number of solutions to be generated, providing that $p$ admits at least $k$ feasible solutions.
- A first solution $S_0$ to problem $p$; and in the case that $p$ is an optimization problem, preferably optimal or near-optimal.
- A function $f^{loss}$ for evaluating the optimality of a solution with respect to $f(S_0)$. For instance, $f^{loss}$ can be defined as:

$$f^{loss}(X) = \max(0, f(X) - f(S_0)). \tag{6}$$

- A quality notion $Q_\ell(Y_\ell, \mathcal{V}_\ell)$, such that $\{q_\ell\} \subseteq \mathcal{V}_\ell$.
- A set of functions $\{\Delta_\ell^1, \Delta_\ell^2, \ldots, \Delta_\ell^k\}$ such that $\forall h \in \{1, \ldots, k\}, \Delta_\ell^h : (\mathbb{Q}^n)^{h+1} \mapsto \mathbb{Q}_+$.
- A new objective function $r$ defined from $\Delta, f^{loss}$ and $q_\ell$, that attempts to balance three measures: solution diversity, original objective function of $p$, and quality notion $Q_\ell$. For each new solution to be generated, let $\mathcal{S}_\ell$ be the set of previously generated solutions by the Algorithm (initially $\mathcal{S}_\ell = \emptyset$). This could be accomplished, for example, by stating $r$ as a ratio, and using three constants $a, b$ and $c$ for normalizing the three quantities:

$$r = \frac{a \times \Delta_\ell^h(\{X\} \cup \mathcal{S}_\ell \cup \{S_0\})}{b \times q_\ell + c \times f^{loss}(X)}. \tag{7}$$

Given a set of quality notions $\mathcal{Q}$, Algorithm 2 uses Algorithm 1 to generate a set $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{|\mathcal{Q}|}\}$ of solution sets mapped in a one-to-one fashion with a data matrix of score vectors $\Psi = \{\Psi_1, \Psi_2, \ldots, \Psi_{|\mathcal{Q}|}\}$. Each $\Psi_\ell \in \Psi$ is a matrix of $k \times |\mathcal{Q}|$ values, that provides quality scores for each solution $S_j \in \mathcal{S}_\ell$. Algorithm 2 considers that $\forall\, \ell \in \{1, 2, \ldots, |\mathcal{Q}|\},\ k = |\mathcal{S}_\ell|$, but it may easily be generalized to the case where solutions sets have different sizes depending on quality notions.

Algorithm 2 requires solving $|\mathcal{Q}|$ (potentially) $\mathcal{NP}$-hard problems in Algorithm 1. Hence, one may envision stating a fixed time limit for Algorithm 1, returning the best found solution thus far. Moreover, Algorithm 2 is impartial to the technology used for solving the combinatorial problem derived from $p$. As such, we choose in the next two sections to demonstrate the implementation of Algorithm 1 in both mathematical programming and constraint programming.

**input :** List of parameters $\mathcal{L}$.
**output:** A set of $k$ diverse solutions generated using quality notion $Q_\ell(Y_\ell, q_\ell)$.

$\mathcal{S}_\ell := \emptyset$;
$h := 1$;
$M' := M$;
**while** $h \leq k$ **do**
  $X := X \cup \mathcal{V}_\ell$;
  $\mathcal{C} := \mathcal{C} \cup \mathcal{C}_\ell \cup \{$Constraint(s) to forbid repeating any solution in $\mathcal{S}\}$;

$$\begin{aligned} &\text{maximize} && r \\ &\text{subject to} && \mathcal{C}. \end{aligned}$$

  **if** *No solution within (optional) time limit,* **then return** $\mathcal{S}_\ell$ ;
  Save the data $(r, \Delta_\ell^h(\{X\} \cup \mathcal{S}_\ell \cup \{S_0\}), f(X))$;
  $h := h + 1$;
  $\mathcal{S}_\ell := \mathcal{S}_\ell \cup \{$values assigned to variables $X$ in the solution$\}$;
  $M := M'$;
**return** $\mathcal{S}_\ell$;

**Algorithm 1:** Online generation of diverse solutions enhanced with quality notion $Q_l$.

**input :** A model $M = (X, \mathcal{C}, f)$ for problem $p$, a set of quality notions $\mathcal{Q}$, and an integer $k > 0$.
**output:** A set $\mathcal{S}$ of solution sets and a $|\mathcal{Q}| \times (k \times |\mathcal{Q}|)$ matrix of score vectors, i.e., one set of $k$ $|\mathcal{Q}|$-length score vectors per quality notion.

$S_0 :=$ first solution to $p$;
$\mathcal{S} := \emptyset$;
**optional:** constrain using constant $K \geq 0$ the objective function in $p$:
$\mathcal{C} = \mathcal{C} \cup \{obj \leq f(S_0) + K\}$;
**for** $\ell \in \{1, 2, \ldots, |\mathcal{Q}|\}$ **do**
  Generate the parameter list $\mathcal{L}$ according to $Q_\ell$;
  $\mathcal{S}_\ell =$ **Algorithm 1**$(\mathcal{L})$;
  $\mathcal{S} := \mathcal{S} \cup \{\mathcal{S}_\ell\}$;
$\Psi = \emptyset$;
**for** $\ell \in \{1, 2, \ldots, |\mathcal{Q}|\}$ **do**
  $\Psi_\ell := |\mathcal{S}_\ell| \times |\mathcal{Q}|$ integer matrix;
  **for** $S_j \in \mathcal{S}_\ell$ **do**
    **for** $m \in \{1, 2, \ldots, |\mathcal{Q}|\}$ **do**
      $\Psi_\ell[j][m] :=$ Score of $q_m$ when evaluating $Q_m$ on solution $S_j$;
  $\Psi := \Psi \cup \{\Psi_\ell\}$;
Return $(\mathcal{S}, \Psi)$;

**Algorithm 2:** Online generation of diverse sets of solutions enhanced with quality notions $Q_1, \ldots, |\mathcal{Q}|$.

## 4.3 Solution Engineering Framework Implemented in Mathematical Programming

Algorithm 3 customizes Algorithm 1 to the mathematical programming paradigm, and in particular, we consider combinatorial optimization problems modeled using 0–1 variables. Specifically, for quality notion $Q_\ell(Y_\ell, q_\ell)$, it takes as input a list of parameters $\mathcal{L}$ as outlined in Section 4.2, including (possibly empty) sets of variables $\mathcal{V}_\ell$ and constraints $\mathcal{C}_\ell$. As output, it generates a set $\mathcal{S}_\ell$ of $k$ diverse solutions with respect to $Q_\ell(Y_\ell, q_\ell)$.

To construct such a set $\mathcal{S}_\ell$ of diverse solutions, it is necessary to enforce that a unique solution, if one exists, is returned upon each call, so that previously discovered solutions are not revisited on subsequent iterations. This can be accomplished by prohibiting a known solution $S_j$ through the following inequality (Balas and Jeroslow 1972):

$$\sum_{i:S_j[i]=0} x_i + \sum_{i:S_j[i]=1} (1 - x_i) \geq 1. \tag{8}$$

Concerning diversity, the metric $\Delta_\ell^h$ considers $k$ *reference* solutions $S_1, \ldots, S_k$, and measures the distance from their centroid $\mathbf{c} = \left( \mathbf{c}_i = \frac{1}{k} \sum_{j=1}^{k} S_j[i],\ i = 1, \ldots, n \right)$ in the following fashion:

$$\Delta_\ell^h = \sum_{i=1}^{n} \{(1 - \mathbf{c}_i)x_i + \mathbf{c}_i(1 - x_i)\} = \sum_{i=1}^{n} \mathbf{c}_i + \sum_{i=1}^{n} (1 - 2\mathbf{c}_i)x_i. \tag{9}$$

Moreover, at every algorithmic iteration $i$, a feasible solution $X$ is generated that optimizes $r$, which is defined as the ratio:

$$r = \frac{a \times \Delta_\ell^h(\{X\} \cup \mathcal{S}_\ell \cup \{S_0\})}{b \times q_\ell + c \times f^{loss}(X)}. \tag{10}$$

Algorithm 3 adds to model $M$ any necessary auxiliary variables $\mathcal{V}_\ell$ and constraints $\mathcal{C}_\ell$ to properly express quality notion $Q_\ell$, along with constraints of the form (8) that forbid previously generated solutions. The objective function is a nonlinear ratio to be maximized. Specifically, the numerator expresses the weighted diversity, while the denominator is a sum of the weighted quality notion and the weighted loss from the best objective value found so far. The signs are chosen on the three expressions so that diversity, quality, and the original objective function all improve whenever the ratio is maximized.

The nonlinearity of the ratio objective function $r$ can be addressed via Dinkelbach's Algorithm (Dinkelbach 1967), which is capable of solving optimization problems with ratio objectives. It does so by solving a sequence of linearized problems that are related to the original nonlinear fractional programming problem. The following algorithm details our implementation, which is called when performing Line 7 in Algorithm 3. At iteration $k$, $\lambda^k$ represents an estimate of the ratio $r$. The feasibility set induced by constraints $\mathcal{C}$ is denoted feas($\mathcal{C}$). Moreover, $\epsilon$ is a small positive constant.

The algorithm operates by computing the value for parameter $\lambda^k$ with respect to solution $X^k$. It subsequently uses that value to solve the parametrized optimization problem, returning $X^k$. When the parameter $\lambda^k$ reaches a value such that the resulting $X^k$ satisfies

**input :** List of parameters $\mathcal{L}$.
**output:** A set of $k$ diverse solutions generated using quality notion $Q_\ell(Y_\ell, q_\ell)$.

1 $\mathcal{S}_\ell := \emptyset$;
2 $h := 1$;
3 $M' := M$;
4 **while** $h \leq k$ **do**
5 $\quad X := X \cup \mathcal{V}_\ell$;
6 $\quad \mathcal{C} := \mathcal{C} \cup \mathcal{C}_\ell \cup \{\text{Constraints (8) } \forall\ S_j \in \mathcal{S}_\ell\}$;
7 $\quad$ maximize $r$ as expressed in (10), subject to $\mathcal{C}$;
8 $\quad$ **if** *No solution* **then return** $\mathcal{S}_\ell$ ;
9 $\quad$ **else** Save data $(r, \Delta_\ell^h(\{X\} \cup \mathcal{S}_\ell \cup \{S_0\}), q_\ell, f(X))$ ;
10 $\quad h := h + 1$;
11 $\quad \mathcal{S}_\ell := \mathcal{S}_\ell \cup \{\text{values assigned to variables of } X \text{ in the solution}\}$;
12 $\quad M := M'$;
13 **return** $\mathcal{S}_\ell$;

**Algorithm 3:** Online generation of diverse solutions enhanced with quality notion $Q_l$, adapted to mathematical programming.

**input :** Ratio $r$ as expressed in (10), initial solution $X^0$ (e.g., $S_0$).
**output:** Optimal solution $x^k$.

1 $k := 0$;
2 **repeat**
3 $\quad \lambda^{(k+1)} := r(X^0)$;
4 $\quad k := k + 1$;
5 $\quad X^k := \underset{X \in \text{feas}(\mathcal{C})}{\arg\max} \left\{a \times \Delta_\ell^h(\{X\} \cup \mathcal{S}_\ell \cup \{S_0\}) - \lambda^k \left[b \times q_\ell + c \times f^{loss}(X)\right]\right\}$;
6 **until** $\left|a \times \Delta_\ell^h(\{X\} \cup \mathcal{S}_\ell \cup \{S_0\}) - \lambda^k \left[b \times q_\ell + c \times f^{loss}(X)\right]\right| < \epsilon$;
7 **return** $X^k$;

**Algorithm 4:** Dinkelbach's Algorithm: Find one solution that optimizes $r$.

the stopping condition in Line 6, the algorithm terminates. Upon completion at iteration $k$, it returns an optimal solution $X^k$ that maximizes the ratio of relative solution diversity to relative deterioration in objective function quality. While each subproblem with integer variables is, in general, an $\mathcal{NP}$-hard problem, Dinkelbach's algorithm itself has been shown to have superlinear convergence (see, e.g., Schaible 1976).

## 4.4 Solution Engineering Framework Implemented in Constraint Programming

Constraint programming is a declarative technology for solving combinatorial problems, successfully used over the last thirty years in internet commerce, telecommunications, transportation, network management, supply chain management, and many other fields. A recent noteworthy application is the robot-lab Philae that landed on the comet 67P/Churyumov-Gerasimenko to conduct a series of experiments, which were scheduled using constraint programming (Simonin et al. 2015).

A constraint programming model is defined over a set $X$ of *domain variables*, a finite subset $D$ of $\mathbb{Z}$ called the *domain union*, and a set of constraints $C$. A constraint $c \in C$ is a pair $\{var(c), rel(c)\}$, where $var(c)$, its *scope*, is a subset of $X$, and $rel(c)$ is a relation that restricts the allowed combinations of simultaneous value assignments for the variables in $var(c)$. Each variable $x \in X$ is defined by a domain (unary) constraint, that holds if and only if $x$ takes its value in $D(x) \subseteq D$. The minimum and maximum values in $D(x)$ are $\underline{x}$ and $\overline{x}$. During the search for a solution, domains are modified by the solver, e.g., through a branch and bound scheme. The *search strategy* specifies the branching (variable order, domain cut/assignment policy). To avoid future useless branching below the current node in the search tree, each constraint has an associated *propagator*, which dynamically removes domain values that cannot be part of a solution to that constraint. Depending on the propagators used, domain reduction of constraints can be more or less effective: the notion of *consistency* characterizes propagator effectiveness. A propagator that only keeps domain values that participate in a solution to its constraint achieves *generalized arc-consistency* (GAC). A solution to a constraint is obtained when the domains of all its variables are singletons (that is, the variables are fixed). Any solution to the problem must satisfy all the stated constraints, including domain constraints. Optimization problems are modeled through a specific variable that is minimized or maximized.

### 4.4.1 Diversity Constraint

In our context, at each new generation of a solution, diversity should be measured according to all previous solutions within the set $\mathcal{S}$. We represent diversity through a constraint that embeds an integer variable *div* that expresses a diversity measure. We consider a decomposable diversity function, that is, upon generating a new solution, a component of the diversity value comes exclusively from previously computed solutions and can be aggregated into the current calculation. This approach was introduced in CP for a diversity constraint based on the Hamming distance in Hebrard et al. (2005). We extend it to any decomposable distance measure by considering solution pairs, that is, based on a function $\delta : (\mathbb{Z}^n)^2 \mapsto \mathbb{Z}_+$ that returns the distance of *a pair* of solutions by comparing the value of each variable in the two solutions. The distance is then an aggregation of the values obtained for each solution pair.

**Definition 3** (DiverseSum)**.** *Let prev be a (not necessarily strictly) positive constant,* $X = \{x_1, \ldots, x_n\}$ *be a set of variables, div a variable,* $\mathcal{S}$ *a set of previous* $k-1$ *assignments of* $X$ *and* $\delta : (\mathbb{Z}^n)^2 \mapsto \mathbb{Z}_+$ *be a distance measure, based on pairwise comparisons of the values taken by each variable in* $X$*. That is, for any solution* $S_j \in \mathcal{S}$*,*

$$\delta(X, S_j) = \sum_{i=1}^{n} \delta^x(S_j[i], x_i). \tag{11}$$

DiverseSum$(X, div, \mathcal{S}, \delta, prev)$ *is satisfied if and only if:*

$$div \leq \sum_{j=1}^{|\mathcal{S}|} \delta(X, S_j) + prev. \tag{12}$$

Definition 3 makes the assumption that variable *div* would likely be maximized. If one wished to minimize diversity between solutions, the arithmetic operator $\leq$ can easily be replaced by the operator $\geq$.

Propagation algorithms of this constraint, the objective constraint, quality notions, and search strategies are discussed in Section 1 of the Online Supplement.

## 4.5 Specialized Online Scheme

$\mathcal{S}_\ell := \emptyset$;
$h := 1$;
$M' := M$;
$prev := 0$;
$mindiv := \underline{div}$;
$sumdiv := 0$;
**while** $h \leq k_\ell$ **do**
  $prev := \max(h \times mindiv - sumdiv, 0)$ ;
  $D(div) := [prev, \overline{div}]$;
  $X := X \cup \mathcal{V}_\ell \cup \{div\} \cup \{z^{loss}\}$ ;
  $\mathcal{C} := \mathcal{C} \cup \mathcal{C}_\ell \cup \{\textsc{Table}(X, \mathcal{S}_\ell)\} \cup \{\mathcal{C}_{obj}(X, z^{loss}, S_0)\} \cup \{\textsc{DiverseSum}(X, \mathcal{S}_\ell \cup \{S_0\}, div, 0)\}$;

$$\begin{aligned} \text{maximize} \quad & r = \frac{a \times div}{b \times q_\ell + c \times z^{loss}} \\ \text{subject to} \quad & \mathcal{C}. \end{aligned}$$

  **if** *No solution* **then return** $\mathcal{S}_\ell$ ;
  Save the data $(r, div, q_\ell, f(X))$ ;
  $h := h + 1$;
  $\mathcal{S}_\ell := \mathcal{S}_\ell \cup \{$values assigned to variables $X$ in the solution$\}$;
  $sumdiv := sumdiv + div$;
  $M := M'$;
**return** $\mathcal{S}_\ell$;

**Algorithm 5:** Online generation of diverse solutions enhanced with quality notion $Q_l$, adapted to constraint programming.

We now implement Algorithm 1 in constraint programming. Using the notation of Algorithm 1, we consider that the new objective is a ratio between: (1) diversity, and (2) a scalar product of the quality score and loss from the best objective value found so far (solution $S_0$). Other ratios could be similarly considered. In addition to the notation of Algorithm 1, we declare:

- *div* as an integer variable used for diversity measure;
- $z^{loss}$ as an integer variable used for representing the loss in objective value;
- $\mathcal{C}_{obj}(X, z^{loss}, S_0)$ as a constraint used for representing the objective loss function $f^{loss}$, where $S_0$ is the first solution to $p$ generated in Algorithm 2.

Two existing online schemes (Hebrard et al. 2005, Petit and Trapp 2015) can be viewed as particular cases of Algorithm 5 (neither featured quality notions and, in Hebrard et al. (2005), it was restricted to satisfaction problems). Both represent the new problem objective through a variable, corresponding to $r$ in Algorithm 5. However, there is a technical difference: in our implementation, the diversity variable domain does not explicitly represent the sum of diversity values of all solution pairs, which makes our algorithm more robust to the generation of large solution sets. This avoids reaching a limit induced by maximum and minimum representable integers (see Section 2 of the Online Supplement for details).

A negative table constraint (see, e.g., Lecoutre et al. 2015) is used to forbid duplicating any solution in $\mathcal{S}$, that is, a constraint explicitly stated through a set of forbidden combinations of values for the variables in its scope. We call this constraint $\textsc{Table}(X, \mathcal{S}_\ell)$.

# 5. Computational Experiments

We implemented two prototypes of the solution engineering framework, one in mathematical programming (MP), and one in constraint programming (CP). All MP experiments were run on an Intel core i7-6700HQ computer with 2.60GHz and 64.0 GB RAM running 64-bit Windows 10 Enterprise, using IBM ILOG CPLEX optimizer (IBM 2017) in conjunction with the C callable library. All CP experiments were run on an Intel core i7-2720QM computer with 2.20GHz and 8.0 GB RAM running OS X 10.10.3, using the Java constraint programming solver Choco 3.3.3 (Prud'homme et al. 2015). These prototypes were implemented using the diversity measures and specialized algorithms as presented in Sections 4.4 and 4.3, respectively. Source code for both implementations is made available for download at the author's website (Trapp 2018).

As our framework takes any model and any formulation of quality notions as arguments, the solving efficacy may substantially vary depending on these parameters. Moreover, different technologies tend to outperform others on different problem classes. Hence, rather than compare technologies, the goal of our computational studies is to demonstrate that what we have proposed can be implemented in a realistic manner to achieve the desired output: a set of (near-) optimal, diverse solutions infused with quality notions. Further, we seek to ascertain whether it is possible to partition the solution space into sets of solutions having specific features, thereby making solutions in one set distinct from the solutions of other sets; a lack of such evidence would suggest that our approach is primarily theoretical.

To reach this objective, we analyzed the solution sets obtained on GAPa instances (Chu and Beasley 1997) of the GAP problem (as described in Section 2), with the following quality notions $Q_i, i \in \{2, 3, 4, 5\}$:

- Set 1: Original problem, no quality notion.
- Set 2: $Q_2$: Maximal cardinality distance between any two agents.
- Set 3: $Q_3$: Maximal resource distance between any two agents.
- Set 4: $Q_4$: Smooth violations between sequential jobs, $gap = 15$.
- Set 5: $Q_5$: Smooth violations between sequential jobs, $gap = 5$.

The above quality notions correspond to the examples discussed in Section 2. As all GAPa instances lead to similar outcomes, in the following we focus on the first instance of GAPa (Chu and Beasley 1997), namely GAPa_5_100, which has 5 constraints and 100 variables. Prior to presenting the experimental analysis and the results, we formally define the quality notions used. We define them as modeling blocks that are stated by their satisfaction condition (so-called *global constraints* in CP). We describe their implementation in both mathematical programming and constraint programming.

## 5.1 Quality Notions

The maximal pairwise distance in a set of integer variables can be encoded using the following global constraint.

**Definition 4** (MaxDistance, Katsirelos et al. (2012), Quimper et al. (2006))**.** MaxDistance *holds on integer variables* $\{x_1, \ldots, x_n\}$ *and an integer variable* $q$ *if and only if* $|x_i - x_j| \leq q$ *for all* $i \neq j$, $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n\}$.

Smoothing violations, that is, constraining the number of consecutive variables whose absolute difference is strictly greater than a certain gap, can be encoded using the following global constraint.

**Definition 5** (Smooth, Beldiceanu et al. (2007))**.** *Let* $X = \{x_1, \ldots, x_n\}$ *be a set of integer variables,* $p$ *a positive integer and* $q$ *an integer variable,* Smooth *holds if and only if* $q = |\{(x_i, x_{i+1}) \text{ such that } |x_i - x_{i+1}| > p\}|$.

### 5.1.1 Quality Notion Implementation in Mathematical Programming

Mathematical programming techniques can be used to augment the generalized assignment problem (GAP) with expressions of both MaxDistancenotions, as well as Smooth constraint violations. As discussed in Section 2, we assume there are $n$ jobs $J = \{1, 2, ..., n\}$, $m$ agents $A = \{1, 2, ..., m\}$, and the goal is to determine a minimum cost assignment subject to assigning each job to exactly one agent while satisfying a resource constraint for each agent. Assigning job $j$ to agent $i$ has a cost of $c_{ij}$ and consumes an amount $a_{ij}$ of resource, whereas the total amount of the resource available for agent $i$ is $cap_i$.

Define binary variables $x_{ij}$ as:

$$x_{ij} = \begin{cases} 1 & \text{if job } j \text{ is assigned to agent } i; \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

**Minimizing Maximum Distances Between Agents.** The number of jobs assigned to any agent $i$ is expressed as $\sum_{j=1}^{n} x_{ij}$, hence the cardinality distance of assigned jobs between two agents $i$ and $k$ is $|\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{kj}|$. To minimize the maximum cardinality distance of assigned jobs between all agents, we introduce new continuous variable $q \in \mathbb{R}_+$, as well as the following constraint set:

$$q \geq \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{kj}, \ i = 1, \ldots, m, \ k = 1, \ldots, m, \ i \neq k. \tag{14}$$

The variable $q$, to be minimized, will represent the quality notion of the maximum cardinality distance between all pairs of agents. This can be accomplished through one additional continuous variable $q$, and $m^2 - m$ linear constraints.

We also consider the maximum distance of resources consumed between any two agents. This can be modeled in analogous fashion by modifying (14) to account for total resources consumed by each agent:

$$q \geq \sum_{j=1}^{n} a_{ij}x_{ij} - \sum_{j=1}^{n} a_{kj}x_{kj},\ i = 1, \ldots, m,\ k = 1, \ldots, m,\ i \neq k. \tag{15}$$

The quality notion of the maximum distance of resources consumed between all pairs of agents is represented by $q \in \mathbb{R}_+$, and is to be minimized. This can similarly be accomplished through one additional continuous variable $q$, and $m^2 - m$ linear constraints.

**Minimizing Smooth Violations.** The SMOOTH condition captures the (absolute) change between consecutive (weighted) variable values. In particular, we consider the smoothness in the objective function value for the assignment of each job $j$ to agent $i$. Formally, the objective smoothness between two consecutive jobs $j$ and $j + 1$ can be expressed as:

$$\left| \sum_{i=1}^{m} c_{ij}x_{ij} - \sum_{i=1}^{m} c_{i,j+1}x_{i,j+1} \right|. \tag{16}$$

Suppose we are interested in finding solutions where the smoothness is bounded by (integer) value $s$. Define and precompute constants $M_{j,j+1}$ and $M_{j+1,j}$ for all $j = 1, \ldots, n-1$, as:

$$M_{j,j+1} = (\max_i c_{ij} - \min_i c_{i,j+1}) - s, \tag{17}$$

$$M_{j+1,j} = (\max_i c_{i,j+1} - \min_i c_{i,j}) - s. \tag{18}$$

Define $n - 1$ binary indicator variables $y_j$ as:

$$y_j = \begin{cases} 1 & \text{if objective smoothness (16) between jobs } j \text{ and } j+1 \text{ exceeds } s; \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

The SMOOTH condition can be implemented using the following two constraint sets:

$$\sum_{i=1}^{m} c_{ij}x_{ij} - \sum_{i=1}^{m} c_{i,j+1}x_{i,j+1} \leq s + M_{j,j+1}y_j,\ j = 1, \ldots, n-1, \tag{20}$$

$$\sum_{i=1}^{m} c_{i,j+1}x_{i,j+1} - \sum_{i=1}^{m} c_{ij}x_{ij} \leq s + M_{j+1,j}y_j,\ j = 1, \ldots, n-1, \tag{21}$$

with the objective of minimizing the number of violations $\sum_{j=1}^{n-1} y_j$. Hence, SMOOTH can be implemented in mathematical programming via $n-1$ additional binary variables and $2n-2$ linear constraints.

### 5.1.2 Quality Notion Implementation in Constraint Programming

While various propagators have been published for the MaxDistance and Smooth global constraints, they are not implemented by default in most systems. Alternatively, the two global constraints can be decomposed using constraints that are standardly available in CP solvers. To decompose MaxDistance, that holds on on integer variables $\{x_1, \ldots, x_n\}$ and an integer variable $q$, one may state integer variables $y_{ij}$ for each $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n\}$, $i < j$, and state the following constraints:

$$\forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, n\}, i < j, y_{ij} = |x_i - x_j|. \tag{22}$$

$$\left( \max_{(i,j):i<j} y_{ij} \right) \leq q. \tag{23}$$

To decompose Smooth one may define an integer variable $y_i$ and a binary variable $b_i$ for each $i \in \{1, \ldots, n-1\}$, and state the following constraints:

$$\forall i \in \{1, \ldots, n-1\}, |x_i - x_{i+1}| = y_i \text{ and } b_i = (y_i > p). \tag{24}$$

$$q = \sum_{i=1}^{n-1} b_i. \tag{25}$$

In the above definition, the notation $b_i = (y_i > p)$ is used to state the binary constraint:

$$[b_i = 0 \wedge y_i \leq p] \vee [b_i = 1 \wedge y_i > p].$$

In our experiments we made the choice to present results obtained with those decompositions, so as to remain faithful to the objective of using our framework without any problem specific tuning, as a non-expert user might do.

### 5.1.3 Experimental Analysis and Results

For each prototype (MP and CP) and for each of Sets 1 to 5, we generated 50 solutions, using a 5 minute time limit per solve. The quality score vector has also been created for every solution in every set by evaluating and recording the respective quality notions. We denote by $q_1$ the value of the original objective of the GAP in each solution, while $q_i$ records the value of quality notion $Q_i$ in the solution, $i \in \{2, 3, 4, 5\}$.

Regarding algorithmic specializations, for the sake of consistency we include in both technologies a constraint à la Line 3 of Algorithm 2 that bounds the original objective function according to within 3% of the best value found so far. Hence, $K = 0.03 \cdot f(S_0)$. Moreover, in both technologies the ratio to be maximized in our experiments was the following:

$$r = \frac{500 \times \Delta_\ell^h(\{X\} \cup \mathcal{S}_\ell \cup \{S_0\})}{50 \times q_\ell + f^{loss}(X)}, \tag{26}$$

except concerning Set 1, where the $q_\ell$ component is absent from the ratio. These parameters provided for diversity within the solution sets. We first discuss the relative loss concerning the GAP objective value in the solution sets.

| | Set 1 | Set 2 | Set 3 | Set 4 | Set 5 |
|---|---|---|---|---|---|
| MP | 0.05 | 0.47 | 1.1 | 2.94 | 2.94 |
| CP | 0.05 | 0.49 | 2.77 | 2.44 | 2.84 |

Table 2: Average gap from the optimal objective, in %.

Table 2 presents the average gap from the optimal objective value across all sets, and in every case demonstrates that it is reasonable, which is to be expected as per the implemented bound on the original objective function value. Across the five sets, there were some minor differences in optimality gaps, essentially highlighting that two distinct technologies were used for implementations, including different diversity measures.

To illustrate the resulting variety of generated solutions, in Table 3 we computed the number and percentage of unique score vectors with both the objective value included, i.e., $(q_1, \ldots, q_5)$, as well as not included, i.e., $(q_2, \ldots, q_5)$. By unique, we mean that we do not count score vectors appearing more than once in the same set. While it is possible that two distinct solutions map to the same score vector, in Table 3 we focus on assessing whether the solutions in sets have differences that are identifiable through the quality notions.

Each row of Table 3 considers 1) a technology (MP or CP), 2) an absolute count (#) or percentage (%), and 3) either $(q_1, \ldots, q_5)$ or $(q_2, \ldots, q_5)$. The columns represent the respective performance across the solution sets as well as two sets formed from unions. So, the initial entry 20/50 for MP indicates that among the 50 solutions in Set 1, there were 20 unique score vectors as expressed by their entries $q_1, \ldots, q_5$ (that is, 40%). In light of the fact that two distinct technologies and diversity metrics were used to generate these solutions, the consistency of the results in Table 3 is rather remarkable.

| | Set 1 | Set 2 | Set 3 | Set 4 | Set 5 | $\cup_{1-5}$ | $\cup_{2-5}$ |
|---|---|---|---|---|---|---|---|
| MP: $(q_1, \ldots, q_5)$ | $\frac{20}{50}$ (40%) | $\frac{35}{50}$ (70%) | $\frac{35}{50}$ (70%) | $\frac{50}{50}$ (100%) | $\frac{50}{50}$ (100%) | $\frac{190}{250}$ (76%) | $\frac{170}{200}$ (85%) |
| MP: $(q_2, \ldots, q_5)$ | $\frac{19}{50}$ (38%) | $\frac{21}{50}$ (42%) | $\frac{14}{50}$ (28%) | $\frac{50}{50}$ (100%) | $\frac{50}{50}$ (100%) | $\frac{154}{250}$ (62%) | $\frac{135}{200}$ (68%) |
| CP: $(q_1, \ldots, q_5)$ | $\frac{19}{50}$ (38%) | $\frac{29}{50}$ (58%) | $\frac{42}{50}$ (84%) | $\frac{50}{50}$ (100%) | $\frac{50}{50}$ (100%) | $\frac{190}{250}$ (76%) | $\frac{171}{200}$ (86%) |
| CP: $(q_2, \ldots, q_5)$ | $\frac{19}{50}$ (38%) | $\frac{14}{50}$ (28%) | $\frac{18}{50}$ (36%) | $\frac{50}{50}$ (100%) | $\frac{50}{50}$ (100%) | $\frac{151}{250}$ (61%) | $\frac{132}{200}$ (66%) |

Table 3: Fraction and percent of unique quality notion vectors $(q_i, \ldots, q_j)$.

Perhaps the most interesting comparison is between Set 1 (no quality notion) with the union of Sets 1 to 5 (column $\cup_{1-5}$) – and also with Sets 2 to 5 (column $\cup_{2-5}$). The percentages of unique vectors are significantly higher for the set unions, supporting our reasoning that the system generates solutions that have specific characteristics, that may otherwise remain undiscovered using classical online diversity schemes. For column $\cup_{1-5}$, the fact that both the MP and CP technologies identified 190 (out of 250) unique solutions indicates

that no solutions were shared in common between the individual five sets (as 190 is the number of unique solutions across all individual sets), further underscoring the usefulness of the methodology. However, more analysis should be performed, for example to determine if solutions in Sets 2 to 5 have quality notion scores that cannot be reached in Set 1. This, and related queries, are topics for future research.

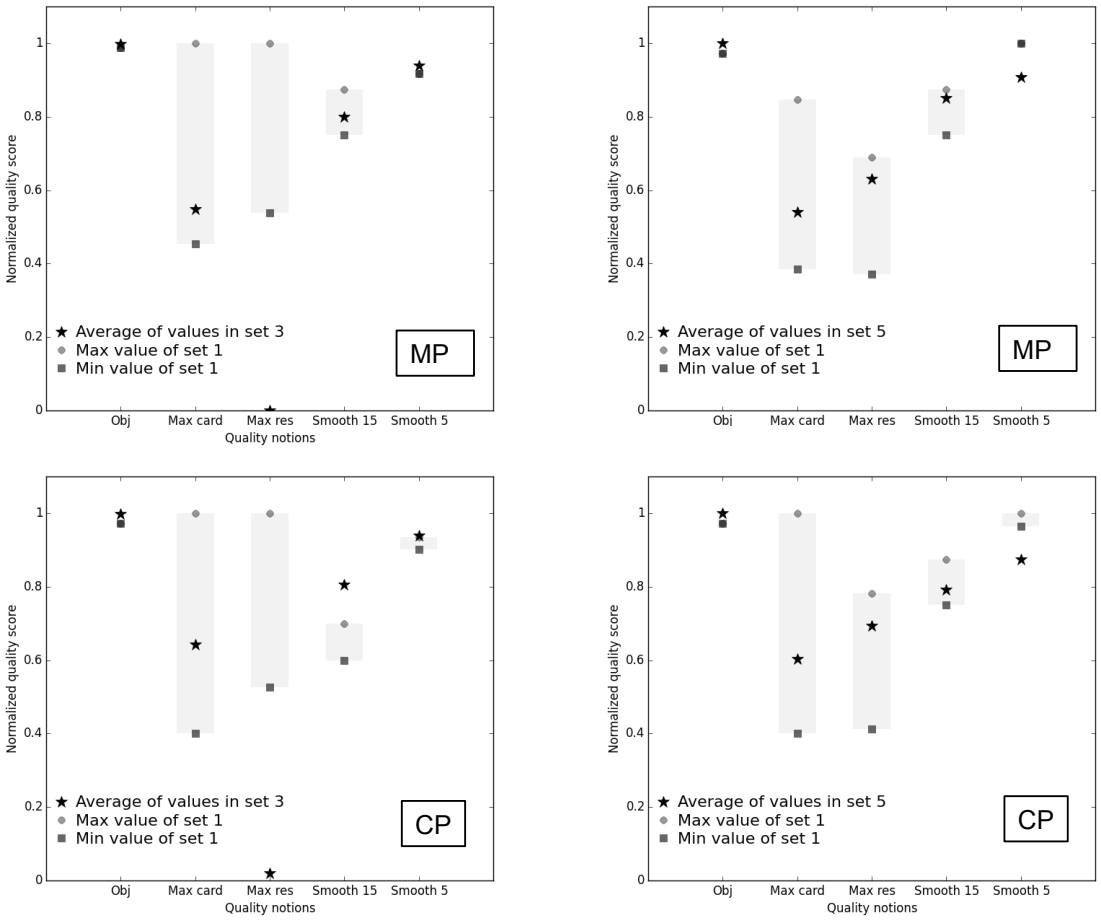


Figure 3: Comparing metrics on two sets of solutions to GAPa_5_100 instance in MP and CP.

Figure 3 illustrates the minimum (dark gray points) and maximum (lighter gray points) score vector values of $(q_1, \ldots, q_5)$ in Set 1 normalized over $[0, 1]$, and compares these with the *average score vector values of* $(q_1, \ldots, q_5)$ in Sets 3 (left subfigures) and 5 (right subfigures), normalized over $[0, 1]$. Mathematical programming is depicted in the top subfigures, with constraint programming in the bottom. The $x$-axis corresponds to the quality notions: $q_1$ is "Obj", $q_2$ is "Max card", $q_3$ is "Max res", $q_4$ is "Smooth 15", $q_5$ is "Smooth 5". The results obtained from sets other than 3 and 5 are similar. The figure shows that concerning the

quality notion minimized in Sets 3 or 5, respectively, the average values for both MP and CP are lower than the best value found in Set 1, further demonstrating that the selected solutions were only discovered through the quality notions stated in our framework. Moreover, it is apparent in Figure 3 that there are only slight differences in the MP and CP technologies.

This experiment demonstrates the possibility for a broader exploration of the solution space by infusing desirable features via solution engineering, based on rigorously defined metrics. Additional limited experimentation, not detailed herein for the sake of brevity, indicates that similar results can be obtained by aggregating multiple quality notions. This demonstrates that it is not necessary to restrict each set to contain only extreme solutions for one quality notion, offering even more perspectives to optimization practitioners, for example to simplify the process of iterative design for real-world optimization modeling.

# 6. Conclusion

We propose the *solution engineering* framework to enable the discovery of multiple optimal or near-optimal, diverse solutions for problems of a combinatorial nature, that are further infused with desirable features via general quality notions selected by the user. Our framework requires an original combinatorial model expressed in a particular technology (such as mathematical programming or constraint programming), together with one or more ad-hoc quality notions that can be expressed in that particular technology. We customize the framework to two specific technologies, namely mathematical programming and constraint programming, and subsequently demonstrate its practicality by implementing and further conducting computational experiments on existing test instances from the literature. Our computational results largely underscore that it is indeed possible to construct sets containing multiple, diverse solutions that are infused with various desirable features via quality notions. We are unaware of any other means to facilitate recovery of such sets of solutions, hence they would otherwise remain undiscovered.

For a particular combinatorial problem, we propose that the user simply selects general quality notions, external to the original model. Building upon earlier work in identifying diverse optima and near-optima to combinatorial problems (Trapp and Konrad 2015, Petit and Trapp 2015), we then propose an approach that sequentially constructs a set of diverse solutions that emphasize each quality notion together with the original objective (or if none, omitted). Subsequently taking the union of these sets effectively constructs a superset containing solutions that have been infused with dimensions of quality. If the resulting set of solutions is large, it can then be further explored, analyzed, ranked and categorized via data science technologies, thereby complementing the natural strengths of human decision-making. While the present work advocates for the methodology, the data science implementation of such a system is left for future work.

# 7. Appendix: Constraint Programming Technical Developments

## 7.1 Propagating the Diversity Constraint

Definition 3 makes the assumption that variable *div* would likely be maximized. If one wished to minimize diversity between solutions, the arithmetic operator $\leq$ can easily be replaced by the operator $\geq$. Notice that there is no need to utilize the Hamming or other commonly-used mathematical distances. For instance, one may define the distance $\delta^x$ between the values $v_1$ and $v_2$ taken by variable $x$ using an ad hoc constraint, e.g., $\delta^x = \min\{\max\{0, |v_1 - v_2| - K\}, 1\}$, where $K \geq 0$ is a constant. This example states that if two values are such that the absolute value of their difference is less than or equal to $K$ then they are considered as "not diverse" (i.e., the diversity value is 0), otherwise the diversity value is 1.

```
1  dist : integer matrix |X| × max_{x∈X}(|D(x)|);
2  foreach x_i ∈ X do
3      foreach v_j ∈ D(x_i) do
4          dom := copy of D(x_i);
5          D(x_i) := {v_j};
6          dist[i][j] := Σ_{k=1}^{|S|} δ^x(x_i, S_k[i]);
7          D(x_i) := dom;
8  # Update div
9  maxSum := Σ_{i=1}^{n} max_{j∈[0,|dist[i]|]}(dist[i][j]) + prev;
10 div̄ := min(maxSum, div̄);                    # Fail if div̄ < div̲;
11 # Update problem variables
12 foreach x_i ∈ X do
13     foreach v_j ∈ D(x_i) do
14         if maxSum − max_{l∈[0,|dist[i]|]}(dist[i][l]) + dist[i][j] < div̲ then
15             D(x_i) := D(x_i) \ {v_j};
```

**Algorithm 6:** filterDiverseSum.

The DiverseSum constraint of Definition 3 requires the design of a specific propagator. We propose a technique based on a principle initially provided for the specific case of the Hamming distance by Hebrard et al. (2005). Algorithm 6 generalizes it to any function $\delta$. Assuming $\delta^x(S_k[i], x_i)$ can be computed in constant time, the time complexity of Algorithm 6 is $O(|\mathcal{S}| \cdot \sum_{x_i \in X} |D(x_i)|)$.

**Proposition 1.** *Algorithm 6 is correct.*

Proof: A propagator is correct if it does not remove from domains values that can be part of an assignment satisfying the constraint. Consider first *div*. By construction, *maxSum* is the maximum possible diversity value of $sum_{j=1}^{|\mathcal{S}|}\delta(X, S_j) + prev$ given the domains of

variables in $X$. From Definition 3, we must have $div \leq \sum_{j=1}^{|\mathcal{S}|} \delta(X, S_j) + prev$: the update of $\overline{div}$ (line 10) is safe. Let us now consider any variable $x_i \in X$. Given $v_j \in D(x_i)$, by construction of *maxSum* and *dist*, the maximum possible diversity of the problem when $x_i = v_j$ is $maxSum - \max_{l \in [0,|dist[i]|]}(dist[i][l]) + dist[i][j]$, as the value $\max_{l \in [0,|dist[i]|]}(dist[i][l])$ exclusively depends on variable $x_i$ in the current solutions of $\mathcal{S}$ (that is, not on other variables in $X$). Therefore, if $maxSum - \max_{l \in [0,|dist[i]|]}(dist[i][l]) + dist[i][j] < \underline{div}$ the condition $div \leq \sum_{j=1}^{|\mathcal{S}|} \delta(X, S_j) + prev$ cannot be satisfied with $x_i = v_j$. Value $v_j$ can be removed from $D(x_i)$ (line 15). No other condition leads to value removals in Algorithm 6. □

**Proposition 2.** *Algorithm 6 achieves generalized arc-consistency.*

Proof: Assume Algorithm 6 does not achieves GAC: a value is still in a domain although it cannot be part of an assignment that satisfies the constraint. First, consider value $v \in D(div)$. If the $v$ cannot be part of a solution, after having applied the algorithm there must exist a valid assignment of $X \cup \{div\}$ such that $v > \sum_{j=1}^{|\mathcal{S}|} \delta(X, S_j) + prev$. This stands in contradiction with Line 10, by construction of *maxSum*. Secondly, consider $v_j \in D(x_i)$, $x_i \in X$, such that $v_j$ cannot be part of a solution. Computing *dist* matrix and $\max_{j \in [0,|dist[i]|]}(dist[i][j])$ while reducing $D(x_i)$ to $\{v_j\}$ should lead to $\sum_{i=1}^{n} \max_{j \in [0,|dist[i]|]}(dist[i][j]) + prev < \underline{div}$. By construction of *maxSum* and since $\max_{l \in [0,|dist[i]|]}(dist[i][l])$ exclusively depends on variable $x_i$, this is in contradiction with the removal condition of Line 14. As DiverseSum scope is $X \cup \{div\}$, the proposition holds. □

## 7.2 Propagating Objective Function and Quality Notions

Constraint solvers routinely embed predefined constraints for designing arithmetic expressions on variables and constants, using standard operators $+$, $*$, $-$, $/$, as well as predefined constraints, such as $|x - y| = z$. Recall that, using integer domains, / is integer division. Therefore, encoding $f^{loss}$ (to state the ratio $r$ in Algorithm 1), as well as performing a weighted sum or a ratio of quantities that are represented through variables, is standard in CP. When no state-of-the-art constraint exists for representing a desired quality notion or when a constraint is not in the list of predefined constraints of the solver, the quality notion can be decomposed into a set of constraints (see, e.g., Bessiere et al. (2009)).

## 7.3 Search Strategies

A simple and effective way to adapt the search strategy *str* originally defined in the model of problem $p$ is to assign first all the model variables using *str*, and then the additional variables for the new objective and quality notion, using a lexicographical order, for instance. As such variables represent a quantity stemming from model variables, this simple principle appears to be quite robust. This was confirmed by our experiments. The solving process may be affected, however, if stating the quality notion $Q_\ell$ requires the addition of many auxiliary variables in addition to $q_\ell$. In this case, the principle of delaying their assignment after model variables remains valid, but it is necessary to define an appropriate strategy for the new variables, rather than a simple lexicographical order.

# 8. Appendix: Constraint Programming Diversity Variable Update

The new objective is stated on the diversity variable, the objective variable of the original model and quality notion variables. Variables are associated with domains whose minimum and maximum values are subject to the limit of integer size of solvers; existing approaches may prevent from generating large sets of solutions, due to the domain of variables used for diversity and quality notions. The principle used in Algorithm 5 for updating the diversity variable is different: we never increase its maximum value. To do so, we adjust the minimum value to be the difference between the product of the number of previously generated solutions and the initial required minimum diversity, less the current sum of diversity (Line 8). If the result is negative then the minimum required diversity for the next solution is 0. This adjustment avoids too much diversity from the next solution to be generated, while the diversity variable domain upper-bound remains the same at all iterations. Observe that in any case all the solutions are distinct, thanks to the table constraint stated in line 11 of Algorithm 5.

# References


Gediminas Adomavicius and YoungOk Kwon. Optimization-based approaches for maximizing aggregate recommendation diversity. *INFORMS Journal on Computing*, 26(2):351–369, 2014.

Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic csps application to configuration. *Artificial Intelligence*, 135(1-2):199–234, 2002.

Achilles A. Armenakis and Arthur G. Bedeian. Organizational change: A review of theory and research in the 1990s. *Journal of Management*, 25(3):293–315, 1999.

E. Balas and R. Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1):61–69, 1972.

Gabriele Barbieri, François Pachet, Pierre Roy, and Mirko Degli Esposti. Markov constraints for generating lyrics with style. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, pages 115–120, 2012. URL `https://doi.org/10.3233/978-1-61499-098-7-115`.

Nicolas Beldiceanu and Helmut Simonis. A constraint seeker: Finding and ranking global constraints from examples. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2011. ISBN 978-3-642-23785-0. doi: 10.1007/978-3-642-23786-7_4. URL `http://dx.doi.org/10.1007/978-3-642-23786-7_4`.

Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62, 2007. doi: 10.1007/s10601-006-9010-8. URL `http://dx.doi.org/10.1007/s10601-006-9010-8`.

D. Bertsimas, K. Natarajan, and C.-P. Teo. Persistence in discrete optimization under data uncertainty. *Math. Program.*, pages 251–274, 2006.

Dimitris Bertsimas, Vivek F. Farias, and Nikolaos Trichakis. On the efficiency-fairness trade-off. *Management Science*, 58(12):2234–2250, 2012. doi: 10.1287/mnsc.1120.1549. URL `http://dx.doi.org/10.1287/mnsc.1120.1549`.

Christian Bessière, Emmanuel Hebrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Among, common and disjoint constraints. In Brahim Hnich, Mats Carlsson, François Fages, and Francesca Rossi, editors, *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2005, Uppsala, Sweden, June 20-22, 2005, Revised Selected and Invited Papers*, volume 3978 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2005.

Christian Bessiere, George Katsirelos, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Decompositions of all different, global cardinality and related constraints. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 419–424, 2009.

Christian Bessiere, Emmanuel Hebrard, George Katsirelos, Zeynep Kiziltan, Émilie Picard-Cantin, Claude-Guy Quimper, and Toby Walsh. The balance constraint family. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 174–189. Springer, 2014. URL `http://dx.doi.org/10.1007/978-3-319-10428-7_15`.

Christian Bessiere, Frederic Koriche, Nadjib Lazaar, and Barry OSullivan. Constraint acquisition. *Artificial Intelligence*, pages 1–50: to appear (preprint version available: http://www.lirmm.fr/ bessiere/Site/Papers.html), 2016.

George EP Box, William Gordon Hunter, and J Stuart Hunter. *Statistics for experimenters: Design, innovation, and discovery*. Wiley, 2005.

Gerald G Brown, Robert F Dell, and R Kevin Wood. Optimization and persistence. *Interfaces*, 27 (5):15–37, 1997.

Hadrien Cambazard and Jean-Guillaume Fages. New filtering for atmostnvalue and its weighted variant: A lagrangian approach. *Constraints*, 20(3):362–380, 2015.

Jeffrey D Camm. ASP, the art and science of practice: A (very) short course in suboptimization. *Interfaces*, 44(4):428–431, 2014.

Rodolfo Carvajal, Miguel Constantino, Marcos Goycoolea, Juan Pablo Vielma, and Andrés Weintraub. Imposing connectivity constraints in forest planning models. *Operations Research*, 61 (4):824–836, 2013.

Gilles Chabert, Luc Jaulin, and Xavier Lorca. A constraint on the number of distinct vectors with application to localization. In Ian P. Gent, editor, *Proceedings CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2009.

P. C. Chu and J. E. Beasley. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24(1):17–23, 1997.

E. Danna and D. L. Woodruff. How to select a small set of diverse solutions to mixed integer programming problems. *Operations Research Letters*, 37(4):255–260, 2009.

E. Danna, M. Fenelon, Z. Gu, and R. Wunderling. Generating multiple solutions for mixed integer programming problems. In Matteo Fischetti and David Williamson, editors, *Integer Programming and Combinatorial Optimization*, volume 4513 of *Lecture Notes in Computer Science*, pages 280–294. Springer Berlin Heidelberg, 2007.

W. Dinkelbach. On non-linear fractional programming. *Management Science*, 13(7):492–498, 1967.

Roman Efremov, David Rios Insua, and Alexander Lotov. A framework for participatory decision support using pareto frontier visualization, goal identification and arbitration. *European Journal of Operational Research*, 199(2):459–467, 2009.

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

T. Eiter, E. Erdem, H. Erdogan, and M. Fink. Finding similar/diverse solutions in answer set programming. *TPLP*, 13(3):303–359, 2013.

Marshall L Fisher, Ramchandran Jaikumar, and Luk N Van Wassenhove. A multiplier adjustment method for the generalized assignment problem. *Management Science*, 32(9):1095–1103, 1986.

F. Glover, A. Løkketangen, and D. L. Woodruff. Scatter search to generate diverse MIP solutions. In M. Laguna and J. L. González-Velarde, editors, *OR Computing Tools for Modeling, Optimization and Simulation*, volume 12 of *Interfaces in Computer Science and Operations Research*, pages 299–317. Kluwer Academic Publishers, 2000.

Fred Glover, Ching-Chung Kuo, and Krishna S Dhir. Heuristic algorithms for the maximum diversity problem. *Journal of Information and Optimization Sciences*, 19(1):109–132, 1998.

P. Greistorfer, A. Løkketangen, S. Voß, and D. L. Woodruff. Experiments concerning sequential versus simultaneous maximization of objective function and distance. *Journal of Heuristics*, 14(6):613–625, 2008.

Tarik Hadzic, Alan Holland, and Barry O'Sullivan. Reasoning about optimal collections of solutions. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 2009.

E. Hebrard, B. Hnich, B. O'Sullivan, and T. Walsh. Finding diverse and similar solutions in constraint programming. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 372–377. American Association for Artificial Intelligence, 2005.

Emmanuel Hebrard, Barry O'Sullivan, and Toby Walsh. Distance constraints in constraint satisfaction. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 106–111, 2007. URL `http://ijcai.org/proceedings/2007`.

Emmanuel Hebrard, Dániel Marx, Barry O'Sullivan, and Igor Razgon. Soft constraints of difference and equality. *J. Artif. Intell. Res. (JAIR)*, 41:97–130, 2011.

P. Van Hentenryck, C. Coffrin, and B. Gutkovich. Constraint-based local search for the automatic generation of architectural tests. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP*, pages 787–801, 2009a.

Pascal Van Hentenryck, Carleton Coffrin, and Boris Gutkovich. Constraint-based local search for the automatic generation of architectural tests. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 787–801. Springer, 2009b.

IBM. *IBM ILOG CPLEX 12.7 User's Manual*. IBM ILOG CPLEX Division, Incline Village, NV, 2017.

Esra Karasakal and Murat Köksalan. Generating a representative subset of the nondominated frontier in multiple criteria decision making. *Operations research*, 57(1):187–199, 2009.

George Katsirelos, Nina Narodytska, and Toby Walsh. The seqbin constraint revisited. In *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pages 332–347, 2012.

Zeynep Kiziltan, Marco Lippi, and Paolo Torroni. Constraint detection in natural language problem descriptions. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 744–750. IJCAI/AAAI Press, 2016. ISBN 978-1-57735-770-4. URL `http://www.ijcai.org/Abstract/16/111`.

Ching-Chung Kuo, Fred Glover, and Krishna S Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185, 1993.

Christophe Lecoutre, Chavalit Likitvivatanavong, and Roland H. C. Yap. STR3: A path-optimal filtering algorithm for table constraints. *Artif. Intell.*, 220:1–27, 2015.

A. Løkketangen and D. L. Woodruff. A distance function to support optimized selection decisions. *Decision Support Systems*, 39(3):345–354, 2005.

T. Morrison. A new paradigm for robust combinatorial optimization: Using persistence as a theory of evidence. *Ph.D dissertation*, 2010.

Alexander Nadel. Generating diverse solutions in SAT. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 287–301. Springer, 2011.

Nina Narodytska, Thierry Petit, Mohamed Siala, and Toby Walsh. Three generalizations of the FOCUS constraint. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 630–636. IJCAI/AAAI, 2013. URL `http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6829`.

Samir Ouis, Narendra Jussien, and Patrice Boizumault. k-relevant explanations for constraint programming. In Ingrid Russell and Susan M. Haller, editors, *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference, May 12-14, 2003, St. Augustine, Florida, USA*, pages 192–196. AAAI Press, 2003. ISBN 1-57735-177-0. URL `http://www.aaai.org/Library/FLAIRS/2003/flairs03-038.php`.

Gilles Pesant and Jean-Charles Régin. Spread: A balancing constraint based on statistics. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 of *Lecture Notes in Computer Science*, pages 460–474, 2005.

Thierry Petit and Lolita Petit. Optimizing molecular cloning of multiple plasmids. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 773–779, 2016.

Thierry Petit and Andrew C. Trapp. Finding diverse solutions of high quality to constraint optimization problems. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 260–267, 2015.

Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2015. URL `http://www.choco-solver.org`.

Claude-Guy Quimper, Alejandro López-Ortiz, and Gilles Pesant. A quadratic propagator for the inter-distance constraint. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 123–128, 2006.

S. Schaible. Fractional programming. II, on Dinkelbach's algorithm. *Management Science*, 22(8): 868–873, 1976.

Pierre Schaus, Yves Deville, and Pierre Dupont. Bound-consistent deviation constraint. In Christian Bessiere, editor, *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, volume 4741 of *Lecture Notes in Computer Science*, pages 620–634. Springer, 2007a. URL `http://dx.doi.org/10.1007/978-3-540-74970-7_44`.

Pierre Schaus, Yves Deville, Pierre Dupont, and Jean-Charles Régin. The deviation constraint. In Pascal Van Hentenryck and Laurence A. Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 4th International Conference, CPAIOR 2007, Brussels, Belgium, May 23-26, 2007, Proceedings*, volume 4510 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2007b.

Pierre Schaus, Pascal Van Hentenryck, and Jean-Charles Régin. Scalable load balancing in nurse to patient assignment problems. In Willem Jan van Hoeve and John N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*, volume 5547 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2009. URL `http://dx.doi.org/10.1007/978-3-642-01929-6_19`.

Daniel Schell. Optimality in musical melodies and harmonic progressions: The travelling musician. *European Journal of Operational Research*, 140(2):354–372, 2002.

P. Schittekat and K. Sörensen. Supporting 3PL decisions in the automotive industry by generating diverse solutions to a large-scale location-routing problem. *Operations Research*, 57(5):1058–1067, 2009.

Samir Sebbah, Claire Bagley, Mike Colena, and Serdar Kadioglu. Service availability optimization in cloud-based in-memory data grids. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - CP 2016, 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings. To appear*, Lecture Notes in Computer Science. Springer, 2016.

Jung P Shim, Merrill Warkentin, James F Courtney, Daniel J Power, Ramesh Sharda, and Christer Carlsson. Past, present, and future of decision support technology. *Decision Support Systems*, 33(2):111–126, 2002.

Gilles Simonin, Christian Artigues, Emmanuel Hebrard, and Pierre Lopez. Scheduling scientific experiments for comet exploration. *Constraints*, 20(1):77–99, 2015.

Mehrdad Tamiz, Dylan Jones, and Carlos Romero. Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of Operational Research*, 111(3): 569–581, 1998.

A. C. Trapp. Source code for solution engineering. `http://users.wpi.edu/~atrapp/tools-and-software.htm`, 2018.

A. C. Trapp and R. A. Konrad. Finding diverse optima and near-optima to binary integer programs. *IIE Transactions*, 47(11):1300–1312, 2015.

Jung-Fa Tsai, Ming-Hua Lin, and Yi-Chung Hu. Finding multiple solutions to general integer linear programs. *European Journal of Operational Research*, 184(2):802–809, 2008.

Philip Voll, Mark Jennings, Maike Hennen, Nilay Shah, and André Bardow. The optimum is not enough: A near-optimal solution paradigm for energy systems synthesis. *Energy*, 82:446–456, 2015.