Actionable Subgroup Discovery and Urban Farm Optimization

Alexandre Millot, Romain Mathonat, Rémy Cazabet, Jean-François Boulicaut

Actionable Subgroup Discovery and Urban Farm Optimization

Alexandre Millot[1], Romain Mathonat[1,2], Rémy Cazabet[3], and Jean-François Boulicaut[1]

[1] Univ de Lyon, CNRS, INSA Lyon, LIRIS, UMR5205, F-69621, France
{firstname.lastname}@insa-lyon.fr
[2] Atos, F-69100 Villeurbanne, France
[3] Univ de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205, F-69622, France
{firstname.lastname}@univ-lyon1.fr

Abstract. Designing, selling and/or exploiting connected vertical urban farms is now receiving a lot of attention. In such farms, plants grow in controlled environments according to recipes that specify the different growth stages and instructions concerning many parameters (e.g., temperature, humidity, CO_2, light). During the whole process, automated systems collect measures of such parameters and, at the end, we can get some global indicator about the used recipe, e.g., its yield. Looking for innovative ideas to optimize recipes, we investigate the use of a new optimal subgroup discovery method from purely numerical data. It concerns here the computation of subsets of recipes whose labels (e.g., the yield) show an interesting distribution according to a quality measure. When considering optimization, e.g., maximizing the yield, our virtuous circle optimization framework iteratively improves recipes by sampling the discovered optimal subgroup description subspace. We provide our preliminary results about the added-value of this framework thanks to a plant growth simulator that enables inexpensive experiments.

Keywords: Subgroup Discovery · Virtuous Circle · Urban Farms.

1 Introduction

Conventional farming methods have to face many challenges like, for instance, soil erosion and/or an overuse of pesticides. The crucial problems related to climate change also stimulate the design of new production systems. The concept of urban farms (see, e.g., AeroFarms, FUL, Infarm[1]) could be part of a solution. It enables the growth of plants in fully controlled environments close to the place where consumers are [8]. Most of the crop protection chemical products can be removed while being able to optimize both the quantity and the quality of plants (e.g., improving the flavor [9] or their chemical proportions [20]).

Urban farms can generate large amounts of data that can be pushed towards a cloud environment such that various machine learning and data mining methods

[1] https://aerofarms.com/, http://www.fermeful.com/, https://infarm.com/.

can be used. We may then provide new insights about the plant growth process itself (discovering knowledge about not yet identified/understood phenomena) but also offer new services to farm owners. We focus here on services that rely on the optimization of a given target variable, e.g., the yield. The number of parameters influencing plant growth can be relatively large (e.g., temperature, hygrometry, water pH level, nutrient concentration, LED lighting intensity, CO_2 concentration). There are numerous ways of measuring the crop end-product (e.g., energy cost, plant mass and size, flavor and chemical properties). In general, for a given type of plants, expert knowledge exists that concerns the available sub-systems (e.g., to model the impact of nutrient on growth, the effect of LED lighting on photosynthesis, the energy consumption w.r.t. the temperature instruction) but we are far from a global understanding of the interaction between the various underlying phenomena. In other terms, setting the optimal instructions for the diverse set of parameters given an optimization task remains an open problem.

We want to address such an issue by means of data mining techniques. Plant growth recipes are made of instructions in time and space for many numerical attributes. Once a recipe is completed, collections of measures have been collected and we assume that at least one numerical target label value is available, e.g., the yield. Can we learn from available recipe records to suggest new ones that should provide better results w.r.t. the selected target attribute ? For that purpose, we investigate the use of subgroup discovery [12,21]. It aims at discovering subsets of objects - called subgroups - with high quality according to a quality measure calculated on the target label. Such a quality measure has to capture deviations in the target label distribution when we consider the overall data set or the considered subset of objects. When addressing only subgroup discovery from numerical data, a few approaches for numerical attributes [6,15] and numerical target labels [14] have been described. To the best of our knowledge, the reference algorithm for subgroup discovery in purely numerical data is SD-Map⋆ [14]. However, like other methods, it uses discretization and leads to loss of information and sub-optimal results.

Our first contribution concerns the proposal of a simple branch and bound algorithm called MinIntChange4SD that exploits the exhaustive enumeration strategy from [11] to achieve a guaranteed optimal subgroup discovery in numerical data without any discretization. Discussing details about this algorithm is out of the scope of this paper and we recently designed a significantly optimized version of MinIntChange4SD in [17]. Our main contribution concerns a new methodology for plant growth recipe optimization that (i) uses MinIntChange4SD to find the optimal subgroup of recipes and (ii) exploits the subgroup description to design better recipes which can in turn be analyzed with subgroup discovery, and so on.

The paper is organized as follows. Section 2 formalizes the problem. In Section 3, we discuss related works and their limitations. In Section 4, we introduce our new optimal subgroup discovery algorithm and we detail our framework for plant growth recipe optimization. An empirical evaluation of our method is in Section 5. Section 6 briefly concludes.

	m_1	m_2	T
g1	1	1	15
g2	1	2	30
g3	2	2	60
g4	3	2	40
g5	3	3	70
g6	4	3	85

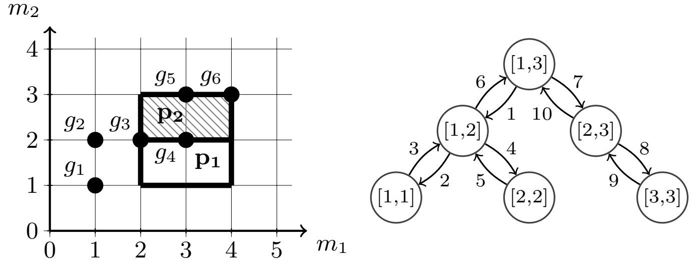

Fig. 1. (**left**) Purely numerical dataset. (**center**) Non-closed ($\mathrm{p}_1 = \langle[2,4],[1,3]\rangle$, non-hatched) and closed ($\mathrm{p}_2 = \langle[2,4],[2,3]\rangle$, hatched) interval patterns. (**right**) Depth-first traversal of m_2 using minimal changes.

2 Problem Definition

Numerical dataset. A numerical dataset (G, M, T) is given by a set of objects G, a set of numerical attributes M and a numerical target label T. In a given dataset, the domain of any attribute $m \in M$ (resp. label T) is a finite ordered set denoted D_m (resp. D_T). Fig. 1 (left) provides a numerical dataset made of two attributes $M = \{m_1, m_2\}$ and a target label T. A subgroup p is defined by a pattern, i.e., its intent or description, and the set of objects from the dataset where it appears, i.e., its extent, denoted $ext(p)$. For instance, in Fig. 1, the domain of m_1 is $\{1, 2, 3, 4\}$ and the intent $\langle[2,4],[1,3]\rangle$ (see the definition of interval patterns later) denotes a subgroup whose extent is $\{g_3, g_4, g_5, g_6\}$.

Quality measure, optimal subgroup. The interestingness of a subgroup in a numerical dataset is measured by a numerical value. We consider here the quality measure based on the mean introduced in [14]. Let p be a subgroup. The quality of p is given by: $q^a_{mean}(p) = |ext(p)|^a \times (\mu_{ext(p)} - \mu_{ext(\emptyset)}), a \in [0,1]$. $|ext(p)|$ denotes the cardinality of $ext(p)$, $\mu_{ext(p)}$ is the mean of the target label in the extent of p, $\mu_{ext(\emptyset)}$ is the mean of the target label in the overall dataset, and a is a parameter that controls the number of objects of the subgroups. Let (G, M, T) be a numerical dataset, q a quality measure and P the set of all subgroups of (G, M, T). A subgroup $p \in P$ is said to be optimal iff $\forall p' \in P : q(p') \leq q(p)$.

Plant growth recipe and optimization measure. A plant growth recipe (M, P, T) is given by a set of numerical parameters M specifying the growing conditions thanks to intervals on numerical values, a numerical value P representing the number of stages of the growth cycle, and a numerical target label T to quantify the recipe quality. In a given recipe, each parameter of M is repeated P times s.t. we have $|M| \times P$ numerical attributes. Our goal is to optimize recipes and we want to discover actionable patterns in the sense that delivering such patterns will support the design of new growing conditions. An optimization measure f quantifies the quality of an iteration. We are interested in the mean of the target label of the objects of the optimal subgroup after each iteration.

The measure is given by $f_{mean} = \frac{\sum_{i \in ext(p)} T(i)}{|ext(p)|}$ where $T(i)$ is the value of the target label for object i.

3 Related Work

Designing recipes that optimize a given target attribute (e.g., the mass, the energy cost) is often tackled by domain experts who exploit the scientific literature. However, in our setting, it has two major drawbacks. First, most of the literature remains oriented towards conventional growing conditions and farming methods. In urban farms, there are more parameters that can be controlled. Secondly, the amount of knowledge about plants is unbalanced from one plant to another. Therefore, relying only on expert knowledge for plant recipe optimization is not sufficient. We have an optimization problem and the need for a limited number of iterations. Indeed, experimenting with plant growth recipes is time consuming (i.e., asking for weeks or months). Therefore, we have to minimize the number of experiments that are needed to optimize a given recipe. There are two main families of methods addressing the problem of optimizing a function over numerical variables: *direct* and *model-based* [18]. For *direct* methods, the common idea is to apply various strategies to sequentially evaluate solutions in the search space of recipes. However such methods do not address the problem of minimizing the number of experiments. For *model-based* methods, the idea is to build a model simulating the ground truth using available data and then to use it to guide the search process. For instance, [9] introduced a solution for recipe optimization using this type of method with the goal of optimizing the flavor of plants. Their framework is based on using a surrogate model, in this case a Symbolic Regression [13]. It considers recipe optimization by means of a promising virtuous circle. However, it suffers from several shortcomings: there is no guarantee on the quality of the generated models (i.e., they may not be able to model correctly the ground truth), the number of tested parameters is small (only 3), and the ratio between the number of objects and the number of parameters in the data needs to be at least ten for Symbolic Regression [10]. Clearly, it would restrict the search to only a few parameters.

Heuristic [15,2] and exhaustive [1,5] solutions have been proposed for subgroup discovery. Usually, these approaches consider a set of nominal attributes with a binary label. To work with numerical data, prior discretization of the attributes is then required (see, e.g., [3]) and it leads to loss of information and suboptimal results. A major issue with exhaustive pattern mining is the size of the search space. Fortunately, optimistic estimates can be used to prune the search space and provide tractability in practice [21,7]. [14] introduces a large panel of quality measures and corresponding optimistic estimates for an exhaustive subgroup mining given numerical target labels. They describe `SD-Map*`, the reference algorithm for subgroup discovery in numerical data. Notice however that for [14] or others [6,15], discretization techniques over the numerical attributes have to be performed. When looking for an exhaustive search of frequent patterns - not subgroups - in numerical data without discretization, we find the `MinIntChange` algorithm [11]. Using closure operators (see, e.g., [4]) has become a popular solution to reduce the size of the search space. We indeed exploit most of these ideas to design our optimal subgroup discovery algorithm.

4 Optimization with Subgroup Discovery

4.1 An Efficient Algorithm for Optimal Subgroup Discovery

Let us first introduce `MinIntChange4SD`, our branch and bound algorithm for the optimal subgroup discovery in purely numerical data. It exploits smart concepts about interval patterns from [11].

Interval patterns, extent and closure. In a numerical dataset (G, M, T), an interval pattern p is a vector of intervals $p = \langle [a_i, b_i] \rangle_{i \in \{1,\ldots,|M|\}}$ with $a_i, b_i \in D_{mi}$, where each interval is a restriction on an attribute of M, and $|M|$ is the number of attributes. Let $g \in G$ be an object. g is in the extent of an interval pattern $p = \langle [a_i, b_i] \rangle_{i \in \{1,\ldots,|M|\}}$ iff $\forall i \in \{1, \ldots, |M|\}, m_i(g) \in [a_i, b_i]$. Let p_1 and p_2 be two interval patterns. $p_1 \subseteq p_2$ means that p_2 encloses p_1, i.e., the hyper-rectangle of p_1 is included in that of p_2. It is said that p_1 is a specialization of p_2. Let p be an interval pattern and $ext(p)$ its extent. p is defined as *closed* if and only if it is the most restrictive pattern (i.e., the smallest hyper-rectangle) that contains $ext(p)$. Fig. 1 (center) depicts the dataset of Fig. 1 (left) in a cartesian plane as well as examples of interval patterns that are closed (p_2) or not (p_1).

Traversing the search space with minimal changes. To guarantee the optimal subgroup discovery, we proceed to the so-called minimal changes introduced in `MinIntChange`. It enables an exhaustive enumeration within the interval pattern search space. A left minimal change consists in replacing the left bound of an interval by the current value closest higher value in the domain of the corresponding attribute. Similarly, a right minimal change consists in replacing the right bound by the current value closest lower value. The search starts with the computation of the minimal interval pattern that covers all the objects of the dataset. The premise is to apply consecutive right or left minimal changes until obtaining an interval whose left and right bounds have the same value for each interval of the minimal interval pattern. In that case, the algorithm backtracks until it finds a pattern on which a minimal change can be applied. Fig. 1 (right) depicts the depth-first traversal of attribute m_2 from the dataset of Fig. 1 (left) using minimal changes.

Compressing and pruning the search space We leverage the concept of closure to significantly reduce the number of candidate interval patterns. After a minimal change and instead of evaluating the resulting interval pattern, we compute its corresponding closed interval pattern. We exploit advanced pruning techniques to reduce the size of the search space thanks to the use of a tight optimistic estimate. We also exploit a combination of *forward checking* and *branch* *reordering.* Given an interval pattern, the set of all its direct specializations (application of a right or left minimal change on each interval) are computed - forward checking - and those whose optimistic estimate is higher than the best subgroup quality are stored. Branch reordering by descending order of the optimistic estimate value is then carried out which enables to explore the most promising parts of the search space first. It also enables a more efficient pruning by raising the minimal quality early. In fact, providing details about the algorithm is out of the scope of this paper though its source code is available at `https://bit.ly/3bA87NE`. The important outcome is that it guarantees the discovery of optimal subgroups for a given quality measure. Indeed, provided that it remains tractable, the runtime efficiency is not here an issue given that we want to use the algorithm at some steps of quite slow vegetable growth processes.

4.2 Leveraging Subgroups to Optimize Recipes

A virtuous circle. Our optimization framework can be seen as a virtuous circle, where each new iteration uses information previously gathered to iteratively improve the targeted process. First, a set of recipe experiments - which can be created with or without the use of expert knowledge - is created. With the use of expert knowledge, values or domain of values are defined for each attribute and then recipes are produced using these values. When generating recipes without prior knowledge, we create recipes by randomly sampling the values of each attribute. Secondly, we use subgroup discovery to find the best subgroup of recipes according to the chosen quality measure (e.g., the subgroup of recipes with the best average yield). Then, we exploit the subgroup description - i.e., we apply new restrictions on the range of each parameter according to the description - to generate new, better, recipe experiments. Finally these recipes are in turn processed to find the best subgroup for the new recipes, and so on until recipes cannot be improved anymore. This way, we sample recipes in a space which gets smaller after each iteration and where the ratio between good and bad solutions gets larger and larger. Fig. 2 depicts a step-by-step example of the process behind the framework. Our framework makes use of several hyperparameters that affect runtime efficiency, the number of iterations and the quality of the results.

Convergence. The first hyperparameter is the parameter a used in the q^a_{mean} quality measure. In standard subgroup discovery, it controls the number of objects in the returned subgroups. A higher value of a means larger subgroups. For us, a larger subgroup means a larger search space to sample. By extension, a higher value of a means more iterations to be able to reach smaller subspaces of the search space. For that reason, we rename the parameter as the *convergence rate*. The second hyperparameter is called the *minimal improvement* ($minImp$). It defines the minimal improvement of the *Optimization measure* - f_{mean} in our setting - needed from one iteration to another for the framework to keep running. After each iteration, we check whether the following statement is true or false.

$$\frac{f_{mean_{it}} - f_{mean_{it-1}}}{f_{mean_{it-1}}} \geq minImp$$

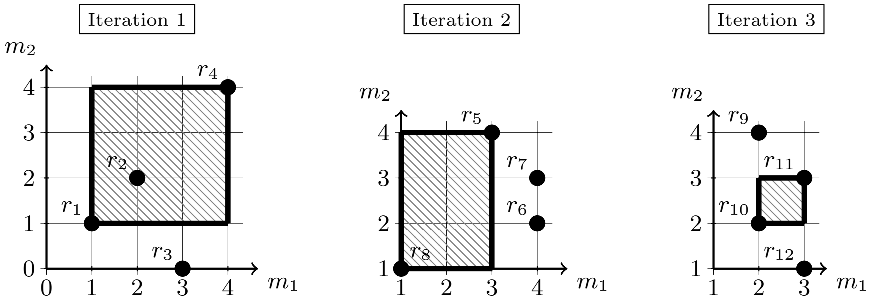

Fig. 2. Example of execution of the optimization framework in 3 iterations. We consider a two-dimensional space (i.e., 2 attributes m_1 and m_2) where 4 recipes are generated during each iteration using our first sampling method. The best subgroup (optimizing the yield) of each iteration (hatched) serves as the next iteration sampling space.

If it is true, then the optimization framework keeps running, else we consider that the recipes cannot be improved any further. This parameter has a direct effect on the number of iterations needed for the algorithm to converge. A higher value for *minImp* means a lower number of iterations and vice versa. We can also forget *minImp* and set the number of iterations by means of another parameter that would denote a budget.

Sampling the subspace. After each iteration, to generate new recipes to experiment with, we need to sample the subspace corresponding to the description of the best subgroup. Three sampling methods are currently available and this defines again a new hyperparameter. The first method consists in sampling recipes using the original set of values of each attribute (i.e., in the first iteration) minus the excluded values due to the new restrictions applied on the subspace. Let D_m^1 be the domain of values of attribute m at Iteration 1 and $[a_m^i, b_m^i]$ be the interval of attribute m at Iteration i according to the description of the best subgroup of Iteration $i-1$. Then, $\forall v \in D_m^1, v \in D_m^i \Leftrightarrow b_m^i \geq v \geq a_m^i$. Using this method, the number of values available for sampling for each attribute gets smaller after each iteration, meaning that each iteration is faster than the previous one. The second consists in discretizing the search space through the discretization of each attribute in k intervals of equal length. Parameter k is set before launching the framework. Recipes are then sampled using the discretized domain of values for each attribute. Finally, we can use *Latin Hypercube Sampling* [16] as a third method. In *Latin Hypercube Sampling*, each attribute is divided in S equally probable intervals, with S the number of samples (i.e., recipes). Using this method, recipes are sampled such that each recipe is the only one in each hyperspace that contains it. The number of samples generated for each iteration is also a hyperparameter of the framework.

An explainable generic framework. Our optimization framework is explainable contrary to black box optimization algorithms. Each step of the process is easily understandable due to the descriptive nature of subgroup discovery. Although we have been referring to our algorithm MinIntChange4SD when introducing the optimization framework, other subgroup discovery algorithms can be used, including [14] and [17]. Notice however that the better the quality of the provided subgroup, the better the results returned by our framework will be. Finally, our method can be applied to quite many application domains where we want to optimize a numerical target given collections of numerical features (e.g., hyperparameter optimization in machine learning).

5 Experiments

We work on urban farm recipe optimization while we do not have access to real farming data yet. One of our partners in the FUI DUF 4.0 project (2018-2021) is designing new types of urban farms. We found a way to support the empirical study of our recipe optimizing framework thanks to inexpensive experiments enabled by a simulator. In an urban farm, plants grow in a controlled environment. In the absence of failure, recipe instructions are followed and we can investigate the optimization of the plant yield at the end of the growth cycle. We simulate recipe experiments by using the PCSE[2] simulation environment by setting the characteristics (e.g., the climate) of the different growth stages. We focus on 3 variables that set the amount of solar irradiation (range [0,25000]), wind (range [0,30]) and rain (range [0,40]). The plant growth is split into 3 stages of equal length such that we finally get 9 attributes. In real life, we can control most of the parameters of an urban farm (e.g., providing more or less light) and a recipe optimization iteration needs for new insights about the promising parameter values. This is what we can emulate using the crop simulator: given the description of the optimal subgroup, we get insights to support the design of the next simulations, say experiments, as if we were controlling the growth environment. At the end of the growth cycle, we retrieve the total mass of plants harvested using a given recipe. Note that in the following experiments, unless stated otherwise, no assumption is made on the values of parameters (i.e., no restriction is applied on the range of values defined above and expert knowledge is not taken into account). Table 1 features examples of plant growth recipes. The source code and datasets used in our evaluation are available at `https://bit.ly/3bA87NE`.

5.1 MinIntChange4SD vs SD-Map⋆

We study the description of the best subgroup returned by MinIntChange4SD and SD-Map⋆, the state-of-the art algorithm for subgroup discovery in numerical data. Table 2 depicts the descriptions for a dataset comprised of 30 recipes generated randomly with the simulator. Besides the higher quality of the subgroup returned by MinIntChange4SD, the optimal subgroup description also enables to extract information that is missing from the description obtained

[2] `https://pcse.readthedocs.io/en/stable/index.html`

Table 1. Examples of growth recipes split in 3 stages (P1, P2, P3), 3 attributes, and a target label (Yield).

R	Rain^{P1}	Irrad^{P1}	Wind^{P1}	Rain^{P2}	Irrad^{P2}	Wind^{P2}	Rain^{P3}	Irrad^{P3}	Wind^{P3}	Yield
r_1	10	23250	5	10	23250	5	15	21000	10	22000
r_2	35	10000	14	5	25000	10	16	19500	30	20500
r_3	15	17500	26	22	15000	18	30	4000	3	8600
r_4	18	22800	17	38	17000	25	38	12000	19	14200

Table 2. Comparison between descriptions of the overall dataset (DS), the optimal subgroup returned by MinIntChange4SD (MIC4SD), the optimal subgroup returned by SD-Map⋆. ”-” means no restriction on the attribute compared to DS, Q and S respectively the quality and size of the subgroup.

Subgroup	Rain^{P1}	Irrad^{P1}	Wind^{P1}	Rain^{P2}	Irrad^{P2}	Wind^{P2}	Rain^{P3}	Irrad^{P3}	Wind^{P3}	Q	S
DS	[0,39]	[1170,23471]	[2,29]	[0,37]	[111,24111]	[0,29]	[2,40]	[964,24197]	[1,30]	0	30
MIC4SD	[16,37]	[1170,22085]	[2,24]	[7,37]	[18309,23584]	[2,24]	[15,37]	[12626,24197]	[1,25]	33874	7
SD-Map*	[21,39]	-	-	-	[14455,24111]	-	-	[12760,24197]	-	30662	5

with SD-Map⋆. In fact, where SD-Map⋆ only offers a strong restriction on 3 attributes, MinIntChange4SD provides actionable information on all the considered attributes, i.e., the 9 attributes. This confirms its qualitative superiority over SD-Map⋆ which has to proceed to attribute discretizations.

5.2 Empirical Evaluation of the Model Hyperparameters

Our optimization framework involves several hyperparameters whose values need to be studied to define proper ranges or values that will lead to optimized results with a minimized number of recipe experiments. We choose to apply a random search on discretized hyperparameters. Note that in this setting, grid search is a bad solution due to the combinatorial number of hyperparameter values and the high time cost of the optimization process itself. We discretize each hyperparameter in several values (the convergence rate is split into 10 values ranging from 0.1 to 1, the minimal improvement parameter is split into 12 values between 0 and 0.05, the sampling parameter is split between the 3 available methods, and the number of recipes for each iteration is either 20 or 30). We run 100 iterations of random search, with each iteration - read set of parameter values - being tested 10 times and averaged to account for randomness of the recipes generated. After each iteration of random search, we store the set of hyperparameter values and the corresponding best recipe found. Fig. 3 depicts results of the experiments. Optimal values for convergence rate seem to be around 0.5, between 0.001 and 0.01 for minimal improvement, and the best sampling method is tied between the first and second one. Generating 30 recipes for each iteration yields better results than 20 (average yield of 23857 for 30 recipes against 22829 for 20 recipes). To compare our method against other methods, we run our framework with the following parameters: 30 recipes times 5 iterations (for a total of 150 recipes), 0.5 convergence rate, using the second sampling

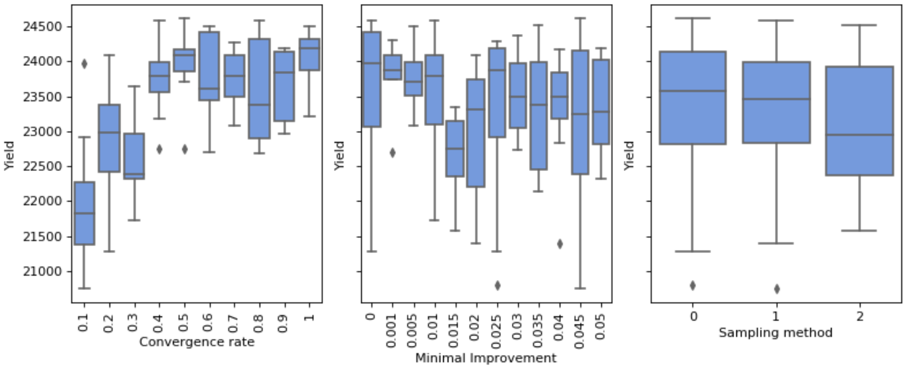

Fig. 3. Yield of the best recipe depending on the value of different hyperparameters using 100 sample recipes for each hyperparameter.

method with $k = 15$. To address the variance in the yield due to randomness in the recipe generation process, we run the framework 10 times, we store the best recipe found at each iteration and then compute the average of the stored recipes. We report the results in Table 3.

5.3 Comparison with Alternative Methods

Good hyperparameter values have been defined for our optimization framework and we can now compare our method with other ones. Let us consider the use of expert knowledge and random search. First, we want to create a model using expert knowledge. With the help of an agricultural engineer, we defined a priori good values for each parameter using expert knowledge and we generated a recipe that can serve as a baseline for our experiments. We then choose to compare our method against a random search model without expert knowledge. We set the number of recipes to 150 for all methods to provide a fair comparison with our own model where the number of recipes is set to 150. To account for randomness in the recipe generation, we run 10 iterations of the random search model, we store the value of the best recipe found in each iteration, and we compute their average yield. Results of the experiments and a description of the best recipe for each method are available in Table 3. Random search and expert knowledge find recipes with almost equal yields, while our framework find recipes with higher yield. Note that in industrial settings, an improved yield of 3% to 4% has a significant impact on revenues.

Let us now compare our framework to the Surrogate Modeling method presented in [9]. To be fair, we give the same number of data points to build the Symbolic Regression surrogate model as we used in previous experiments, i.e., 150 for training the model (we evaluated the RMSE of the model on a test set of 38 other samples). We use gplearn[19], with default parameters, except for the number of generations and the number of models evaluated for each generations, which are respectively of 1000 and 2000, as in [9]. Note that the model obtained

Table 3. Comparison of the description and the yield of the best recipe returned by each method. EK = Expert Knowledge, RS = Random Search, SM = Surrogate Modeling, VC = Virtuous Circle (our framework).

Method	Rain^{P1}	Irrad^{P1}	Wind^{P1}	Rain^{P2}	Irrad^{P2}	Wind^{P2}	Rain^{P3}	Irrad^{P3}	Wind^{P3}	Yield
EK	10	0	5	10	25000	5	10	25000	5	23472
RS	17	23447	8	31	22222	23	39	22385	7	23561
SM	20	44	0	20	24981	0	40	31	30	10170
VC	19	16121	18	25	24052	28	14	21126	7	24336

has a RMSE of 2112, and it is composed of more than 2000 terms (including mathematical operators), therefore the argument of interpretability is questionable. A grid search is finally done on this model and we select the best recipe and obtain their true yield using the PCSE simulation environment. The number of steps for each attribute for the grid search has to be defined. We set it to 5. As we have 9 parameters, it means that the model needs to be evaluated on nearly 9 million potential recipes. Also, the model is composed of hundreds of terms such that experiments are computationally expensive. The best recipe found so far is given in Table 3. The surrogate model predicts a yield value of 21137. Compared to the ground truth of 10170, the model has a strong bias. It illustrates that using a surrogate model for this kind of problem will give good recipes only if it is reliable enough. Interestingly, the RMSE seems to be quite good at first glance, but this does not guarantee that the model will behave correctly on all elements of the search space: on the best recipe found, it largely overestimates the yield, leading to a non-interesting recipe. It seems that this method performs poorly on recipes with more attributes than in [9]. Further studies are here needed.

6 Conclusion

We investigated the optimization of plant growth recipes in controlled environments, a key process in connected urban farms. We motivated the reasons why existing methods fall short of real life constraints, including the necessity to minimize the number of experiments needed to provide good results. We detailed a new optimization framework that leverages subgroup discovery to iteratively find better growth recipes through the use of a virtuous circle. We also introduced an efficient algorithm for the optimal subgroup discovery in purely numerical datasets. It has been recently improved much further in [17]. We avoid discretization and it provides a qualitative added-value (i.e., more interesting optimal subgroups). Future work includes extending our framework to deal with multiple target labels at the same time (e.g., optimizing the yield while keeping the energy cost as low as possible).

Acknowledgment. Our research is partially funded by the French FUI programme (project DUF 4.0, 2018-2021).

References

1. Atzmueller, M., Puppe, F.: SD-map – a fast algorithm for exhaustive subgroup discovery. In: Proceedings ECML/PKDD. pp. 6–17 (2006)
2. Bosc, G., Boulicaut, J.F., Raïssi, C., Kaytoue, M.: Anytime discovery of a diverse set of patterns with monte carlo tree search. Data Min. Knowl. Discov. **32**, 604–650 (2018)
3. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings IJCAI. pp. 1022–1029 (1993)
4. Garriga, G.C., Kralj, P., Lavrač, N.: Closed sets for labeled data. J. Mach. Learn. Res. **9**, 559–580 (2008)
5. Grosskreutz, H., Paurat, D.: Fast and memory-efficient discovery of the top-k relevant subgroups in a reduced candidate space. In: Proceedings ECML/PKDD. pp. 533–548 (2011)
6. Grosskreutz, H., Rüping, S.: On subgroup discovery in numerical domains. Data Min. Knowl. Discov. **19**(2), 210–226 (2009)
7. Grosskreutz, H., Rüping, S., Wrobel, S.: Tight optimistic estimates for fast subgroup discovery. In: Proceedings ECML/PKDD. pp. 440–456 (2008)
8. Harper, C., Siller, M.: Openag: A globally distributed network of food computing. IEEE Pervasive Computing **14**, 24–27 (2015)
9. Johnson, A., Meyerson, E., Parra, J., Savas, T., Miikkulainen, R., Harper, C.: Flavor-cyber-agriculture: Optimization of plant metabolites in an open-source control environment through surrogate modeling. PLOS ONE **14**, e0213918 (2019)
10. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global Optimization **13**(4), 455–492 (1998)
11. Kaytoue, M., Kuznetsov, S.O., Napoli, A.: Revisiting numerical pattern mining with formal concept analysis. In: Proceedings IJCAI. pp. 1342–1347 (2011)
12. Klösgen, W.: Explora: A multipattern and multistrategy discovery assistant. In: Advances in Knowledge Discovery and Data Mining, pp. 249–271 (1996)
13. Koza, J.R.: Genetic programming: On the programming of computers by means of natural selection. pp. 162–169. MIT Press, Cambridge, MA, USA (1992)
14. Lemmerich, F., Atzmueller, M., Puppe, F.: Fast exhaustive subgroup discovery with numerical target concepts. Data Min. Knowl. Discov. **30**(3), 711–762 (2016)
15. Mampaey, M., Nijssen, S., Feelders, A., Knobbe, A.: Efficient algorithms for finding richer subgroup descriptions in numeric and nominal data. In: Proceedings ICDM. pp. 499–508 (2012)
16. McKay, M.D., Beckman, R.J., Conover, W.J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics **21**(2), 239–245 (1979)
17. Millot, A., Cazabet, R., Boulicaut, J.F.: Optimal subgroup discovery in purely numerical data. In: Proceedings PaKDD (2020), 12 pages, In Press.
18. Rios, L.M., Sahinidis, N.V.: Derivative-free optimization: a review of algorithms and comparison of software implementations. Journal of Global Optimization **56**(3), 1247–1293 (2013)
19. Stephens, T.: gplearn. `https://github.com/trevorstephens/gplearn` (2013)
20. Wojciechowska, R., Długosz-Grochowska, O., Kołton, A., Żupnik, M.: Effects of led supplemental lighting on yield and some quality parameters of lamb's lettuce grown in two winter cycles. Scientia Horticulturae **187**, 80 – 86 (2015)
21. Wrobel, S.: An algorithm for multi-relational discovery of subgroups. In: Proceedings PKDD. pp. 78–87 (1997)