Towards Better Modeling of Supermarkets

K. Buza*, A. Buza** and P.B. Kis**
* University of Hildesheim, Hildesheim, Germany
** College of Dunaujvaros, Dunaujvaros, Hungary
buza@ismll.de, buza@mail.duf.hu, piros@mail.duf.hu

***Abstract* — In this paper we introduce the supermarket setout problem which aims at finding an appropriate setout of products (i.e. which product should be placed on which position at which shelf) w.r.t. the expected profit, access time (time required to find a product), etc. Finding a good setout of products is important both in the customer-area and in the background store of supermarkets. The admissible setouts are constrained by laws (like food is not allowed to be placed next to chemicals) and "setout traditions" (even if two products are allowed to be placed next to each other by law, it may look "strange"). After taking all such requirements into account, there are usually still a great number of possible setouts that are generally greatly different from various aspects, like expected profit, customer satisfaction, or the time required to find a product. Therefore, finding an appropriate (closely optimal) solution is a crucial issue. Complex business problems, like the above supermarket setout problem, are often solved by means of artificial intelligence techniques that exploit results of advanced statistical analysis or data mining. In this paper, we develop a new algorithm for the supermarket setout problem. This is based on a combination of constraint satisfaction algorithms and frequent itemset mining (also known as market basket analysis).**

I. INTRODUCTION

Complex business problems are often solved by means of artificial intelligence techniques that exploit results of advanced statistical analysis or data mining. Market basket analysis is one of the most prominent fields of data mining. It aims at finding pairs, triples... of products which are often bought together. This information can effectively be used in business: based on such an analysis, one should determine, for example, which products should be placed next to each other on the shelves of the supermarket.

The corresponding algorithmic problem, the so called frequent itemset mining problem was defined by Agrawal and Shrikant [1]. This aims at fining those sets of products which are often bought together. Such a set is called a frequent itemset. As real-life databases are very large, and the number of frequent itemsets is potentially exponential in the number of products of the supermarket (note, that the number of products in a supermarket can be about 100000 or even more in the practise) thus, in the last decade, researchers mainly focused on runtime, i.e. algorithmic optimisations. Just to mention a few of the most important works in this field, we refer to [1, 2, 3, 4].

The problem has been generalized for different pattern types. In the original formulation, one was only interested for sets of products (called itemsets). However, in which order customers typically purchase products is potentially very important information: rules like *if product A is purchased on day t and product B is purchased on day t+1, then on day t+5 the products C and D are likely to be purchased* may be very interesting. This leads to the *frequent sequence* mining problem [5, 6, 7]. Similarly, if a taxonomy of products is given, one can use the product categories from the taxonomy in frequent itemsets [8, 9, 10, 11, 12].

In the last decade, the main stream research in this area focused on algorithmic optimisation of runtime for different pattern types (itemsets, sequences of items, itemsets with taxonomy,...). On the other hand, there are many works on applying frequent itemsets for various practical tasks as well, e.g. prediction or knowledge organisation (ontology learning). For prediction problems, the data is usually embedded in the frequent itemset space first, then a prediction (which is relevant for the given practical task) is made by some statistical classifier (SVM, decision tree, neural network,...). In other words: the frequent itemsets (their presence or absence in each particular data object) are used as input features for statistical classifiers in order to predict some unknown properties of objects. This way, patterns have been used, for example, for time series classification [13, 14, 15]. In an other algorithm, frequent itemset mining have been used for building knowledge structures integrating expert knowledge and user vocabulary [16].

In this paper, we turn back to the classic supermarket scenario. We point out, that frequent itemsets alone are not sufficient to solve the original business problem (i.e. *Which products should be placed next to each other on the shelves?*, or more generally: *What is the optimal setout of products?*). The reason for this is the presence of regularisations (laws) and traditions for the setout: even if, based on frequent itemsets alone, two products should be placed next to each other, this is not always allowed (for example there are laws, which products are not allowed to be next to food) or this may look "very strange" (it does not fit our "setout traditions").

However, frequent itemsets are still important, and they have the potential to improve the setout: taking both the traditions (including regularisations), and the frequent itemsets into account, one can improve the setout.

Besides the setout of products in the customer-area of the supermarket (that we have already mentioned), prominent practical applications of the supermarket setout problem include the setout of products in the background store of the supermarket (that is usually hidden to customers). Both in the customer-area of the supermarket and in the background store, similar constraints have to be taken into account (e.g. chemicals are not allowed to be placed next to food).

In this paper we (i) introduce the supermarket setout problem, (ii) combine frequent itemset mining and constraint logic programming in order to find a "good" (not necessary optimal) setout, and (iii) show that the

setout based on both frequent itemsets (customer habits) and regularisational-traditional setout rules is better, than the setout based solely on the regularisational-traditional setout rules.

The paper is organized as follows: in Section 2 we introduce the supermarket setout problem. In Section 3, we introduce an algorithm based on the combination of frequent itemset mining and constraint logic programming, which outputs a setout. This setout is evaluated in Section 4. Finally, we conclude in Section 5.

II. The Supermarket Setout Problem and Store Setout Problem

There is a great number of the opportunities for the setout of products inside a supermarket (both in the customer-area and in the background store).

In one respect, the setouts of shelves (i.e. where shelves are placed inside the shop) may be very various; furthermore the setouts of the goods on the shelves (i.e. at which position a product is placed on a given shelf) may also be very diverse.

In general, the goal of a supermarket is the maximization of the profit. As the income originates from the customers, the aim is that customers buy as much as possible products. From our analysis point of view, the following simplified schema characterise customers behaviour at the supermarket: on the one hand, they will certainly look for products on their shopping lists; on the other hand, while walking inside the supermarket they might buy some additional products, before they reach at the exit or to cash desk.

While searching for the products that a customer is originally intended to buy (i.e. products on the shopping list), customers usually walk through the shop on some partially irregular route. If the customer notices opportunity favouring for her/him, then she/he is probably buying something, which is not included in the original shopping list. What she/he will notice and buy, depends strongly on the visible and/or audible advertisements, advantages, discounts, etc., and on the goods to be sold themselves, of course. Marketing studies show: the longer the customer is walking in the supermarket, the more money she/he spends. We can powerfully influence the route of the customer in the shop by the setout of products.

Obviously, the customer starts her/his route at the entrance and finishes at the exit, i.e. cash desk. (From our point of view the exit and cash desk are equivalent, as we assume that the customer goes from the cash desk to the exit directly, i.e. she/he does not buy anything on the way from the cash desk to the exit.) For this reason accessibility of the cash desk is typically manipulated by the setout of the shelves: the customer is forced to take a relatively long way in order to arrive at the cash desk.

The main goal of the customer is to buy the products on her/his shopping list. The position of these products (i.e. where these products are to be found in the supermarket) has major influence on the route of the customer. By analysis of the market baskets we can determine the groups of products which are typically bought together, which can be regarded as an approximation of typical shopping lists. If we put such products (i.e. products typically bought together) far from each other, we "force" the customer to take a longer route in the supermarket.

However, this point of view is not the only one. There are well defined written restrictions, regulations, laws concerning fire protection, healthcare, security, consumer protection, and so on. When determining the setout of products, these have to be taken into account as well. Furthermore, customers prefer the suitable – from his/her point of view – setout of the products: customers often do not tolerate extremely long routes as well as the scattering of the similar goods. If the customer is forced to take long routes, she/he will not enter this shop next time. Therefore instead of increasing the profit, the profit will decrease.

In the background store of a supermarket (that is usually hidden to customers), similar setout rules (regulation laws, etc.) apply. We argue that finding an appropriate setout of the background store is, in fact, an equivalent problem. After customers have bought many products, the staffs go the background store and bring products from the background store into the customer area in order to refill shelves. In the background store, the staff of the supermarket is a "meta-customer": from the background store point of view, the staff acts as a customer. Here, the aim is to minimise the time and the length of the route that is necessary to find a product. Although the staff is conscious (and thus perform random walk only to a limited extent) when searching for those particular products that primarily need refillment, but if "additional products" (that also need refillment, but this refillment is not urgent) are also found on the way to the primarily searched products, the staff may also bring these additional products into the customer-area of the supermarket.

At the conceptual level, the supermarket setout problem is an optimization task; the objective function may be profit, time, etc. In practical cases, the exact formula for this objective function is, however, unknown: customer behaviour is not known exactly, thus needs for refillment of shelves in the future can only be estimated, but not exactly calculated. As the exact formula of the objective function is unknown, we can not apply classic optimisation techniques. Therefore, we introduce an approach that is based on constraints on the one hand, and frequent itemset mining on the other hand.

III. Frequent Itemsets as Setout Constrains

Our approach to solve the supermarket setout problem is based on constraint logic programming. Our "setout rules" (both the ones based on regularisations, traditions, and the ones based one frequent itemsets) are modelled by constraints.

Let different products be denoted by different integer numbers 1, 2, 3, . . . , N, where N is the count of different products in the supermarket. We model each possible position, where a product can be placed at a shelf, as a variable. Let $V_{i,j}$ denote the i-th position of the j-th shelf.

Fig. 1. shows the pseudocode of our approach. In the followings, we describe each step in more details:

1. Regularisations (laws) concerning setout and "setout traditions" can be modelled by constraints. For example, if product A and B are not allowed to be next to each other, this corresponds to the following constraint:

$$V_{i,j} = \mathrm{A} \rightarrow (V_{i-1,j} \neq \mathrm{B} \wedge V_{i+1,j} \neq \mathrm{B}).$$

If two products *A* and *B* must be on the same shelf, one could write the following constraint for that:

$$V_{?,j}=\mathrm{A} \rightarrow V_{?,j}=\mathrm{B},$$

where ? is a "joker", which means an arbitrary position of a shelf.

2. The analysis of the product access data (that refers to product purchase or transportation from the background store into the customer-area) results in frequent itemsets, i.e. after the analysis we know, which products are typically accessed together.
3. Frequent itemsets found in the previous step, can be turned into additional constraints. In case of the setout of the customer-area, the following procedure can be used. Suppose, we know that products *A* and *B* are often bought together, and we may want the customer to traverse through the whole supermarket in order to spend more time in the supermarket (we hope that the customer will buy some other products during her/his tour), thus we should place products *A* and *B* far from each other. Please note, that marketing studies show: the more time a customer spends in the supermarket the more she/he purchases. Using constraints, one can say, for example, that products *A* and *B* must not be placed on the same shelf.

   In case of searching for the appropriate background store setout, products that are accessed together should be placed close to each other, instead of placing them far from each other. Thus for the background store setout case, constraints are derived in a different way from than in the customer-area setout case.
4. Finally, one has to solve the constraint satisfaction problem. This gives the setout of the customer-area or the background store. (This setout is not necessary the best one, however, as we will see in the next section, this is a relatively good setout.)

Algorithm 1 Solver for the Supermarket Setout Problem

Require: Number *min_support*,
Transactional Database $\mathcal{D}$,
Laws concerning the placement of products $\mathcal{L}$
Ensure: $\mathcal{S}$ = Solution of the Supermarket Setout Problem

1: $\mathcal{C}_0$ = constraints derived based on laws $\mathcal{L}$
2: I = mine_frequent_itemsets($\mathcal{D}$, *min_support*)
3: $\mathcal{C}_I$ = derive_constraints(I)
4: $\mathcal{S}$ = solve_csp($\mathcal{C}_0 \cup \mathcal{C}_I$)
5: **return** $\mathcal{S}$

Fig. 1. The pseudocode of our approach.

Before implementing this high-level approach, we note that constraint satisfaction problems are not always solvable. In order to be more precise, the number of solutions may be (i) zero, (ii) one or (iii) more than one. Usually, the more constraints have to be satisfied, the less is the number of solutions. We assume, that the constraint satisfaction problem consisting only of the constraints based on regularisations and traditions (i.e. only of the constraints formulated in the first step) has at least one solution. If this would not be the case, one could not find any suitable setout. In the 3rd step, we derive constraints from the frequent itemsets. We begin this procedure with the most frequent itemset, and we follow with the 2nd, 3rd, … frequent itemset, as long as the resulting constraint satisfaction problem has at least one solution. (This means, that less frequent itemsets may eventually not be taken into account, if this would lead to an unsolvable constraint satisfaction problem.)

We also note, that we assume that the constraint solver applied in the 4th step, is able to handle all the constraints introduced in the first and 3rd steps. This is a nontrivial requirement concerning the constraint solver; however the discussion or enhancement of facilities of constraint solvers is out of the scope of this paper. We assume, that in steps 1 and 3 we formulate such constraints, which can be handled by the constraint solver applied in the last step. This means also, that we assume, that the class of constraints, which can be handled by the constraint solver is rich enough so that the requirements concerning a "good" setout can be formulated.

IV. Experiments

We evaluated our approach in context of the setout of the customer-area of the supermarket.

In our experiments, we use generated data. Thus our experiments are limited, especially quantitative: our aim is to demonstrate the usability of our approach only. Our "simulated" supermarket consists of 2×3 shelves (there are two row of shelves, in each row there are 3 shelves). Products can be placed on each sides of the shelf, 10 products per side. Thus we have 120 products in all.

To generate realistic product purchase data (so called transactional database), first we randomly generated a joint probability distribution of product purchases. This is denoted by *p*. Then we generate the product purchase data according to *p*. This data consists of 1 million transactions (i.e. 1 million baskets of customers). To find the frequent itemsets efficiently in this data, we used a highly efficient variant of the algorithm Apriori based on trie representations and doubly recursive counting scheme [2,3,4].

In our experiments we used only one class of constraints. This was, that products *A* and *B* are not allowed to be on the same shelf: *diffShelf*(*A*,*B*).

If the approach is applied in practice, one would need much more constraint types, like *sameShelf*, *nextPositionOnShelf*, … . For our experiments we assumed, that one can formulate all the regularizations (laws) concerning setout and "setout traditions" as well as the constraints derived from frequent itemsets in form of such *diffShelf* constraints. The constraints concerning regularizations (laws) and "setout traditions" were randomly generated. The constraints based on frequent itemsets were derived as described in the previous section.

As constraint solver we applied classic backtracking algorithm [17] with fixed variable order and fixed value order. (In our experiments we focused on the

demonstration of the usability of our approach. Although there exists many better constraint solvers, in our case this simple algorithm was sufficient.) The variable order was $V_{1,1}$, $V_{2,1}$, $V_{3,1}$, … , $V_{1,2}$, $V_{2,2}$, $V_{3,2}$, … , i.e. first a product will be placed on the first position of the first shelf, then an appropriate product (with respect to the constraints) will be placed on the second position of the first shelf, etc. If the first shelf is full of products, then the next product will be placed on the second shelf, etc. The values of each variable correspond to products. Products were numbered from 1 to 120, concerning value ordering in the backtracking algorithm, we applied the natural ordering 1, 2, 3, … ,120.

Whenever the constraint satisfaction problem (CSP) had more than one solution we always considered the first solution only. Of course, this is not an "intelligent" strategy of selecting the best solution. However, we argue, that any heuristics, which could be applied at this stage, would affect the solution of the supermarket setout problem (one would select the solution according to that heuristic, and in this sort of sense that heuristic would solve the problem); but our aim is not to measure how well a specific heuristic solves this problem, but to measure as clear as possible the effect of taking frequent itemsets into account. To be more concrete: of course, when (first case) taking only regularisational-traditional constraints into account, the set of solutions of the CSP is a superset of those solutions which one would get if (second case) taking both regularisational-traditional constraints and constraints based on frequent itemset into account. Thus, the best solution in the second case is of course among the solutions of the first case. However, how to select in each case the best solution among the possibly high number of solutions, is non-trivial. Our actual aim is to compare a "typical" solution in the first case against "typical" solutions in the second case. We argue that the first solution returned by the constraint solver is a representative in each case, and thus we limit our investigations to those solutions of the CSP. Furthermore, constraint solvers are often able to find a solution quickly, however finding all solutions may take lot of time. Thus in a practice, it would be reasonable to decide for the first solution, unless there are strong arguments against it.

TABLE I. EXPERIMENTS

Setout Constraints	Avg. number of products purchased
only regularisational-traditional constraints	15.5
both regularisational-traditional constraints and on constraints derived from frequent itemsets	17.3

Based on the joint probability distribution of products p, we generated 10 shopping lists of length 12±2. We considered customers, who walk randomly in the shop. Each customer has one of the generated shopping lists, and if the customer is currently next to a product which is contained on his/her shopping list, he/she purchases this product. However, during the random walk, the customer purchases some other products with a small probability (0.01) as well. On the other hand, customers do not like walking for a long time, thus they leave the shop, if they already have 90% of the products on their shopping lists.

In this framework, we compared two setouts: (i) the setout based only on regularisational-traditional constraints, and (ii) the setout based on both regularisational-traditional constraints and on constraints derived from frequent itemsets. When the data was searched for frequent itemsets, we used a minimum support threshold of 0.008 (i.e if a set of products, called itemset is contained in at least 0,8% of all the baskets, it was considered as frequent). In our experiments, on average over the 10 shopping lists, we found that customers buy around 2 more products in the second case.

Note, that our experiments are limited quantitative, because we used generated data. Thus this result only demonstrates the usability of the approach, but this result does not mean that real customers would buy 2 more products.

V. CONCLUSIONS

In this paper we introduced the supermarket setout problem. We combined frequent itemset mining and constraint logic programming in order to find an appropriate solution for this problem. This is contribution in the modelling of a complex business problem.

As future work, one could evaluate this approach in other settings, especially with real market basket data and real regularisational-traditional constraints and/or improved customer behaviour model. In this work, we assumed, that the customer walks randomly through the shop, and purchases some products randomly, this can be improved for example by introducing customer preferences, e.g. some customers may have a negative preference for congestion.

REFERENCES

[1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", *20th Int'l Conf. on Very Large Data Bases* (1994).

[2] F. Bodon, "A trie-based APRIORI implementation for mining frequent item sequences", *1st Int'l Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, Chicago, Illinois (2005).

[3] C. Borgelt, "Efficient Implementations of Apriori and Eclat", *Workshop of Frequent Item Set Mining Implementations*, Melbourne, FL, USA (2003).

[4] C. Borgelt, "Recursion Pruning for the Apriori Algorithm", *2nd Workshop of Frequent Item Set Mining Implementations*, Brighton, UK (2004).

[5] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements", *EDBT*, Avignon, France (1996).

[6] J. Pei, J. Han, J. Wang, H. Pinto, Q. Chen, U. Dayal, M. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", *IEEE Trans. on Knowledge and Data Engineering* (2004) **16:1424–1440.**

[7] K.L. Jensen, M.P. Styczynski, I. Rigoutsos, G.N. Stephanopoulos, "A generic motif discovery algorithm for sequential data", *Bioinformatics* (2006), **22:21–28.**

[8] J. Hipp, A. Myka, R. Wirth, U. Güntzer, "A new algorithm for faster mining of generalized association rules", *European Conf. on Principles of Data Mining and Knowledge Discovery*, Nantes, France (1998).

[9] I. Pramudiono, M. Kitsuregawa, FP-tax: "Tree structure based generalized association rule mining", *ACM/SIGMOD Int'l W'shop on Research Issues on Data Mining and Knowledge Discovery*, Paris (2004).

[10] K. Sriphaew, T. Theeramunkong, "A new method for finding generalized frequent itemsets in generalizes association rule

mining", *Int'l Symp. on Computers and Communications*, Taormina, Italy (2002).

[11] K. Sriphaew, T. Theeramunkong, "Fast algorithms for mining generalized frequent patterns of generalized association rules", *IEICE Transactions on Information and Systems* (2004) **E87-D(3).**

[12] W. Gaul, L. Schmidt-Thieme, "Mining Generalized Association Rules for Sequential and Path Data". *IEEE Int'l Conf. on Data Mining*, San Jose (2001).

[13] T. Knorr, "Motif Discovery in Multivariate Time Series and Application to Hemodialysis Treatment Data", *MSc-Thesis, Albert-Ludwigs-Univ*., Freiburg (2006).

[14] T. Knorr, "Identifying Patients at Risk: Mining Dialysis Treatment Data*" 2nd German Japanese Symposium on Classification*, Berlin (2006).

[15] K. Buza, L. Schmidt-Thieme, "Motif-based Classification of Time Series with Bayesian Networks and SVMs", *Proceedings of the 32nd Annual Conference of the Gesellschaft für Klassifikation*, Springer (to appear).

[16] L. Balby Marinho, K. Buza, L. Schmidt-Thieme, "Folksonomy-based Collabulary Learning", *Proc. of the International Semantic Web Conf., LNCS 5318, Springer* (2008).

[17] S. Russel, P. Norvig, "Artificial Intelligence – A Modern Approach", *Prentice Hall* (2003).