Proceedings of the 19th CIRP Design Conference – Competitive Design, Cranfield University, 30-31 March 2009, pp388

Supporting Knitwear Design Using Case-Based Reasoning

P. Richards, A. Ekárt
School of Engineering and Applied Science, Aston University, Birmingham, B4 7ET, U.K.
{richardp, ekarta}@aston.ac.uk

Abstract
Knitwear design is a creative activity that is hard to automate using the computer. The production of the associated knitting pattern, however, is repetitive, time-consuming and error-prone, calling for automation. Our objectives are two-fold: to facilitate the design and to ease the burden of calculations and checks in pattern production. We conduct a feasibility study for applying case-based reasoning in knitwear design: we describe appropriate methods and show how they can be implemented.

Keywords:
computer-aided design (CAD), case based reasoning, knitwear design, similarity, adaptation

1 INTRODUCTION

The design and production of patterns for hand knitting is a very tedious process, involving a highly qualified team of a designer, a pattern writer, several checkers and knitters, a typesetter and proofreaders. There is very little documentation and training material available, most of the skills are acquired through learning by doing. The designers and other members of the team find it difficult to formulate the rules which decide how they perform various steps, an outsider would have to watch them, try and understand what they are doing and then learn by asking questions.

The designer first designs the pattern and produces a hand-drawn sketch on graph paper, which includes measurements. The pattern writer then manually calculates the pattern for hand knitting in different sizes. The checkers perform an initial check of the pattern and make any necessary corrections. The knitters then hand knit the pattern in one size and make alterations to the pattern if needed (for example to obtain the desired shape of the garment). The alterations are made on the pattern for six different sizes. The checkers manually check that all the calculations are correct. For a proper check, they sometimes hand knit a sample of the complicated parts of the pattern (a finishing, for example) to make sure that the result corresponds to the description. In the next step, the typesetter typesets the pattern using a standard word processor. Finally, at least three proofreaders check the typeset pattern for any mistakes (they perform a read-through check). If there are alterations needed, the typesetter will make them on the typeset pattern.

The process is very time consuming and requires a lot of human attention and effort. Everything in the process, except the typesetting, is done manually, without the use of computing technology. We demonstrate here how specialised tools can be developed to help the designer in producing the sketch and also to automate the checks, calculation and production of the written patterns.

Although knitting and knitwear design has a long history and also includes many calculations and steps that are almost mechanical, there have been surprisingly few attempts to use computers for improving the underlying processes. The design of machine-produced knitwear was investigated by Eckert et al. [1]. Communication difficulties arise in large design teams since the specifications of the garments are often inaccurate, incomplete and inconsistent. To alleviate this, the authors proposed that designers use an intelligent CAD system. Active critiquing, i.e. pointing out differences between a designer's goal and what is actually happening, can be used to catch errors as early as possible. In the user interface which was presented, choices were only offered if they were relevant, based on the answers to previous questions. Shapes were described using shape grammars and Bézier curves. Interestingly, they say that "case based reasoning could be employed to provide starting measurements for cutting pattern construction".

A system for automatically designing knitting stitch patterns for hand knitwear was presented by Ekárt [2]. It is possible to produce many designs that are not knittable, because they violate the rules of knitting. For example, if a width of a garment does not change then the number of

stitches in a row must be equal to that of the previous row. A method of representing knitwear based on trees is able to avoid many of these invalid designs. Some heuristic measures were described, which can identify aesthetically pleasing patterns. This avoids the fatigue which is involved in the alternative techniques of asking humans to evaluate patterns which have been generated.

Many studies on creative design conclude that human designers create new designs by studying past designs from various resources and combining them together, adding new elements. To assist the design and production of knitting patterns, we propose a case based reasoning system that allows for structured step-by-step production of new patterns from scratch or reuse and adaptation of similar patterns from the past.

2 CASE-BASED REASONING

Case-based reasoning (CBR) is a generic problem solving methodology based on the idea that a solution to a new problem can be obtained from solutions to similar problems encountered in the past [3]. In addition to using knowledge from previously experienced problems, CBR has an element of incremental learning: every new experience is recorded for future use.

CBR is based on typical human problem solving: an engineer designs a new product or part by possibly reusing features of a past design, whose specification presents a certain degree of similarity with the expected new product; a doctor examining a patient builds up the diagnosis based on a comparison of the symptoms with those described in a book or previously shown by another patient; a decision in court is taken by drawing the similarity to a case in the past. The key in all these situations is to remember the similar problem case from the past and adapt its solution to the new situation.

The main cycle of CBR consists of four steps [3]:

- **retrieve** the most similar case(s) in the case base;
- **reuse** these case(s) to attempt to solve the new problem;
- **revise** the proposed solution;
- **retain** the new problem and its solution in a new case in the case base for later use.

There are various methods to represent cases, in most situations the problem description and the solution are represented as attribute-value pairs. A suitable similarity function has to be devised to compare the new problem case with the ones stored in the case base. The best matching cases are retrieved from the case-base and adapted for reuse. Adaptation is often very simple, such as slightly changing values of selected attributes, or left for the human to do. In domains where there is a huge number of past cases available, it is very likely to find a very similar past case whose solution needs little adjustment to the new case. However, where there are few past cases available only, the adaptation needs to be more substantial and prepared to deal with larger differences between past cases and the new case. It is customary to allow the user to rate how well the new solution proposed by the CBR system performed and possibly repair it in the revise phase. The new case is then retained for future use in the case base if it is judged to be sufficiently different from the existing cases.

Lopez De Mantaras et al. [4] comprehensively discuss retrieval in CBR, e.g. whether to use surface features or data which is derived from a more in-depth analysis of the case. Some of the methods they discuss are variants on nearest neighbour algorithms, others rely on a complex representation of cases, e.g. in a hierarchy or as a graph. They mention systems where a subset of cases is retrieved and presented to the user; in some of these systems they sacrifice some of the similarity in order to introduce diversity. They also discuss “adaptation-guided retrieval”, which uses domain specific knowledge to reduce adaptation failures in situations where the most similar case is not necessarily the easiest to adapt.

Price and Pegler [5] describe the Wayland advisor for the setting up of die-casting machines. Wayland consists of a series of fields, each of which has a weight chosen to set the significance; these are summed to give an overall match value. Wayland is an uncommon example of the successful use of CBR in a difficult design problem. There is a large value space for the inputs, and the relationship between those inputs and the desired outputs is not clearly understood. Nevertheless the system was able to find a ‘close enough’ solution, and as a result was successfully deployed in foundries.

Kolodner [6] discusses indexing and flexible methods of organising cases into a network, e.g. a memory organisation packet. She gives four methods of computing similarity: using an abstraction hierarchy, a qualitative scale, a quantitative scale, or comparison of roles. The choice of representation and similarity measure depends on the nature of the data, e.g. whether it is hierarchical, discrete choices or continuous range.

Bergmann et. al [7] discuss case representation, pointing out that the choice of *case representation and similarity measure are related*. Straightforward representations are feature vectors, textual and object-oriented. Generalised cases are achieved by introducing variables into cases to represent solutions to a wide range of problems. Complex cases may involve a choice of variables and wide ranges for their values. The importance of generalised cases lies in the fact that they can enable a CBR system *to perform well with a small case base*.

Bergmann et al. also discuss CBR systems with a hierarchical representation. Larry Leifer of Stanford University famously said “all design is redesign”. This applies to knitwear since at the lowest level garments are composed of the same types of stitch. Therefore, a partonomic hierarchy is conceivably a good way of representing knitwear. However, comparing the detail of complex cases in order to judge their similarity is much more difficult than simply matching on surface features.

Craw [8] describes k-NN (nearest neighbour) algorithms and makes the point that recording how many times a particular case has been reused can, in future, help to identify problems with the coverage of the case base.
Mejasson et al. [9] use a weighted sum algorithm in their intelligent design assistant system, which assists designers of submarine cables. The algorithm involves the sum of the squares of individual distances; the weights applied to these distances can be set by the user. In order to compare values, their range was often mapped onto a qualitative scale, with the points on the scale treated as being of equal distance. They use an abstraction hierarchy to measure the distance between components.

Watson [10] underlines the fact that CBR is a methodology and implementers are free to use whichever technology is appropriate to implement, mentioning nearest neighbour, induction, fuzzy logic and SQL as examples.

The existing work shows the prevalence of k-NN algorithms using weighted sums and techniques to deal with symbolic, rather than numeric data. This approach is suitable for our domain, since adaptation of our hierarchical representation (see below) will mean we do not need to index our cases or organise them in a particularly complex way, leaving us to focus on the challenges of adaptation.

CBR is particularly suitable for the knitwear design problem. Designers themselves study many past patterns before they create a new design. A new design will always contain some new element or an interesting combination of elements used in previous patterns, but will be similar to previous designs. A new sweatshirt will most commonly contain the four ordinary parts: front, back, right and left sleeves. However, their shape and structure will be designed according to new trends and wool type. There could be many neck and armhole types, various different lengths and sleeve lengths, possible accessories (such as a belt) or interesting borders, and the stitch pattern can vary across the whole sweatshirt.

Sirdar Spinning Ltd. is producing about 300 new patterns every year, so the case base will contain a limited number of very specialised cases. A new case can be solved by combining and adapting several sufficiently similar cases (over different attributes) rather than adapting one single case, as it is unlikely to find a very similar past case across all dimensions. Once the high level representation is created for the new case, the details at the lowest level must be rigorously produced in order to obtain a full solution. Our system cannot solely rely on the case base as the new trends may involve a new element or shape that has not been encountered before. The human user will be offered the possibility to create a new design themselves with the help of the system and then store it in the case base.

3 CBR APPLIED TO KNITWEAR DESIGN

3.1 Case Representation

We represent our cases using three levels of detail:

1. *Questionnaire*: A high-level specification that is similar to a sheet currently used by knitwear designers.
2. *Sketch*: A visual illustration of the shape, consisting of points, lines or curves between them, and rules that govern them.
3. *Chart*: A detailed and complete description of the knitting stitch pattern that makes up the garment.

The three-level representation is consistent with the existing process, where detail develops as the design progresses. It facilitates case-based reasoning since we can use the questionnaire in the retrieval stage, so indexes are not required.

We store our cases as XML files, in a human-readable format. This affords portability, and facilitates testing, since no interpretation of the files is required.

3.2 Similarity Algorithm for Retrieval

We use a weighted sum algorithm to assess the similarity of two garments. The first step is the calculation of the raw similarity *raw_sim*; the higher the value of *raw_sim*, the more similar the garments are judged to be. Each feature in the questionnaire for the garment that we are creating is compared to the equivalent in the previously created garment, stored in the case base. If the features match, a weight is added to *raw_sim*. The values of the weights are determined by the user (see section 4.4).

In many cases, the values in the questionnaires are “yes/no” choices. However, in some scenarios there is a choice from a list, and options may differ in their similarity to each other. For example, a wrist-length sleeve is more similar to a 3/4 length sleeve than it is to a very short one. We define a 2 x 2 matrix for the attribute values and the user can assign a score to each pair of values. In these situations the the weighted score is added to *raw_sim*.

If the feature is not present in the garment then the weight is not calculated and other features which are dependent on this feature are ignored in the existing garment. For example, if we are creating a sleeveless garment then a weight ($W_{hassleeves}$) would be used for existing sleeveless garments; but not for sleeved ones. In the latter case, details such as the cuffs are irrelevant.

The normalised similarity (norm_sim) is given by:

$$norm\_sim(a,b) = \frac{raw\_sim(a,b) - min\_sim(a,b)}{max\_sim(a,b) - min\_sim(a,b)} \quad (1)$$

where:

$$min\_sim(a,b) = W_{armholestyle} \times min\_score_{armholestyle} + W_{neckshape} \times min\_score_{neckshape} + W_{wearer} \times min\_score_{wearer}$$

$$max\_sim(a,b) = raw\_sim(a,a)$$

$min\_score_{armholestyle}$, $min\_score_{neckshape}$ and min_score_{wearer} are the minimum values in the score matrix of the garment being created for the row corresponding to the armhole style, neck shape and wearer respectively. All these attributes are mandatory and independent of other data, so it is possible that some rows on this matrix will have values that cause the minimum similarity not to be zero. So, *norm_sim(a,b)* lies in the range [0,1], where 0 corresponds to completely different and 1 to identical. Finding the most similar garment to the new one is a maximisation problem. We argue that maximisation of similarity is more intuitive to the non-technical user than minimisation of distance.

Let us now examine the axioms of distance metrics, as discussed in Tversky [11]. Minimality applies to our algorithm; this is exploited in our description of the normalisation, as explained above. However, as in many of the situations Tversky describes, symmetry and the triangle inequality do not necessarily apply to our algorithm. According to Craw [9], roles are important where there are asymmetric similarity measures; here the roles are ‘garment being created’ and ‘previously created garment’. Symmetry and triangle inequality axioms need not apply in our scenario, since what is really relevant is the adaptation distance. For example, if the sleeves are removed from garment A to produce garment B, then B is quite similar to A but not vice versa since it is easier to remove a sleeve than re-design it.

The obvious retrieval strategy is to attempt to adapt the existing garment(s) with the highest similarity. If this is unsuccessful then the next most similar garment can be attempted, and so on. However, we may consider introducing diversity, as discussed in Aamodt et al. [5], since if the most similar case cannot be adapted then attempting adaptation on cases that are very similar to this may also be futile.

Our algorithm was implemented in Java application on a computer with an Intel Core 2 Duo 2GHz processor, and 2GB of RAM. We experimented with similarity evaluation on a large case base consisting of 25000 pseudo-randomly generated cases, to ensure that efficiency is not an issue with the algorithm (we envisage only about 300 cases a year being added). Our system has been designed for ease of maintenance and flexibility; if efficiency becomes a problem, then the architecture has to be altered to improve execution speed.

We have explicitly coded our features and similarity measure into our Java software, as opposed to ‘adaptive programming’, described by Long et al. [12], using metadata instead. Our users require a tool to create new cases manually, so the features must be specified in our program code anyway. Our software offers both ‘creation from scratch’ and creation using CBR.

3.3 Adaptation for Reuse

One of the assumptions of CBR is that if two objects are similar then one can be adapted into the other. However, the well-known '15-puzzle' explained by Archer [13] illustrates that this assumption is questionable in some circumstances. In the 15-puzzle, numbered tiles are moved around on a board with the goal to finish with the tiles in a particular order. Some configurations of tiles have no tile in the correct position but are solvable. However, other configurations involve just two of the 15 tiles being in the wrong position but are completely unsolvable. It needs further consideration to establish whether this holds in our domain. As a fallback remedy, our users always have the option to manually create a knitting pattern.

Mitra and Basak [14] survey the adaptation methods used in many CBR systems, and classify them in various ways, e.g. knowledge lean and knowledge intensive. Our domain perhaps lends itself more to *knowledge intensive adaptation* since the cases are highly structured. Stochastic methods are likely to require extensive repair mechanisms. Derivational replay is not applicable since the way designs are constructed by humans is likely to be idiosyncratic and inconsistent.

Substitution (swapping parts of the existing and newly created cases) and transformation (making structural changes to the newly created case) seem particularly applicable. However, the drawback of these techniques is that they require domain knowledge, so they reduce one of the claimed benefits of CBR, i.e. the elimination of the knowledge elicitation bottleneck.

It is worth noting that our search space is fairly large: the questionnaire (see section 4.3) is capable of specifying approximately 10^{12} different garments. If we assume that ease of adaptation correlates with structural similarity, then the danger is that the gap between the new solution and the nearest existing case will be too large for any effective adaptation strategy.

One of the ways to avoid the large search space problem is proposed by Watson and Perera [15] in their 'divide and conquer approach'. Knitwear is composed of separate pieces, e.g. cardigans typically consist of a back, front, and two arms. The CBR could be done separately on these pieces. So, our new cycle might be:

- **repartition** the query case into constituent parts
- **retrieve** cases which have parts that are similar to the equivalent parts in the new case
- **reuse** the parts
- **revise** using one or more of the adaptation methods discussed
- **recompose** the parts back into a whole
- **repair** any inconsistencies between the parts
- **retain** the solution for the future.

The major disadvantage of the divide and conquer approach is that the parts need to be independent, as noted by Watson and Perrera. The potentially introduced inconsistencies are usually dealt with in the repair phase. In the case of a cardigan design, the inconsistency could be the production of parts with slightly mismatched armhole, which can be repaired using well-defined rules. One could imagine an inconsistency of a front and a back with different length as such, but this can be very simply avoided by using strong constraints on the size of the parts in the first place in the repartitioning phase.

Purvis & Pu's COMPOSER system [16] used multi-case adaptation in two case-based design problems. The adaptation phase was viewed as a constraint satisfaction problem; they use a greedy algorithm to find a good initial solution then employ a minimum conflicts algorithm. The algorithm uses heuristics to change the values of variables until all constraints have been satisfied. They claim that their approach requires less adaptation knowledge. The fact that the constraints have to be identified appears to be a disadvantage of this approach, but in our case most constraints are simple to specify (for example, matching length and matching armhole for parts, symmetries) and the creation of garments 'from scratch' will have to include them anyway for the purpose of consistency checking.

To ensure compliance with potentially unpredicted detailed user requirements, the user will always have the final say in accepting the adaptation proposed by the system and potentially propose changes before the solution is produced.

4 ARCHITECTURE AND IMPLEMENTATION

4.1 Process Flow

A useful case-based design system effectively automates the full CBR cycle: retrieve, reuse, retain and revise [3]. However, we start with an empty case base, so manual methods must be supported. Many studies (e.g. [17]) have confirmed that iteration must be supported when producing a semi-automatic design system. Accordingly, the users will be allowed to move back and forward between the questionnaire, sketch and chart stages. Automated creation may jump straight from the questionnaire to the chart.

4.2 Transformation Rules

To support the manual creation of garments, we require rules which will enable us to convert the questionnaire into the sketch, and the sketch into the chart. Equally, to ensure that the representations do not become inconsistent, we need inverses to these rules. This is analogous to the discussion given in Börner [18]. The rules are coded in the system using domain knowledge elicited from experts at Sirdar. For example, the feature-vector representation of an armhole shape is just a single integer. This is transformed into a series of points which either form the outline of the shape, or in some cases control points for a Bézier curve. This is then discretised in the chart into a matrix of individual stitches.

4.3 Editing and Creating Patterns

Figure 1 shows the stages of our knitwear design system starting with the questionnaire, followed by sketch editing and approval, then chart production and finally the sketch is converted into the written knitting pattern. Figure 1(a) shows an example of a fully completed questionnaire. Note that we seek to hide unnecessary complexity, so things are only shown when they are relevant. Also, feedback from designers indicated that they want to be able to proceed to a sketch as soon as possible. The 'basic' tab contains everything which is necessary to proceed to the sketch stage; the advanced tab can be left until later. The Random button creates a garment with pseudo-randomly generated data, utilising some heuristics, and populates the fields accordingly. The Clear button resets all the fields to blank and populates the fields accordingly, therefore offering potential for new garments. The other buttons are self-explanatory.

If the divided box is checked, this indicates that the front of the garment separates, i.e. it is a cardigan rather than a sweater, and the user can choose the type of fastener; if it is buttoned then they can additionally choose how many

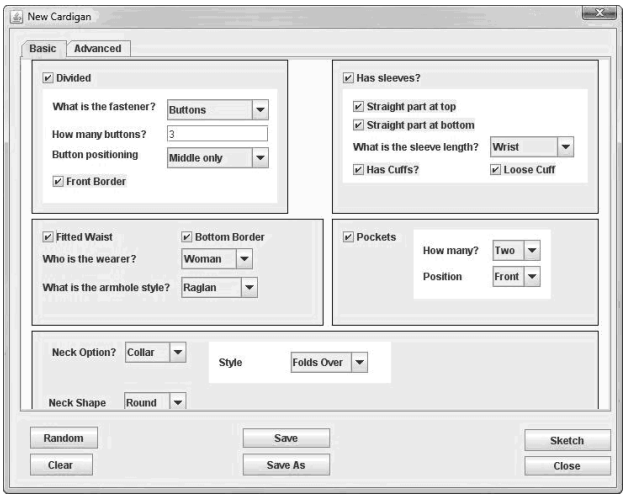
buttons and where they are. If there are sleeves then the user ticks the box and specifies the shape, and their length. If the garment has pockets, the user can choose the option indicating their position. There are additional items for the shape of the body, neck, and armhole.

On the advanced tab, which is not shown here, the user can select an option indicating what the 'background stitch' is. This is the default that will show on the chart (figure 1(c)), unless the user selects another stitch later. Similarly, if the user has opted for a bottom border, cuffs, front border or collar then they can choose the stitch for these. When the user presses the Save button the design is stored as an XML file, ready for reuse.

When users open an existing garment, our system will automatically jump to the sketch stage. The sketch initially shows the whole garment, as it would be constructed by a knitter in separate pieces, i.e. sleeves, back, and front. The user can click on any one of those pieces, for example a sleeve, which is shown in figure 1(b). The sketch is scalable and the line thickness can be varied. The shape of a piece can be modified through dragging of points with the mouse, within the limits allowed for that particular piece. For example, a set-in sleeve can change the shape of the underlying curve (implemented using Bézier curves), but cannot become a raglan sleeve.

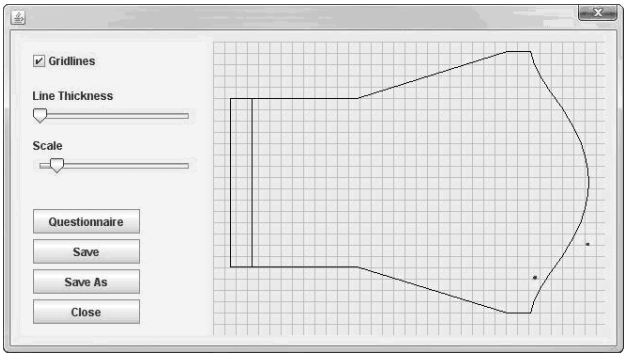
Figure 1(c) shows a portion of a knitting chart. The knitting chart is a matrix of elements indicating what the knitter should actually do step-by-step to achieve the shape and structure of the garment. Each element corresponds to one stitch and is represented by a symbol carrying significance. For example, a circle typically means to create a hole and a triangle to combine two stitches into one. Each row in the chart corresponds to a row of knitting in the finished garment. The user can edit these symbols by numerous means, e.g. drag and drop with a mouse.

The product of the design process, a written knitting pattern, is shown in figure 1(d). It is a series of textual instructions to the knitter, partly codified, using industry standard terminology. The knitting pattern can be produced in a fairly straight-forward automatic way from the knitting chart, using predefined rules.



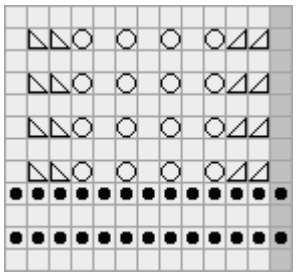
(a) Fully completed basic tab

↓



(b) Sketch

↓



(c) Chart

↓

1st Row.K0 [2:2:0:0:1], p2 [2:2:1:2:2], * k3, p2, rep from * to last 0 [2:2:4:0:1] sts, k0 [2:2:3:0:1], p0 [0:0:1:0:0].2nd Row.K0 [2:2:0:0:1], p2 [2:2:1:2:2], * yon, s1, k2tog, psso, yfwd, yrn, p2, rep from * to last 0 [2:2:4:0:1] sts, k0 [2:2:0:0:1], (yon, s1, k2tog, psso, yfwd, yrn, p1) 0 [0:0:1:0:0] times.3rd Row.K0 [0:0:1:0:0], p0 [2:2:3:0:1], * k2, p3, rep from * to last 2 [4:4:1:2:3] sts, k2 [2:2:1:2:2], p0 [2:2:0:0:1].From 1st to 3rd row sets patt. Keeping continuity of patt as set (throughout) work until back measures 44 [46:50:54:58:60]cm, (17 1/4 [18:19 ¾:21 1/4:22 3/4:23 3/4]in), ending with a rs row.

(d) Pattern

Figure 1: The phases of computer aided knitwear design

4.4 Similarity Preferences

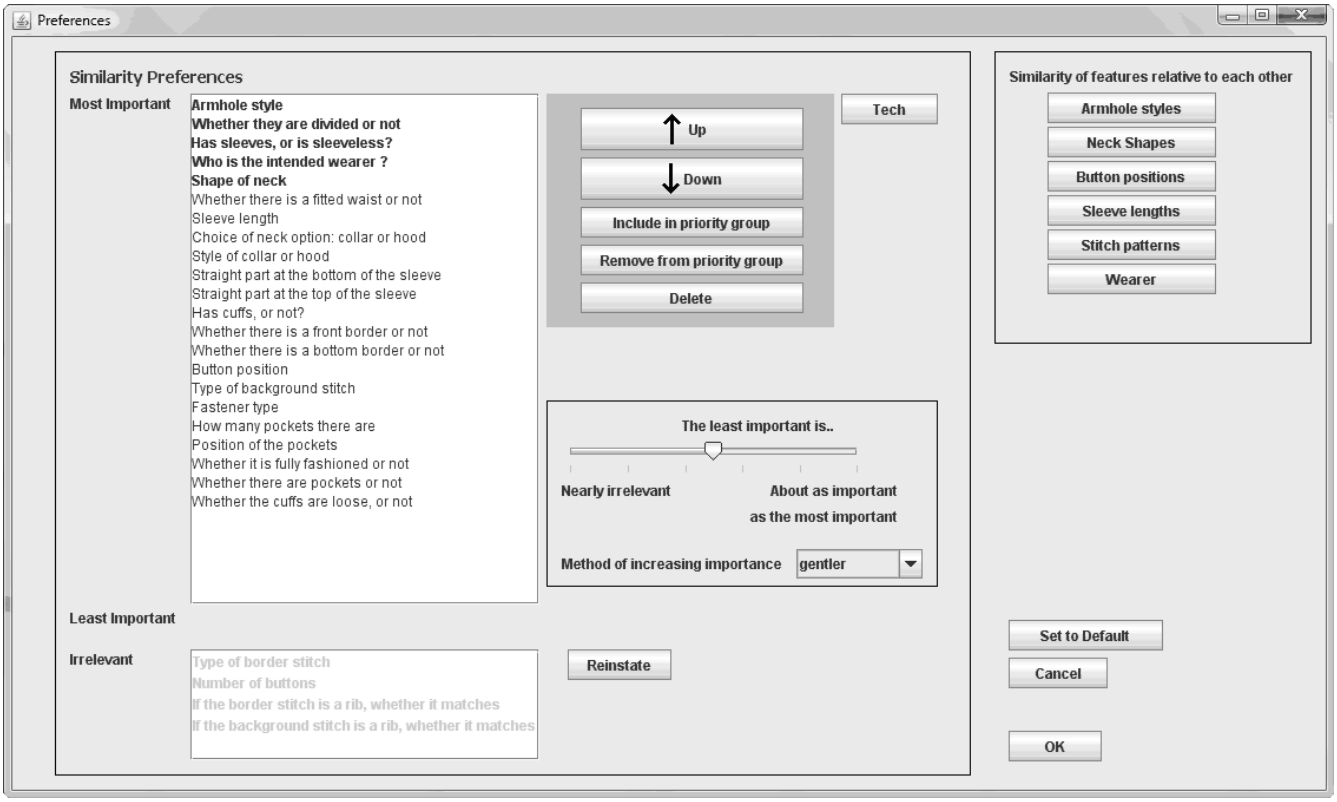


Figure 2. Preferences

In order to facilitate the retrieve stage of CBR, we allow the user to specify their preferences for similarity between the new pattern being created and the retrieved pattern from the case base. We allow the user to indicate the relative importance of features, for comparison; our approach involves ranking features in order of preference, since people are typically more comfortable doing this than they are assigning numeric weights.

Figure 2 shows the preferences window. On the left, the users can rank in order the importance of features; important items towards the top of the list are highly relevant. Items in bold are all of equal and maximum relevance, and items in the separate list at the bottom are irrelevant. Hence, we cater for situations in which the user feels that several factors are highly significant, but they cannot decide which one is more important. Also, by allowing factors to be marked as irrelevant, we allow the user to exclude things that have no bearing on similarity. The slider is used to set the scale. We offer a choice of seven functions for the progression of weights from one (most important) to zero (irrelevant). Figure 3 shows an example with 28 features, none of which are irrelevant. The shapes of the functions are shown; the middle is arithmetic progression.

In the preferences (Figure 2) we also allow the user to set the relative score for similarity between options for neck style, armhole style, etc. These options are available by clicking on the corresponding button in the top right box. Sleeve shapes are an example: set-in is similar to semi-set in, but different from raglan. Similarities are expressed using a Likert [19] scale, which is mapped linearly into the [0,1] range and the weighted scores are added up to produce the overall garment similarity as described in Section 3.2.

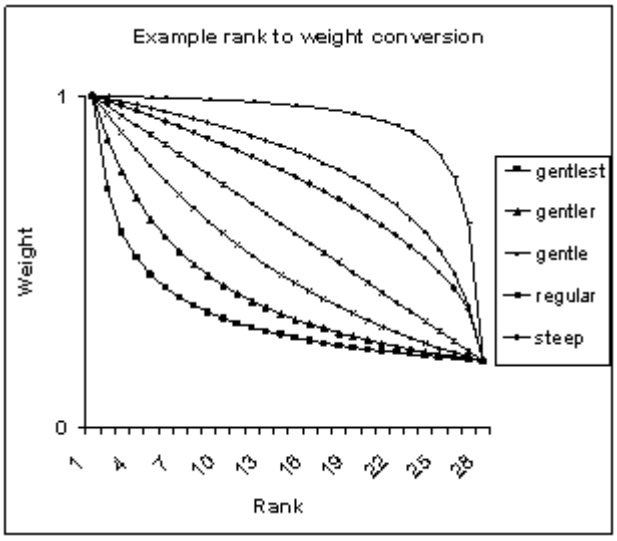


Figure 3: Functions for converting ranks to weights

5 CONCLUSIONS

Knitwear design involves many creative processes and as such it is a human activity that is hard to reproduce or automate using the computer. At the same time, the production of the associated knitting pattern includes

several stages that are repetitive and mechanical for the human, which suggests automation. In this paper we described how the process of knitwear design, including the production of the knitting pattern can be automated using case-based reasoning.

We proposed a software system that allows both design from scratch and design based on previously created patterns. The system works in stages, starting with a simple questionnaire, then a sketch, then a detailed chart. These stages are interactive; the user can decide how much they want to change the options offered by the system. The chart is then used to produce the pattern automatically. The proposed system has two main goals: to facilitate the design and to ease the burden of calculations and checks. The designers will be able to easily reuse past patterns or their parts. Errors that are often present in early stages of pattern production and propagate to later stages will be prevented from occurring.

We envision that through the use of case-based design, the company will be able to respond rapidly to fashion trends. Case-based reasoning systems reuse good practice, learning with each design that is produced, and thus becoming an automated repository of knowledge. Our feasibility study shows that CBR is particularly suitable for this problem domain. We investigated the important aspects of CBR: representation, similarity measures, adaptation, reuse, and retention using a prototype system for cardigans. We are planning to implement adaptation next, in order to have a fully functional system.

6 ACKNOWLEDGEMENTS

We are grateful for the support of EPSRC and Sirdar Spinning Ltd. via a CASE studentship. We also thank Sue Batley-Kyle for her valuable help on knitwear design.

REFERENCES

[1] Eckert, C.M., Cross N., Johnson, J.H., 2000, Intelligent support for communication in design teams: garment shape specifications in the knitwear industry, Design Studies, 21/1:99-112.

[2] Ekárt, A., 2007, Evolution of lace knitting stitch patterns by genetic programming, Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, 2457-2461.

[3] Aamodt, A. and Plaza, E., 1994, Case-based reasoning: Foundational Issues, Methodological Variations and System Approaches, AI communications, 7:39-59.

[4] Lopez De Mantaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Baltings, B., Maher, M.L., Cox, M.T, Forbus, F., Keane, M., Aamodt, A., Watson, I., Retrieval, reuse, revision, and retention in case based reasoning, The Knowledge Engineering Review, 20/3:215-240.

[5] Price, C.J. and Pegler, I.S., 1995, Deciding Parameter Values with Case-Based Reasoning, 1st UK Workshop on Case-Based Reasoning, Salford, 121-133.

[6] Kolodner, J.L., 1993, Case-Based Reasoning Morgan Kaufmann Publishers Inc.

[7] Bergmann, R., Kolodner, J. and Plaza, E., 2005, Representation in case-based reasoning, The Knowledge Engineering Review, 20/3: 209-213.

[8] Craw, S., 2008, CM3016 Knowledge Engineering (Case-Based Reasoning), available online at http://athena.comp.rgu.ac.uk/staff/smc/teaching/cm3016, accessed 23 June 2008.

[9] Mejasson, P., Petridis, M., Knight, B., Soper, A., Norman, P., 2001, Intelligent design assistant (IDA): a case base reasoning system for material and design, Materials & Design, 22/3:163-170.

[10] Watson, I., 1999, Case-based reasoning is a methodology not a technology, Knowledge-Based Systems 12/5:303–308

[11] Tversky, A 1977, Features of Similarity Psychological review,. 84/4:327-352.

[12] Long, J., Stoeckin, S, Schwartz, D.G. and Patel M.K., 2004, Adaptive Similarity Metrics in Case-based reasoning, Proceedings of Intelligent Systems and Control, Honolulu, Hawaii, USA.

[13] Archer, A.F., 1999, A Modern Treatment of the 15 Puzzle, The American Mathematical Monthly, 106/9:793-799.

[14] Mitra, R and Basak, J, Methods of Case Adaptation: A Survey, International Journal of Intelligent Systems archive, 20/6: 627:645

[15] Watson, I and Perera, S, 1998, A hierarchical case representation using context guided retrieval, Knowledge-Based Systems 11:285-292.

[16] Purvis, L. and Pu, P., 1998, COMPOSER: A Case-Based Reasoning System for Engineering Design, Robotica, 16/3:285-295

[17] Parmee, I., Cvetkovic, D., Bonham, C., Packham, I., 2001, Introducting prototype evolutionary systems for ill-defined, multi-objective design environments, Advances in Engineering Software, 32/6:429-441.

[18] Börner, K., 1994, Structural similarity as guidance in case-based design, in: S. Wess, K.D. Althoff, M. Richter (Eds.), Topics in Case based Reasoning, Springer, Kaiserslautern, 1993, pp. 197–208

[19] Likert, R., 1932, A Technique for the Measurement of Attitudes, Archives of Psychology, 140: 1-55.