Transactions on Case-Based
Reasoning for Multimedia Data
Vol.1, No 1 (2008) 21-36
© ISSN: 1867-366X (Journal),
IBaI Publishing ISSN 1864-9734

www.ibai-publishing.org

Case-Based Reasoning Adaptation for High Dimensional Solution Space

Ying Zhang[1], Panos Louvieris[1] and Maria Petrou[2]

[1]Surrey Defence Technology Centre, University of Surrey, Guildford, UK
[2]Electrical and Electronic Engineering, Imperial College, London, UK
{Y.Zhang, Panos.Louvieris}@surrey.ac.uk, Maria.Petrou@imperial.ac.uk

Abstract. Case-Based Reasoning (CBR) is a methodology that reuses the solutions of previous similar problems to solve new problems. Adaptation is the most difficult stage in the CBR cycle, especially, when the solution space is multi-dimensional. This paper discusses the adaptation of a high dimensional solution space and proposes a possible approach to it. A Visualisation induced Self Organising Map (ViSOM) is used to map the problem space and solution space first, then a Back Propagation (BP) network is applied to analyse the relations between these two maps. A simple military scenario is used as a case study for evaluation purposes.

1. Introduction

Case-Based Reasoning is a methodology that reuses the solutions of previous similar problems to solve new problems. Adaptation is considered to be one of the most difficult parts of the CBR cycle. Wilke and Bergmann [1] classify adaptation into three main types: Null adaptation, transformation adaptation and generative adaptation.

Null adaptation simply applies the solution from the retrieved cases to the target case. It is the approach adopted by a simple Nearest Neighbour (NN) technique and maybe combined with taking the inverse distance weighted mean for K Nearest Neigbours (KNN) when K>1. *Transformation adaptation* modifies the old solution derived from the retrieved cases. These are structural transformations, which are based on some function of the target and retrieved case feature vectors, and rule-based transformations. The rules are either elicited from the domain experts or learnt using an induction algorithm. *Generative adaptation* entails deriving the solution to the problem from scratch. The derivation is handled by the case based system, largely independent of the case base. In practical application, all the above adaptations could be combined. A general framework of case adaptation is proposed, which can be easily expanded if necessary.

Creating an automatic adaptation mechanism that is able to determine how the solution needs to be modified to fit the new circumstances is a complex affair. Case adaptation generally requires detailed knowledge of both the task and domain at hand. However, adaptation knowledge is not always accessible and available.

The simplest adaptation strategy consists of using adaptation rules to resolve differences and possible conflicts between the old case and the new problem. In order to overcome the difficulties and limitations of rule-based adaptation, Leake *et al.* [2] proposed a hybrid case-adaptation process combining memory of previously applied adaptations with rules that are able to find in the system's memory the appropriate information for guiding and implementing the necessary transformation. The system's memory retains not only the transformation operation during any adaptation process, but also a trace of the steps taken during the memory's search. Although considered powerful [3], Leake's approach is limited by the need to consider only one adaptation target at any time. In addition, this approach is not appropriate for CBR systems that have a modest knowledge acquisition capability. This is because the method relies on the availability of substantial adaptation knowledge and its explicit representation. Finally this method also relies on the intensive involvement of the user whenever the system's adaptation knowledge is insufficient. Hanney and Keane [4] proposed building adaptation rules directly from the case-base by analysing the differences between cases and their corresponding solutions and identifying, if possible, a plausible pattern. Jarmulak *et al.* [3] also developed an adaptation method based on the use of the CBR knowledge content. Each case in the system's memory is used as a test case and compared with the others in the case-base that are most similar to it. For each comparison made, an adaptation case is constructed. This contains information about the differences between the problems and solutions of the test case and the retrieved cases, as well as the description of the test case and the proposed solution. As a new problem arises, the adaptation cases are utilised to estimate the correctness of the proposed solution and suggest the necessary adjustments.

The above-mentioned methods are referred to as "knowledge-light methods" [5], learning adaptation knowledge from the CBR system's own cases and treating them as sources of knowledge. They initially pre-process the information extracted and, afterwards, pass it to a learning algorithm. The learning algorithm must be designed with respect to the problem domain under investigation and the adaptation goal considered. It transforms the pre-processed knowledge to obtain the required adaptation solution. Knowledge-light adaptation methods must be supported by a significant amount and variety of knowledge contained in the CBR system. Insufficient knowledge may adversely affect performance. Furthermore, adaptation knowledge obtained from a learning algorithm must be correctly and properly combined with knowledge already stored in the adaptation module, resolving, if necessary, possible contradictory and incompatible situations.

Many CBR systems' solution spaces are only one dimensional, such as the price of a property, or the classification of a case, etc. In our research, CBR is applied to find the suitable course of action (COA) for a military scenario. A COA is represented by the entities' waypoints with corresponding times. Therefore, our case solution space is multi-dimensional. We could simply treat it as several single dimensions, apply the same methodology on each individual dimension and then combine the results. This, however, would treat a COA as a collection of independent decisions. This clearly is not correct. Therefore we propose another approach to solve case adaptation in this situation.

The rest of this paper is organised as follows. Section 2 discusses possible normal adaptation approaches. Section 3 introduces the Self Organizing Map (SOM) and ViSOM. In section 4, we discuss how to find the target case solution using the techniques outlined in Section 3. In Section 5, we present the experimental design. Evaluation and results are in section 6. Finally, we present our discussion and conclusions in section 7.

2. Normal adaptation approach

For our system, the straightforward and direct adaptation approach is based on domain knowledge. We could ask human commanders to revise the suggested solution directly, or adapt retrieved cases using the adaptation rules, based on army doctrine or other domain knowledge acquired from human commanders. Gradually the performance of the system could be improved by adding new cases.

When domain knowledge is not available, a neural network can be used to adapt the retrieved cases automatically. This is the approach adopted here. In particular, we set up a three layer BP network. The network is trained using problem space differences between all pairs of cases as input, while the solution space differences between the corresponding cases are the target output for each pair. For example, assume that there are five cases (C1, C2, C3, C4, and C5). Then the input of the BP network consists of the problem space differences C1-C2, C1-C3, C1-C4, C1-C5, C2-C3, C2-C4, C2-C5, C3-C4, C3-C5 and C4-C5. The target outputs are the solution space differences between the same pairs of cases. Because CBR is based on the idea that similar problems have similar solutions, we analyse how similar these cases are, and how similar their solutions are. Once the BP network is trained, the problem space difference between the target case and its most similar case is input into the network, and then the solution space difference between these two cases is obtained. Thus the solution of the target case can be achieved.

However, when the case problem space and solution space both are of high dimensionality, the construction of the neural network under these circumstances is complex, and a large number of cases are required to train the neural network. This is particularly problematic here, given that only a small number of training samples are available. To solve this problem, we apply SOM to the case problem and solution spaces first to reduce the size of the BP network. SOM can dramatically reduce the data dimensionality, and can be used to help us visualise the case base as well.

3. SOM and ViSOM

A SOM is an unsupervised neural network algorithm that has been successfully used in a wide variety of applications, such as pattern recognition, image analysis, fault diagnosis etc. The basic algorithm was inspired by ideas about the way in which various human sensory impressions are neurologically mapped into the brain such that spatial or other relations between stimuli correspond to spatial relations between the neurons. This is called competitive learning [7]. A SOM consists of two layers of neurons, an input layer with n input nodes, giving the n-fold dimensionality of the input vectors, and N output nodes, giving N decision regions. Every input node is connected to every output node. All the connections are weighted. A SOM forms a nonlinear projection from a high-dimensionality data manifold onto a low-dimensionality grid. The basic process is as follows.

1. Initialization: choose the random value of weight vectors of all neurons
2. Similarity matching: using the angular or the Euclidean distance. The Euclidean distance between patterns *i* and *j* is:

$$d_{ij} = \|x_i - x_j\| = \sqrt{\sum_{n=1}^{N}(x_{in} - x_{jn})^2} \tag{1}$$

The smaller the Euclidean distance d_{ij}, the closer the vectors. The angular distance is based on the inner product of the vectors

$$\cos\theta = \frac{x^T y}{\|x\|\|y\|} \tag{2}$$

where $\|x\| = \sqrt{x^T x}$ is the Euclidean norm of the vector.

Here, the bigger the cosine value $\cos\theta$, the more similar the vectors. So we can find the best-matching winner $i(x)$ at time t:

$$i(x) = \text{argmin}_j \, \| x(t) - w_j \| \qquad j = 1,2,\ldots,N \tag{3}$$

3. Updating: adjust the synaptic weight vectors of all neurons, using the update formula:

$$w_j(t+1) = \eta(t)[(d_r - d_j) * w_j(t)], \text{ where } j \in \mathbf{N}i(x), \tag{4}$$
$$\text{unchanged otherwise}$$

Here $\eta(t)$ is the learning rate at the current time t and **N** is the neighborhood function centred around the winner, $i(x)$. The winning unit and its neighbors are adapted to represent the input by modifying their reference vectors towards the current input. The amount the units learn will be governed by a neighborhood kernel, which is a decreasing function of the distance of the units from the winning unit on the map lattice. The largest weight adjustment which is positive occurs for the winner, and smaller positive changes are made to adjacent neurons according to distance of node j, d_j, from the winning node, until at some radial distance d_r the weight adjustment falls to zero (maximum allowable distance within **N**). This effect can be conveniently implemented by the Gaussian function:

$$\Lambda(j,t) = \exp(\frac{-d(j)^2}{2\sigma(t)^2}) \qquad (5)$$

Where σ^2 is the variance parameter specifying the spread of the Gaussian function according to the distance d of node j from the winning node. Its effect decreases as the training progresses due to t.

In order to speed up the computation, a "Chef Hat" function can be used, in which only identical positive weight changes are made to those neurons within radius r.

4. Continuation. Continue with step 2. Both η and Λ are dynamically calculated during training, until no significant changes are observed. At the beginning of the learning process the radius of the neighborhood is fairly large, but it shrinks during learning. This ensures that the global order is already established at the beginning, whereas towards the end, as the radius gets smaller, the local corrections of the model vectors in the map will be more specific.

SOM is a very good tool for mapping high-dimensionality data into a low dimensionality feature map, typically of one or two dimensions. However, it does not faithfully portray the distribution of the data and its structure. Several measures can be used to quantify the quality of map to indicate the best result. The average quantization error is the average distances between the vector data and their prototypes. Generally, when the size of the map increases, there are more units to represent the data, therefore each data vector will be closer to its best matching unit, thus the average quantization error will be smaller.

One of the most important beneficial features of SOM is the ability to preserve the topology in the projection. The accuracy of the maps in preserving the topology or neighbourhood relations of the input space has been measured in various ways. The topographic product, introduced by Bauer and Pawelzik [6], tries to find folds on the maps. Since the SOM approximates the high dimensionality input space by folding it, the topographic product can be an indicator of topographic error. However, this measure does not differentiate the correct folds following a folded input space from erroneous folds. Kohonen himself proposed another approach to measure the proportion of all data vectors whose first and second best matching units are not adjacent [7]. This is called topographic error. The smaller the topographic error, the better the SOM preserves the topology. Generally, the higher the dimensionality of an input space, the larger the topographic error. This is due to the increasing difficulty of projecting units in the right order and the increasing dimensionality of the prototype.

There have been a number of research efforts to enhance topology preservation of SOM. In [8], SOM was trained to minimize the quantization error first, and then minimize the topological error in the second stage. A Double SOM (DSOM) uses a dynamic grid structure instead of a static structure, together with the classic SOM learning rules to learn the grid structure of the input data [9]. The Expanding SOM (ESOM) preserves not only the neighbourhood information, but also the ordering relationships, by learning the linear ordering through expanding [10].

ViSOM uses a similar grid structure as normal SOM [11]. The difference is, instead of updating the weights of neurons in the neighbourhood of the winner by

$$w_k(t+1) = w_k(t) + \alpha(t)\eta(v,k,t)[x(t) - w_k(t)] \tag{6}$$

where $\eta(v,k,t)$ is the neighbourhood function, ViSOM updates the neighbourhood according to:

$$w_k(t+1) = w_k(t) + \alpha(t)\eta(v,k,t)\Big([x(t) - w_v(t)] + [w_v(t) - w_k(t)]\frac{(d_{vk} - \Delta_{vk}\lambda)}{\Delta_{vk}\lambda} \Big) \tag{7}$$

where $w_v(t)$ is the weight of the winning neuron at time *t*.

d_{vk} is the distance between neurons *v* and *k* in the input space

Δ_{vk} is the distance between neurons *v* and *k* on the map.

λ is a positive pre-specified resolution parameter. The smaller it is, the higher resolution the map provides.

The key feature of ViSOM is that the distances between the neurons on the map can reflect the corresponding distances in the original data space. The map preserves the inter-neuron distances as well as topology as faithfully as possible.

We employ ViSOM on both the case problem space and the case solution space. Once two such ViSOM have been set up, the location of cases in the case problem space ViSOM can be used as input while the location of the corresponding case in the case solution space ViSOM can be used as output. Because these locations are only two dimensional, the BP network structure is much simpler than the one created from directly inputting the original dataset. This approach tries to mimic the case problem space and the case solution space as input and output patterns, respectively, and map the problem to the solution by adjusting the weights of the connections.

Instead of using the actual location, an alternative approach is to employ, as input, the difference of locations between each case pair of the case problem space ViSOM. Likewise, the difference of locations between the same case pair of the case solution ViSOM is the output. The difference between target case and its nearest case is input to the network after it is trained. In this way the target case location in the case solution space map is acquired.

Because the ViSOM can preserve the inter-point distances of the input data on the map, the located nearest case to the target case can be adapted to the target case solution. In our experiment, a 3-layer BP network was used. The input vector had 2 elements. There were 5 neurons in the hidden layer while 2 neurons in the output layer. The transfer function for both layers was a tan-sigmoid. The training function employed was *trainlm* (the Levenberg-Marquardt algorithm), a very fast training method suitable for small networks.

4. How to find the target case solution

After the target case location in the case solution space ViSOM is acquired, if there is a previous case projected on the same location, the solution of this case will be chosen as the target solution. If there are more than one previous cases projected on the same location, then the mean of these case solutions will be used as the solution for the target case. However, if there is no previous case projected on this location, how can we find the corresponding high dimensional solution for this exact location? There are several possible solutions, as follows.

First, the prototype vector of the corresponding node of the case solution space ViSOM may be used. Once the solution space ViSOM is trained, each node has its corresponding prototype vector. As the location of the target case on the solution space ViSOM is known, the corresponding node may be regarded as the winning node for the target case solution, therefore its weight may be regarded as the output suggestion.

Second, KNN with distance inverse weight may be used as well. However, instead of distances in the problem space, the distances between the target case location and its neighbours in the solution space map are used.

5. Experiment Design

We apply CBR to military decision-making. When military commanders face a tactical mission, they develop a set of COAs to achieve their objective. Generally, A COA is composed of commands for each entity in the troops. It is difficult to transform these human natural language commands into a form which the computer can employ. We choose MAK VR-Forces [13] as our simulation environment. It provides both a set of Application Program Interfaces (API) for creating computer generated forces and an implementation of those APIs. By using VR-Forces, we can interactively add individual entities to a simulation. The entities may include land vehicles, such as a T-80 tank, a BTR-80 combat vehicle, a BMP-2 infantry fighting vehicle, air entities such as F-16 Falcon fighting aircraft, F/A-18 Hornet aircraft, and surface and subsurface entities such as a Landing Craft Air Cushion (LCAC).

Because VR-Forces is our scenario simulation environment, our system directly outputs to it. No matter how complicated a COA is, (e.g. attack hastily, attack deliberately, drawback, remain stationary or occupy), it boils down to a series of movements of the entities in VR-Forces simulation. So a COA is represented by a series of positions of entities. Once the entities reach suitable waypoints, they fire at appropriate targets automatically. Therefore, we decided to represent a COA by the entities' waypoints and the corresponding times for each entity to reach its waypoint. In other words, a COA is represented by a matrix, quite similar to the *synchronization matrix* [14] used by human commanders to formulate COAs. The matrix is composed of each entity's waypoint location at different time steps during the scenario. Each row corresponds to one entity and each column at one time step.

In this paper, an example scenario in VR-Forces is presented as a test. In this exercise, four hostile vehicles (BMP2 1, BMP2 3, T80, and BMP2 2) are arrayed behind a minefield. Three platoons (Blue, Red and White) of four tanks each suppress the hostile vehicles, allowing two engineering vehicles to clear the minefield.

The scenario, which is shown in Figure 1, plays out as follows:

Blue platoon stays in position and fires on hostile forces.

Red platoon advances to waypoint red and provides cover for the engineers.

White platoon advances to waypoint white and provides cover for the engineers.

The engineering entities follow behind the Red platoon. When the Red platoon is at waypoint red and then hostile forces are destroyed, the engineering entities advance on the minefields to clear them.

5.1 Scenario representation

In related military CBR systems, cases can be created from war stories, prior experience, tactics and doctrine [16]. According to domain knowledge and experience, Mission, Enemy, Terrain, Troops, and Time (METT-T) are the factors human commanders usually consider in the real battle-field. It is also very convenient to use these parameters to represent a scenario here.

In this simple scenario, the mission, namely to breach the minefield, is fixed. In order to simplify the scenario representation, we shall not include mission representation for this project, but it may be included in a future study.

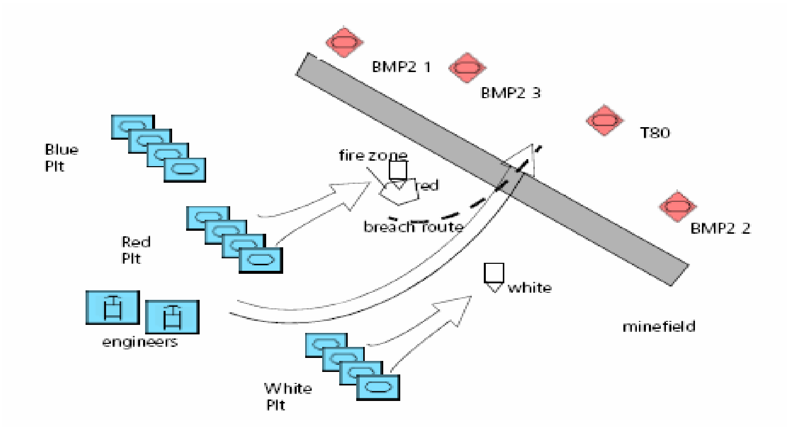

Figure 1. Breaching Scenario

Each entity may be represented by a symbolic object. In that case, however, comparing the effectiveness of the opposing sides becomes a problem. So, instead of representing each troop individually, we combine them together, and use their Scaled Strength Ratio to represent their capabilities. According to domain knowledge, a T80 tank is assigned power 5 while an M1A2 is assigned power 6, and BMP2 is assigned power 3 [15]. Actually, it is not so easy to estimate an entity's power because its power may be affected by other factors, such as the location of the entity (e.g. even if the entity is very powerful, it is ineffective when it is far away from its intended target) or when maneuverability is restricted by the current terrain (e.g. even the most powerful tanks are not a major threat to an enemy on the other side of an impassable river). However, in order to simplify the problem, we currently omit these considerations. Meanwhile, we may treat Combat Effectiveness as ordinal data instead of symbolic. Therefore, "full capability" is represented by 1, while "Inoperable" is represented by 0 and "Degraded" is represented by 0.5. Then the scaled strength ratio is defined as:

$$\text{Scaled Strength Ratio} = \frac{\sum_{i=1}^{n} T_i C_{ti}}{\sum_{j=1}^{m} E_j C_{ej}} \tag{8}$$

Where T_i is the power assigned to friendly troop *i* and C_{ti} is its combat effectiveness. E_j is the power assigned to enemy *j* and C_{ej} is its combat effectiveness.

In VR-FORCES, the whole simulated battle-field is overlaid by a grid coordinate system. Instead of using complicated geocentric locations, we shall use this (X, Y) grid information to represent an entity's location. Entities not exactly at the centre of a grid cell will be assigned to the closest grid cell. The following table shows an example.

We can store the grid information for all the entities as a matrix. If there is no entity in a grid cell, we assign 0 to it; otherwise, we put the entity type in that grid cell. In Table 1, we have entities *a* and *b* in the battle field, so the corresponding matrix is:

(0, 0, 0, 0, 0… 0;

0, 0, *a*, 0, 0… 0;

0, 0, 0, 0, 0… 0;

0, 0, 0, 0, *b*… 0;

0, 0, 0, 0, 0… 0;

……

0, 0, 0, 0, 0… 0)

There are two kinds of enemy entity: BMP2 and T80. The numbers of each type of entity may vary. Assume that we represent BMP2 by 1 and T80 by 2. The enemy force then may be represented by a matrix the elements of which may take values 0, 1 or 2. We consider this type of representation as categorical data.

There is another way to represent the data in Table 1. The entity location may be stored according to the coordinate system used. For example, the entities in Table 1 may be represented by:

a= (3, 2)

b= (5, 4)

Table 1. Grid representation

0	1	2	3	4	5	..	N
1							
2			a				
3							
4					b		
5							
..							
N							

This approach is suitable for representing the friendly forces in our experimental scenario. There are four main parts of friendly troops, namely the Blue platoon, the Red platoon, the White platoon and the Engineers. So, we may store their locations using the coordinate system. Because the terrain is fixed, only the X and Y-axes values are needed. These are continuous data.

Finally, in order to simplify things even more, we omit time in the case representation as well. Table 2 shows the representation of this scenario.

Table 2. Scenario representation

Scaled Strength Ratio	Enemy Force Matrix (3 X 9)	Blue Platoon (Xb, Yb)	Red Platoon (Xr, Yr)	White Platoon (Xw, Yw)	Engineers (Xe, Ye)

5.2 Case solution part

As discussed earlier, a COA may be represented by the matrix composed of the entities' waypoint locations at different time steps during the scenario. Each row corresponds to one entity and each column to one time step. For this simple scenario, we use the following four routes to represent the solution part: Blue platoon route, Red platoon route, White platoon route and Engineers' route. Each route is composed of five waypoints at corresponding time steps, including the start point and the end point, which can be derived from the case description part. For each waypoint, because the terrain is fixed, only X and Y are required. Therefore in the solution part we only need to describe another three waypoints, as shown in Table 3. In a more sophisticated version of this scenario, one should also include time to each waypoint to indicate the time in which the troops should reach that waypoint.

5.3 Data Collection

In order to collect data to populate our case base, we randomly chose values for the Scaled Strength Ratio, enemy entities locations and friendly troops entities' locations to generate the case description part for 300 cases. Then according to each of the case descriptions, we chose a suitable COA based on common sense and simulated it in VR-Forces. We recorded the result, including factors such as whether the goal was achieved or not, the enemy's remaining (fire) power, the friendly troops' remaining power, etc. The COA with the highest winning value *W*, which was a weighted function of these factors, was chosen as the suitable solution.

$$W = w_1 G + w_2 \frac{fr+1}{er+1} \tag{11}$$

Here w_1, w_2 are weights defined from domain knowledge. *G* describes whether the goal was achieved or not. fr is the friendly troops' remaining power while er is the enemy's remaining power.

Table 3. Troops section route representation

Blue Platoon Route $(Xb^1,Yb^1),(Xb^2,Yb^2),(Xb^3,Yb^3)$
Red Platoon Route $(Xr^1,Yr^1),(Xr^2,Yr^2),(Xr^3,Yr^3)$
White Platoon Route $(Xw^1,Yw^1),(Xw^2,Yw^2),(Xw^3,Yw^3)$
Engineers' Route $(Xe^1,Ye^1),(Xe^2,Ye^2),(Xe^3,Ye^3)$

Because it is impossible to exhaust all possible enemy plans with different dispositions, their waypoints may also be changed. We need to choose cases that are varied and cover most of the variations in the scenario. Figure 2 and Figure 3 show two examples of them.

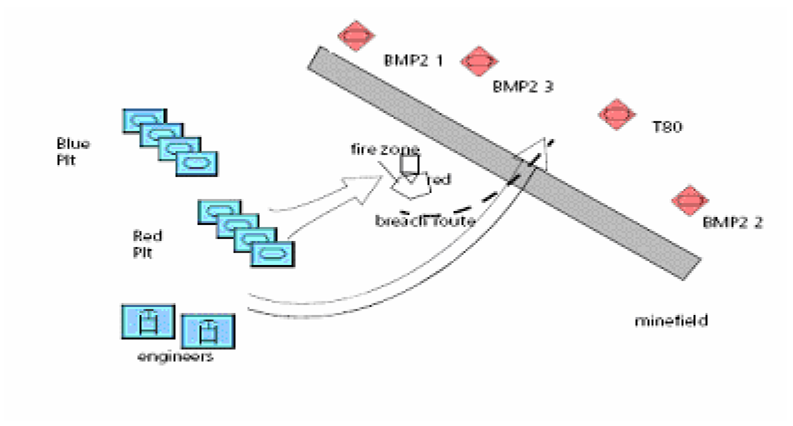

Figure 2. Breaching exercise variation 1 (White platoon is missing)

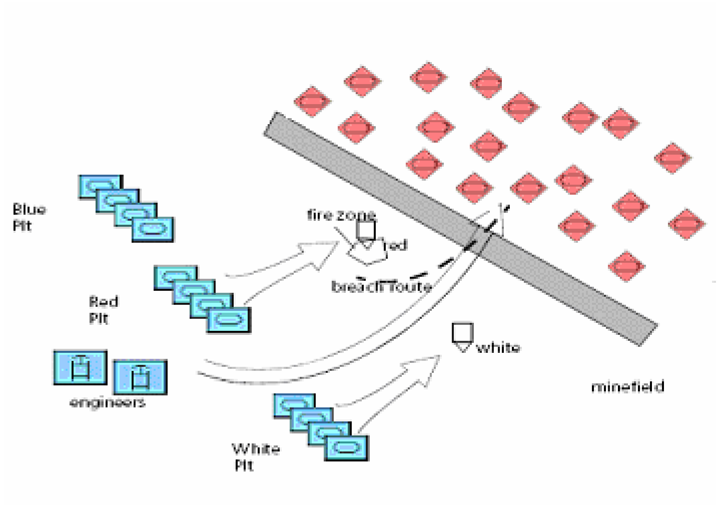

Figure 3. Breaching exercise variation 2 (Enemy firepower is stronger than that of the friendly troops)

6. Evaluation

There are not many military decision support systems, and even similar projects are based on different scenario data, so it is difficult to apply benchmarking. The direct evaluation approach is based on domain experts, such as military Subject Matter Experts (SME). We may utilize the Turing test on the evaluation cases, and compare the resultant outputs with the suggestions of SMEs. A more practical approach is to simulate the generated COAs in VR-Forces, and find whether the corresponding COA can help the friendly troops to achieve their goal or not. The feedback can be input back to the system to increase its learning ability.

There were 300 cases in our case base. We divided the case base into two sets: one for training, and the other one for evaluation. Cases in the training set were input to the system for training the ViSOM and the BP networks while cases in the evaluation set were used to judge how good the outputs were.

In our experiment, the cases were divided into two sets in the ratio of 2:1. Therefore, 200 cases were chosen randomly as the training set, while the remaining 100 cases were used for evaluation. This process was repeated 10 times. The results are shown in Table 4. The Mean Error (ME) is the mean Euclidean difference between the predicted COA and the corresponding real COA of these 100 cases. In order to calibrate them, these differences are divided by the norm of their corresponding real COA, their average is then recorded as Mean Percentage Error (MPE).

Row LocKNN in table 4 shows the ME and MPE results obtained using the location in the map (for different sizes of map) to train the BP, with KNN (with different K values) to acquire the solution. Row LocProto shows the results obtained using the location in the map (with different size of map) to train the BP, with the map prototype vector for the solution. DifKNN shows the results obtained using the locations' difference (with different size of map) to train the BP, with KNN (with different K values) to acquire the solution. DifProto shows the results obtained using the locations' difference (with different size of map) to train the BP, with the map prototype vector for the solution.

Table 4. Experimental results for all the methods discussed

		ME	MPE
LocKNN (with different K)	K=1	0.439	6.4142
	K=3	0.435	6.3173
	K=5	0.445	6.573
LocProto (with different size of map)	10 X 10	0.469	6.772
	20 X 20	0.458	6.204
	10 X20	0.473	6.808
	30 X30	0.466	6.647
DifKNN (with different K)	K=1	0.432	6.128
	K=3	0.421	5.946
	K=5	0.430	6.437
DifProto (with different size of map)	10 X 10	0.412	6.119
	20 X 20	0.399	5.874
	10 X20	0.408	6.107
	30 X30	0.393	5.475

Table 5 shows the ANOVA result of the various approaches discussed, in which, *P-value* << 0.05 and *F* >> *F crit* . Thus, the differences between the results in Table 4 are statistically significant.

By inspecting Table 4, it can be seen that using the difference of locations to train the BP and map prototype achieves the best result. Using map locations to train the BP and KNN for the solution gives the worst result. This is because the difference of locations contains more information than pure location itself. Using difference also increases the size of the training sample. Furthermore, using the map prototype vector for the solution, one must consider the whole solution space, not just the nearest neighbours.

Table 5. ANOVA result of different approaches

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Loc k=1	10	4.39	0.439	8.89E-05
Loc k=3	10	4.35	0.435	2.22E-05
Loc k=5	10	4.45	0.445	1.34E-08
Loc 10X10	10	4.69	0.469	2.25E-05
Loc 20X20	10	4.58	0.458	4.45E-05
Loc 10X20	10	4.73	0.473	1.38E-08
Loc 30X30	10	4.66	0.466	3.53E-08
Dif k=1	10	4.32	0.432	4.44E-05
Dif k=3	10	4.21	0.421	2.27E-05
Dif k=5	10	4.3	0.43	1.08E-06
Dif 10x10	10	4.12	0.412	2.29E-05
Dif 20X20	10	3.99	0.399	3.21E-05
Dif 10x20	10	4.08	0.408	8.07E-05
Dif 30x30	10	3.93	0.393	4.89E-08

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	0.087469	13	0.006728	246.4893	7.45E-83	1.798584
Within Groups	0.003439	126	2.73E-05			
Total	0.090908	139				

7. Conclusions

In this paper, we described how to achieve case adaptation for a case base with high dimensional solution space. This is a very difficult task, especially for high dimensional data and a limited size case base. We proposed to map the problem space and the solution space to two different ViSOM and then analyse the mapping between these two maps. A simple military scenario was used as an example. Although all the case attributes had numeric values in our example dataset, non-numeric attributes can be converted to numeric first and also used. Thus our approach has the potential to be applied to other datasets as well.

Once two ViSOM were set up, our target was to find the relationship between these two maps. We employed a BP network, and used the difference in locations to train the network. ViSOM was used here because the SOM does not proportionally represent the distances between nodes. However, if we could use a Hebbian network [17] to find the relation between two SOM, the process would be more direct and easier to explain. In Hebbian learning, when two neurons on each side of a synapse are activated simultaneously, the synaptic weights are reinforced, while the synaptic weights are weakened when two neurons are uncorrelated. It is based on the modification of the connections between neurons, or more specifically is an unsupervised training algorithm that increases the synaptic strength between two neurons that are active at the same time. Once a Hebbian network has been set up for the two SOM, future experiments will be needed to prove whether the result is better. However, it is a very interesting approach as it mimics the learning process of biological brains and combines the projection between the case problem and case solution space.

The research and development of military applications is a very demanding area, but CBR is a very suitable methodological approach to mimic the naturalistic decision making process of human commanders. Therefore, using CBR to suggest possible COA for a military scenario is a reasonable assertion. From our experiment, the initial suitability of CBR, as a proof of concept, has been investigated within the context of an established simulation environment VR-FORCES. However it is recognised that this research is an initial ‘proof-of-concept’ endeavour and there is the need for further work and improvement.

References

1. Wilke, W., Bergmann, R.: Techniques and knowledge used for adaptation during case-based problem solving. Tasks and Methods in Applied Artificial Intelligence, LNAI 1416, Springer-Verlag: Berlin, pp.497-505, 1998
2. Leake, B., Kinley, A., Wilson, D.: Acquiring case adaptation knowledge: a hybrid approach, Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI Press, Menlo Park, CA, 1996.
3. Jarmulark, J., Craw, S., Rowe, R.: Using case-base data to learn adaptation knowledge for design, Proceedings of the Seventeenth IJCAI Conference, Morgan Kaufmann, San Mateo, CA, pp. 1011-1016, 2001.

4. Hanney, K., Keane, M. T.: The adaptation knowledge: how to easy it by learning from cases, Proceedings of the Second International Conference on Case-Based Reasoning, Springer, Berlin .pp. 359–370,1997
5. Wilke, W., Vollrath, I., Althoff, K.D., Bergmann, R.: A framework for learning adaptation knowledge based on knowledge light approaches, Proceedings of the Fifth German Workshop on Case-Based Reasoning, 1997
6. Bauer, H.U., Pawelzik, K. R.: Quantifying the neighbourhood preservation of self-organizing feature maps. IEEE Transactions on Neural Networks, Vol.3, No.4, pp. 570-579,1992.
7. Kohonen, T.: Self-Organizing Maps. Vol.30 of Springer Series in Information Sciences, 3rd ed., Springer-Verlag,Berlin Heidelberg ,2001.
8. Kirk, J. S., Zurada, J.M.: A two-stage algorithm for improved topography preservation n self-organizing maps, in: 2000 IEEE International Conference on Systems, Man, and Cybernetics, Vol.4, IEEE Service Center,2000, pp.2527-2532,2000
9. Su, M.C., Chang, H.T.: A new model of self-organizing neual networks and its application in data projection, IEEE Transactions on Neural Networks 12(1) pp. 153-158,2001
10. Jin, H.D., Shum, W.H., Leung, K.S., Wong, M.L.: Expanding Self-Orgainizing Map for data visulaization and cluster analysis, Information Sciences, 163, pp. 157-173, 2004.
11. Yin, H.: ViSOM - A novel method for multivariate data projection and structure visualization, IEEE Transactions on Neural Networks 13(1).pp. 237-243, 2001
12. Chang, C.G., Cui, J.J., Wang, D.W., Hu, K.Y.: Research on case adaptation techniques in case-based reasoning, In: Proceeding of the third international conference on Machine Learning and Cybernetics, Shanghai, pp. 26-29 August 2004.
13. MAK Technologies, MAK VR-Forces 3.7.1 User's Guide.
14. Rasch, R., Kott, A., Forbus, K.D.: Incorporating AI into military decision making: an experiment, Intelligent Systems, IEEE 18(4): 18 – 26, 2003.
15. White, G.: Private communication, 2005
16. Pratt, D.R.: Case based reasoning for the next generation synthetic force. Technical Report SAIC-01/7836&00, Science Applications International Corporation, Orlando, FL. 2001.
17. Hebb, D. O. The organization of behavior. Wiley, New York, 1949.