Case-Based Reasoning Adaptation for High Dimensional Solution Space

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

[1] Surrey DTC, 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 problem to solve new problems. Adaptation is one of the most difficult parts of CBR cycle, especially, when the solution space with multi-dimension. This paper discusses the adaptation of high dimensional solution space and proposes a possible approach for it. Visualisation induced Self Organising Map (ViSOM) is used to map the problem space and solution space first, then a BackPropagation (BP) network is applied to analyse the relations between these two maps. A simple military scenario is used as case study for evaluation.

1 Introduction

Case-Based Reasoning is a methodology that reuses the solutions of previous similar problems to solve the new problems. Adaptation is considered to be one of the most difficult parts of CBR cycle. Wilke and Bergmann classify adaptation into three main types [1]: 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. There 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 adaptation could be combined. A general framework of case adaptation is proposed [13], 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

R.O. Weber and M.M. Richter (Eds.): ICCBR 2007, LNAI 4626, pp. 149–163, 2007.

to overcome the difficulties and limitation of rule-based adaptation, Leake[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 arise, the adaptation case are utilised to estimate the correctness of the proposed solution and suggest the necessary adjustments.

The above mentioned methods are referred as "knowledge-light methods" [5], learn adaptation knowledge from the CBR system's own cases and treat 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 can badly affect its performance. Furthermore, the 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, the classification of a case etc. But in our project, CBR is applied to find the suitable course of action (COA) for the military scenario. A COA is represented by the entities' waypoints at the 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. That is why we propose another approach to solve case adaptation in this situation.

The rest of the paper is organised as follows. Section 2 discusses possible normal adaptation approaches. Section 3 introduces 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, our discussion and conclusions in section 7.

2 Normal Adaptation Approach

For our system, the easy 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, which are also based on the 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 case in our project. In particular, we set up a three layers BP network. The network is trained by using as input the problem space differences between all pairs of cases, while the solution space differences between the corresponding cases are the target output for each pair. For example, suppose 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, in this way we could 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 must be complicated and a large size of cases are required to train the neural network. This is particularly difficult for our project, given only small size training samples we have. To solve this problem, we propose to employ SOM to the case problem space and the solution space first to reduce the size of the BP network. SOM can dramatically reduce the data dimensionality, and help us to visualise the case base as well.

3 SOM and ViSOM

The SOM is an unsupervised neural network algorithm that has been successfully used in a wide variety of applications, such as pattern recognition, image analysis and fault diagnosis etc. The basic algorithm was inspired by the way in which various human sensory impressions are neurologically mapped into the brain such that spatial or other relations among stimuli correspond to spatial relations among the neurons. This is called competitive learning. A SOM consists of two layers of neurons, an input layer with n input nodes, which are according to the n –fold dimensionality of the input vectors, and N output nodes, which are according to the N decision regions, with every input node 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 dot product or the Euclidean criterion as:

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

The smaller the Euclidean distance is, the closer the vectors are. Another measure of similarity is base 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.

The bigger the cosine is, the similar the vectors are. So we can find the best-matching winner $i(x)$ at time t:

$$i(x) = \arg\min_j \left\| x(t) - w_j \right\|, \qquad j = 1,2,\ldots,N \tag{3}$$

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

$$w_j(t+1) = \begin{cases} w_j(t) + \eta(t)\left[x(t) - w_j(t)\right], & j \in \Lambda_{i(x)}(t) \\ w_j(t) & otherwise \end{cases} \tag{4}$$

Where $\eta(t)$ is the learning rate and is the neighborhood function around the winner, the winning unit and its neighbors adapt 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 neuron, and so on until at some distance d the weight adjustment go to zero, all these can be implemented by Gaussian function,

$$\Lambda_{i(x)}(t) = \exp\left(\frac{-d(j)^2}{2\sigma(t)^2}\right) \tag{5}$$

Where σ^2 is the variance parameter specifying the spread of the Gaussian function, and it is decreasing as the training progresses.

In order to speed up the computation, Chef Hat function can be used, in which only identical positive weight changes are made to those neurons within the r radius. Continue with these above steps until no noticeable changes.

4. Continuation. Continue with step 2, both $\eta(t)$ and $\Lambda_{i(x)}(t)$ are dynamically during the training, until no noticeable changes are observed. At the beginning of the learning process the radius of the neighborhood is fairly large, but it is made to shrink during learning. This ensures that the global order is obtained already 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 for obtaining the best project result. The average quantization error is the average distances between the vector data to their prototypes. Generally, when the size of the map increase, there are more unit 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. Topographic product introduced by Bauer and Pawelzik [6], try to find folds on the maps. Since the SOM approximates the higher dimension of input space by folding itself, topographic product can be an indicator about topographic error. However, it does not differentiate the correct folds following a folded input space and the actually erroneous folds. Kohonen himself proposed another approach to measure the proportion of all data vectors whose first and second best matching unit are not adjacent [7]. This is called topographic error. The smaller the topographic error is, the better the SOM preserves the topology. Generally, the higher dimensionality of input space has, the larger the topographic error is. This is because the increasing difficulty to project units in right order and the dimensionality of the prototype grows too.

There have been some research efforts to enhance topology preservation of SOM. In [8], SOM was trained to minimize the quantization error first, and then minimise the topological error in the second stage. A Double SOM (DSOM) uses dynamic grid structure instead of a static structure, together with the classic SOM learning rules to learn the grid structure of 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 structures 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 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 provide.

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 ViSOM are set up, the location of cases on the case problem space ViSOM can be used as input while the location of corresponding case on the case solution space ViSOM as output. Because these locations are only two dimensional, the BP network structure is much simpler than 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 weight as the connections.

Instead of using the actual location, an alternative approach is to employ, as input, the location's difference between each case pair of case problem space ViSOM. Likewise, the location's difference between the same case pair of case solution ViSOM is the output.The difference between target case and its nearest case is input to the trained network after the network is trained, in this way the target case location on 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 has 2 elements. There are 5 neurons in the hidden layer while 2 neurons in the output layer. The transfer function for both layers is tan-sigmoid. The training function is trainlm, the Levenberg-Marquardt algorithm, a very fast training method which is suitable for small network.

4 How to Find the Target Case Solution

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

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

Third, an approach called kriging [12] which usually get better interpolation result can be used. Kriging is the best linear unbiased estimator. The estimation of an unsampled location is given as the weighted sum of the circumjacent observed pointes. It is unbiased since it tries to have the mean residual or error equal to 0, it is best because it aims at minimizing the variance of the errors. It is a powerful spatial interpolation technique and widely used throughout the earth and environment sciences. Kriging is normally suitable for 2 or 3 dimensional data, so it can not be applied directly on the original dataset. However, our case solution space map is only two dimensional now.

In general, in order to solve the problem we must model the covariance matrix of the random variable which was done by choosing covariance function and calculating all the required co variances form the function by modelling the "variogram" of the function. The variogram $\gamma(h)$ is defined as

$$\gamma(h) = 0.5E[Z(x+h) - Z(x)]^2 \tag{8}$$

Where x and $x+h$ are points in the n (n<4) dimension. For a fixed angel, the variogram indicates how different the values becomes as the distance increases. When the angle is changed, the variograms disclose direction features such as anisotropy [12].

In order to calculate experimental variogram of a solution space with m dimensions, for each dimension, $S_1, S_2, ..., S_m$, a variogram will be created as a function of the distance in the case solution space. Such as:

$$\gamma_1(h) = \frac{1}{2N(h)} \sum_{i=1}^{N(h)} (S_{1\alpha}^i - S_{1\beta}^i)^2 \tag{9}$$

Where $\gamma_1(h)$ is the variogram of first dimension of the solution.

$N(h)$ is the total number of pairs of cases in the solution space map which are separated by a distance h.

$S_{1\alpha}^i$ is the first parameter of solution of case α

$S_{1\beta}^i$ is the first parameter of solution of case β .

case α is h away case β in the case solution space.

Similarly, for the second dimension of case solution, we have

$$\gamma_2(h) = \frac{1}{2N(h)} \sum_{i=1}^{N(h)} (S_{2\alpha}^i - S_{2\beta}^i)^2 \tag{10}$$

And so on.

Actually, Kriging can be used directly on the case problem space map as well, instead of the solution space map. The same varigram formula can be used. The only difference is, this time h is the distance in the problem space map, and $N(h)$ is the

total number of pairs of cases in the problem space map which are separated by a distance h.

5 Experiment Design

In our research, we apply CBR to military decision making. When the military commanders face a tactical mission, they will develop a set of COAs to achieve their objective. Generally, it 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 understand. We choose MAK VR-Forces [14] 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 T-80 tank, BTR-80 combat vehicle, 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 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 hasty, attack deliberately, drawback, remain stationary or occupy), it boils down to a series of movements of the entities in the VR-Forces. So a COA is represented by a series of positions of entities. Once the entities reach the suitable waypoints, they fire automatically. Therefore, we decide 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 [15] which human commanders apply to decide 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 by Fig.1, plays out as:

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 Red platoon. When 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 made up of war stories, prior experience, tactics and doctrine [17]. According to domain knowledge, METT-T (Mission, Enemy, Terrain, Troops, and Time) are factors human commanders usually consider in the real battle field. It is very convenient to use these parameters to represent a scenario.

In this simple scenario, the mission, namely to breach the minefield, is fixed. In order to simplify the scenario representation, we currently omit mission, but will include it in the future when it is necessary.

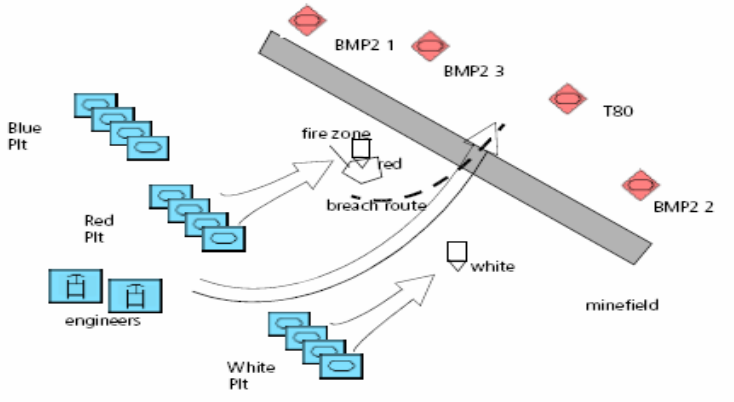

Fig. 1. Breaching Scenario

Each entity may be represented as a symbolic object, but comparing the effectiveness of the opposing sides then is 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, T80 is assigned power 5 while M1A2 is assigned power 6, and BMP2 is assigned power 3 [16].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 the entity is very powerful, it is useless when it is far away from the target) and 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 this consideration. 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 ½. Then the scaled strength ratio is defined as:

$$\text{Scaled Strength Ratio} = \frac{\sum_{i=1}^{n} T_i C_{ti}}{\sum_{j=1}^{m} E_j C_{ej}} \tag{11}$$

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 battle field is covered by a grid. Instead of using long complicated geocentric locations, we can use the grid information to represent an

entity's location. The entities which are 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, assign 0 to it. Otherwise, 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)

Table 1. Grid representation

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

There are two kinds of enemy entities: BMP2 and T80. The numbers of each type of entity may vary. Suppose we represent BMP2 as 1 and T80 as 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 can be stored according to the coordinate system used. For example, the entities in Table 1 can be represented by:

a= (3, 2)
b= (5, 4)

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 can 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. 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, COA can 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 at 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 de 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 choose values for the Scaled Strength Ratio, enemy entities locations and friendly troops entities' locations

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 $(Xe^1, Ye^1), (Xe^2, Ye^2), (Xe^3, Ye^3)$

to generate the case description part for 300 cases. Then according to each of the case description, we choose a suitable COA based on common sense and simulate it in VR-Forces. We record the result, including factors such as whether the goal was achieved or not, the enemy leftover power, the friendly troops leftover power, etc. The COA with the highest winning value *W*, which is a weighted function of these factors, will be chosen as the suitable solution.

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

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 approach is based on domain experts, such as military Subject Matter Experts (SME). We can 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 are 300 cases in our case base.We can divide the case base into two groups: one for training, and the other one for evaluation. Cases in the training group are input to the system for training the ViSOM and the BP networks while cases in the evaluation group will be used to judge how good the outputs are.

In Table 4, normalised average errors for cases in evaluation group are calculated, by comparing the output with the real solution in the case base. Loc-KNN represents using the location of map to train BP, with KNN to acquire the solution. Loc-Proto

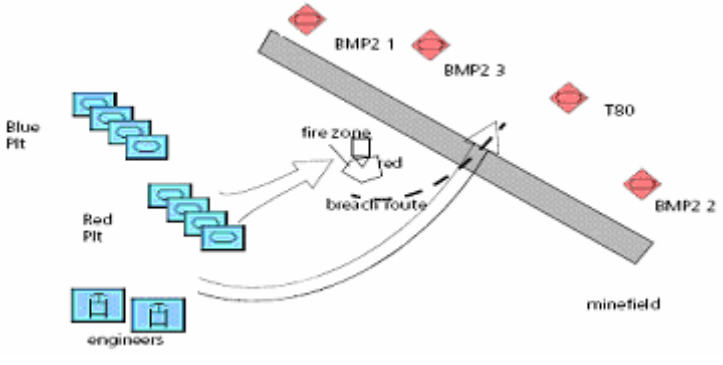

Fig. 2. Breaching exercise variation 1 (White platoon is missing)

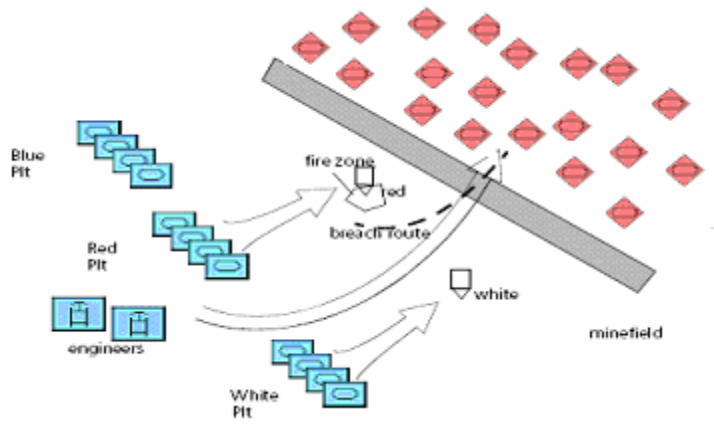

Fig. 3. Breaching exercise variation 2 (Enemy firepower is stronger than that of the friendly troops)

represents using the location of map to train BP, with map prototype vector for the solution. Loc-Krig represents using the location of map to train BP, with Kriging to acquire the solution.

Dif-KNN represents using the location's difference to train BP, with KNN to acquire the solution. Dif-Proto represents using the location's difference to train BP, with map prototype vector for the solution. Dif-Krig represents using the location's difference to train BP, with Kriging to acquire the solution. Kriging represents using kriging directly on the case problem space.

When Kriging is applied, we found this dataset is locally stationary, once we limit the neighbour size to 5, instead of using the whole case problem space to create the variogram, the result become better. The best result is obtained by using case pair location difference to train the BP and chose the prototype vector as the suggested solution.

We can also apply the *k*-fold cross-validation method [18] to reduce the possible bias which might be caused by the small sample. Table 5 shows the result of *k*-fold cross-validation, where *k* is 10 for Dif-Proto. The output COAs of cases in the evaluation group are often not exactly the same as the corresponding COAs stored in the case base. This is because for a given scenario, there is not only one successful COA. If all the waypoints in the suggested COA are reasonable, we think this COA is good, if only one or two waypoints deviate, we think the COA is satisfactory. Otherwise the COA is considered as an error.

Table 4. Experiment Results for all the methods discussed

Method	Loc-KNN	Loc-Proto	Loc-Krig	Dif-KNN	Dif-Proto	Dif-Krig	Kriging
Avg Err	0.435	0.458	0.447	0.421	0.393	0.407	0.396

Table 5. *k*-fold cross-validation ($k = 10$) for Dif-Proto

	Test1	*Test2*	*Test3*	*Test4*	*Test5*	*Test6*	*Test7*	*Test8*	*Test9*	*Test10*
Good	23	21	22	23	25	24	23	19	26	24
Satisfactory	5	8	3	5	1	4	4	7	2	3
Error	2	1	5	2	4	2	3	4	2	3

7 Conclusion

In this paper, we discuss how to achieve case adaptation for case base with high dimensional solution space. It is a very difficult task, esp. for high dimensional data and limited size of case base. We propose to map problem space and solution space in two different ViSOM. Then analyse the mapping between these two maps. A simple military scenario is used as a test. Although all the case attributes have numeric values in our example dataset. In fact, non-numeric attributes can be converted to numeric first. Thus our approach has the potential to be applied to other datasets as well.

Military application is very demanding area, but CBR is very suitable to mimic decision making process of human commander, thus using CBR to suggest possible COA for military scenario is reasonable. For this simple scenario, the result is promising. However it is still an initial endeavours, further effort need to be paid.

In order to evaluate the generality of this approach, we are looking for other simple direct datasets with high dimensional solution space. Once suitable dataset is available, additional experiments will be processed and further discussed.

References

1. Wilke, W., Bergmann, R.: Techniques and knowledge used for adaptation during case-based problem solving. In: Mira, J.M., Moonis, A., de Pobil, A.P. (eds.) Tasks and Methods in Applied Artificial Intelligence. LNCS, vol. 1416, pp. 497–505. Springer, Heidelberg (1998)
2. Leake, B., Kinley, A., Wilson, D.: Acquiring case adaptation knowledge: a hybrid approach. In: 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. In: Proceedings of the Seventeenth IJCAI Conference, pp. 1011–1016. Morgan Kaufmann, San Mateo, CA (2001)
4. Hanney, K., Keane, M.T.: The adaptation knowledge: how to easy it by learning from cases. In: Leake, D.B., Plaza, E. (eds.) Case-Based Reasoning Research and Development. LNCS, vol. 1266, pp. 359–370. Springer, Heidelberg (1997)
5. Wilke, W., Vollrath, I., Althoff, K.D., Bergmann, R.: A framework for learning adaptation knowledge based on knowledge light approaches. In: 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 3(4), 570–579 (1992)

7. Kohonen, T.: Self-Organizing Maps, 3rd edn. Springer Series in Information Sciences, vol. 30. Springer, 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, IEEE Service Center, 2000, vol.4, 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), 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, 157–173 (2004)
11. Yin, H.: ViSOM-A Novel Method for Multivariate Data Projection and Structure Visualization. IEEE Transactions on Neural Networks 13(1), 237–243 (2001)
12. Armstrong, M.: Basic Linear geostatistics, pp. 25–57. Springer, Heidelberg (1998)
13. 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)
14. MAK Technologies, MAK VR-Forces 3.7.1 User's Guide
15. Rasch, R., Kott, A., Forbus, K.D.: Incorporating AI into military decision making: an experiment. Intelligent Systems, IEEE 18(4), 18–26 (2003)
16. White, G.: Private communication (2005)
17. 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)
18. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth, Belmont, CA (1984)