Decision Support from Learning Multiple Boundaries on Military Operational Plans from Simulation Data*

Johan Schubert
Department of Decision Support Systems
Division of Information and Aeronautical Systems
Swedish Defence Research Agency
SE-164 90 Stockholm, Sweden
johan.schubert@foi.se
http://www.foi.se/fusion/

Anna Linderhed
Department of Sensor Informatics
Division of Sensor and EW Systems
Swedish Defence Research Agency
P.O. Box 1165
SE-581 11 Linköping, Sweden
anna.linderhed@foi.se

Abstract-**In this paper we provide decision support regarding robustness during execution of a military operational plan. We learn boundaries from simulated data from alternative plan instances of an expeditionary operation beyond which drastic changes can occur. These are boundaries that an operation must not move beyond without risk of failure. We receive simulated and evaluated plan instances from a simulation-based decision support system. These patterns are clustered by an unsupervised neural Potts spin clustering method into clusters where the instances in each cluster have similar characteristics and outcomes. This gives all plans a classification. We use a belief function based model screening method where all actions of the plan are evaluated as to their differentiating capacity between the two sets of plan instances. All plan instances are projected from their full representation to a subset of actions with high differentiating capacity. We apply supervised learning by support vector machine using the previous classification to learn support vectors for all pair of clusters given the reduced plans from the model screening. From these support vectors we derive a lower dimension hyperplane. One hyperplane between each pair of clusters will make up a full set of boundaries for this operational plan. We provide decision support during execution of an operational plan by continuously calculating the distance from the plan to the closest hyperplane step-by-step as action-by-action is being executed. This way a commander may observe if the operation is approaching a boundary during execution of the plan, beyond which it should not move without risk of failure.**

Keywords-military operational planning; effects-based planning; indicators; clustering; neural network; Potts spin; Dempster-Shafer theory; factor screening, support vector machine; hyperplane; data analysis; big data analytics; decision support.

I. INTRODUCTION

In this paper we provide decision support regarding robustness during execution of a military operational plan. We learn boundaries from simulated data from alternative plan instances of an expeditionary operation, beyond which drastic changes can occur. This is performed in a simulation-based decision support system through an event-based simulation that model plans according to the effects-based planning (EBP) approach [1–3]. We model the plan and evaluate alternative plan instance on how well they are able to drive the entire state of the simulation model, simulating a large set of actors, towards a predetermined end state. These plan instances are evaluated as to their performance and clustered by neural Potts spin clustering [4, 5] into clusters where all plan instances have both common characteristics and outcomes [6, 7].

We extend a methodology for learning the boundaries between single pairs of clusters [8] to manage any number of clusters. These boundaries are represented as high dimensional hyperplanes. We use a support vector machine (SVM) [9, 10] that learn support vectors for all clusters and we derive all hyperplanes from the support vectors.

In order to reduce the dimensionality of the hyperplanes we use Dempster-Shafer theory [11–13] for model screening of all actions of the plan. The idea is to find subsets of action alternatives that partition the set of all alternative ways to perform the action. This is done individually for each pair of clusters, with the purpose of finding those actions of the plan with the highest average discriminating capacity between all clusters.

We provide decision support during execution of a plan by calculating the distance from the plan to the closest hyperplane step-by-step as action-by-action is being executed. By visualizing the change in distance during execution a commander may observe if the operation is approaching a boundary beyond which outcomes may be uncertain.

In section II we introduce EBP. In section III we describe the scenario of an expeditionary operation. In section IV we describe event-based simulation of military operational plans and the A*-search through simulation increments of alternative plan instances. In section V we present a method for clustering all simulated and evaluated plans into clusters with common characteristics and outcomes. In section VI we perform an information fusion based model screening of all actions of the plan in order to find the actions with the highest differentiating capacity between the plans of different clusters. In section VII we use an SVM to learn support vectors of each pair of clusters using the reduced plan instances. From these vectors we derive low dimensionality hyperplanes that work as boundaries between military plans of different characteristics and outcomes of different clusters. The implementation of SVM is described in section VIII. In section IX we elaborate on the

*This work was supported by the FOI research project "Real-Time Simulation Supporting Effects-Based Planning", which is funded by the R&D programme of the Swedish Armed Forces.

usage of hyperplanes and demonstrate how the distance from a plan during execution to the boundary it should not move beyond can be visualized for decision support. When a plan is at or close to the boundary an unexpected outcome may push the plan over that boundary. Finally, in section X conclusions are drawn.

II. Effects-based Planning

How we model a phenomenon depends on the purpose of the model and the questions we want to answer. Since our simulation system aims to support decision-making within an effects-based approach to operations (EBAO) [14, 15] the modeling has to be based on EBAO and the concepts used within it, such as plan, action, effect, end state, etc.

EBAO is a military approach to the management and implementation of efforts at the operational level. According to the United States Joint Forces Command (USJFCOM) EBAO are "operations that are planned, executed, assessed, and adapted based on a holistic understanding of the operational environment in order to influence or change system behavior or capabilities using the integrated application of selected instruments of power to achieve directed policy aims" [16].

Within the framework of EBAO, EBP is a method for developing objectives and effects to be achieved through a series of synchronized actions within a military operational plan, conceptually developed starting top-down from a desired end state.

III. Bogaland Scenario

We make use of the same scenario that has regularly been used by the Swedish Armed Forces in the Combined Joint Staff Exercises. The scenario comprises several fictitious countries, two of which, Xland and Bogaland, have been described in-depth. Background histories offer explanations to why and how sentiments, stances, identities, loyalties, economic dependencies and inequalities have evolved over time, occasionally resulting in shifts of power. Phenomena that are commonly found in conflict areas and post conflict areas have been embedded in scenario contexts that make the origins of the phenomena plausible, Fig. 1.

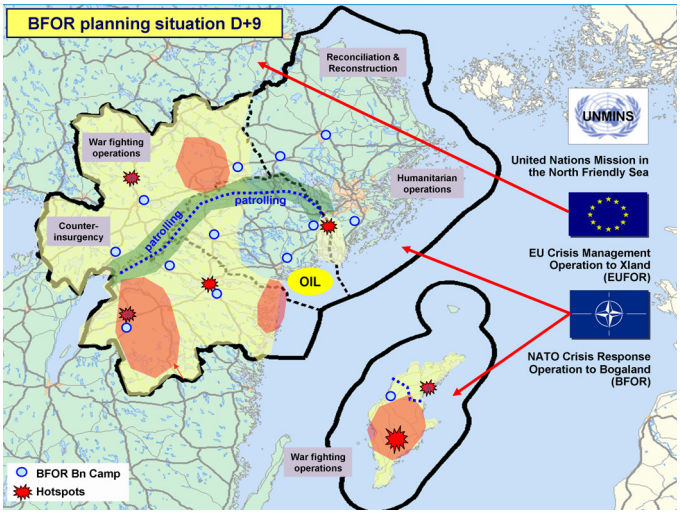


Fig. 1. The Bogaland scenario.

In Bogaland, a newly industrialized country, a civil war broke out ten years ago when discontent within the minority ethnic-religious group had reached very high levels. The root cause was increasing social stratification caused by what members of the minority group perceived as unjust distribution of revenues from a natural resource located in an area populated by the minority group. The civil war put an end to the exploitation of the resource, in this case oil, and revenues dropped to very low levels. The country was split into two parts, roughly along ethnic lines, with each part having its own government. A post-war economy evolved over the next decade, and several irregulars and insurgents are now challenging the incumbent presidents.

The incumbent presidents have signed a peace-agreement, and an international force, BFOR, is present to support the implementation of the agreement. Irregular groups in Bogaland seek to preserve or increase their influence by undermining the efforts of BFOR, the governments or competing irregulars.

IV. Event-based Simulation of Plans Using A*

A. Plans

A *plan* as it is defined in the context of EBAO is a sequence of *actions* that together leads to a desired end state which is set by a military force.

A typical plan instance P_1 in our scenario of study is

[1 2 41 61 43 108 20 8 21 12 0 62 64 50
60 52 18 25 63 53 55 56 70 78 67 82 90
79 80 97 0 95 96 104 106 29 30 0 0 0 58
46 47 3215.1 2422.0 793.1]

where all but the three last numbers in this sequence are the numbers of the selected alternative for each action in this plan instance. For example, action number 3 (i.e., position 3 in the sequence) takes alternative number 41. Note, that alternatives for different actions are numbered with running numbers in no particular order. The three last parameters are different evaluation measures called f, g and h. They are distance measures calculated from changes in the scenario state and used in an A*-search algorithm. If all possible plan instances are represented in a tree, a full plan instance is a path from the root of the tree down to one particular leaf. Obviously, the depth of the tree is the length of the sequence minus three (i.e., not counting the f, g and h estimates). Plan instance P_1 above corresponds to a sequence of 43 specific simulations for the 43 actions where the actions take the numbered alternative listed in the sequence as its input parameter [2]. A few actions have value "0". These are actions not performed in this particular plan instance.

B. Simulating action alternatives

The scenario consists of participating actors, their initial states and probability distribution for different actions, environmental data, as well as the plan that is to be evaluated.

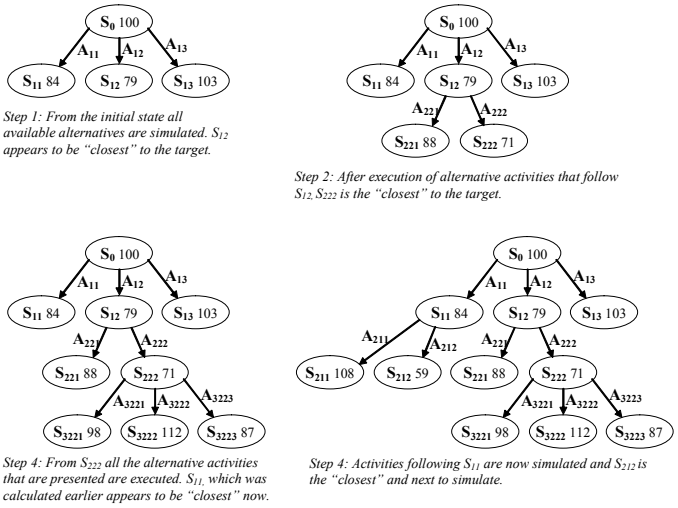
The system state S_n is defined as the combination of all actors' state parameters. Consider action A_n. It transforms system state S_{n-1} according to $S_n = f(S_{n-1}, A_n)$. The implementation of A_n is an interaction between our own action,

other actors' agendas and response operations, and other external events. Hence, our function $f(S_{n-1}, A_n)$ is designed as an event-driven simulation model in order to manage the complex interactions in a transparent manner [17].

We know that the goal of the simulation is to execute different plans and identify those plans that result in system states that are closest to our end state, i.e., has the shortest distance to it.

C. Searching among action alternatives

To find good combinations of alternatives for all actions of the plan we apply A*-search. It means that, on the basis of a given system state, we simulate the effect of each alternative action in our plan, but only one step at the time. Doing so, for every alternative, we get a new system state whose distance to the desired end state is calculated. Given the alternative that is best, i.e., closest to our end state, we simulate possible subsequent alternative actions provided, but again only one step ahead in our action/event list. One of these alternatives leads to a condition that is closer than the others. However, it is possible that all the alternatives actually lead away from the target as seen by Fig. 2.



Step 1: From the initial state all available alternatives are simulated. S_{12} appears to be "closest" to the target.

Step 2: After execution of alternative activities that follow S_{12}, S_{222} is the "closest" to the target.

Step 4: From S_{222} all the alternative activities that are presented are executed. S_{11}, which was calculated earlier appears to be "closest" now.

Step 4: Activities following S_{11} are now simulated and S_{212} is the "closest" and next to simulate.

Fig. 2. An example illustrating the four first steps in a simulation of a plan starting with initial system state S_0 with the *distance* of 100 to the desired end state.

Therefore, we must also compare the new distance with the best of the distances that have been simulated and recorded in the previous simulation steps, but then had opted out in favor of a better sequence of alternative actions. The best sequence now becomes the basis for the next simulation step.

During simulation an assessment is made of how well each action is performed. This is done by the functions g and h. Function g measures the consequence of all performed actions as a distance from the initial state $S_{0,0}$ to the current simulated state S_{x,y_x} action-by-action [1, 2]. We have,

$$g(y_x) = \sum_{i=0}^{x-1} \Delta(S_{i,y_i}, S_{i+1,y_{i+1}}) . \quad (1)$$

Function h is a heuristic estimate of the remaining distance from S_{x,y_x} to the end state.

The estimated distance from the current state to the end state is given by

$$h(y_x) = \Delta(S_{x,y_x}, S_e) . \quad (2)$$

With the total weighted estimated distance from the initial state to the end state via the current state S_{x,y_x} is

$$f(y_x) = g(y_x) + 80h(y_x) . \quad (3)$$

This is the distance function that is minimized by A*. The weight "80" was derived by experimentation to balance the performance of minimizing g and h and is domain dependent.

In an experiment performed, we used a full size operational plan of 43 actions and simulated 10 000 different plan instances. The total computation time was 66.5 hours using a single core processor. This problem scales well for parallel computing as we have a Monte Carlo loop within the event-based simulation that is easy to compute in parallel.

V. Clustering Simulated Plan Instances

We cluster the patterns of plan instances that are similar in structure and consequences. Similar in structure means that they have more or less carried out similar alternative actions. Similar in consequences means that they travel on average the same distance action-by-action towards the end state.

We observe the difference in consequences between two plans. We compare the difference in the incremental changes of g and h called ΔG and ΔH, respectively, for each action A_k and both plans P_i and P_j as they progresses down the sequence of additional actions A_k. We have, for each A_k,

$$\Delta G(P_i.A_k, P_j.A_k) = \left|\Delta g(P_i.A_k) - \Delta g(P_j.A_k)\right| \quad (4)$$

and

$$\Delta H(P_i.A_k, P_j.A_k) = \left|\Delta h(P_i.A_k) - \Delta h(P_j.A_k)\right| \quad (5)$$

where

$$\Delta g(P_i.A_k) = g(P_i.A_k) - g(P_i.A_{k-1}) \quad (6)$$

and

$$\Delta h(P_i.A_k) = h(P_i.A_k) - h(P_i.A_{k-1}) , \quad (7)$$

and i is an index for different plan instances and k is the index for action.

Thus, $P_i.A_k$ is a variable referring to the kth action of the ith plan. It takes an integer as its value that is the number of the alternative for this action, e.g., $P_1.A_3 = 41$ imply that action number 3 of plan number 1 performs alternative number 41.

In addition, we need to measure the structural distance between two plans. This is done by the Hamming [18] distance *Ha* which measures the structural distance between P_i and P_j.

We have,

$$Ha(P_i.A_k, P_j.A_k) = \begin{cases} 0, P_i.A_k = P_j.A_k \\ 1, P_i.A_k \neq P_j.A_k \end{cases} \quad (8)$$

when both actions $P_i.A_k$ and $P_j.A_k$ exists within the simulated sequences P_i and P_j, otherwise 0 by definition.

Using this measure, we compare each action in two different plans to calculate the structural distance between the plans. For each action we observe the alternative chosen in both plans.

We put these three measures together into an interaction functions that measures the overall distance between plan P_i and P_j [2].

We have,

$$J_{ij}^{-} = 1 - \left[1 - \frac{1}{|A_k|}\sum_k Ha(P_i.A_k, P_j.A_k)\right] \times \left[1 - \frac{1}{|A_k|}\frac{\sum_k |\Delta g(P_i.A_k) - \Delta g(P_j.A_k)|}{\max_k\{|\Delta g(P_i.A_k) - \Delta g(P_j.A_k)|\}}\right] \times \left[1 - \frac{1}{|A_k|}\frac{\sum_k |\Delta h(P_i.A_k) - \Delta h(P_j.A_k)|}{\max_k\{|\Delta h(P_i.A_k) - \Delta h(P_j.A_k)|\}}\right]. \quad (9)$$

Here the sums of the second and third lines are normalized by the maximum difference, and all sums of the three lines are normalized by the number of actions of the plan. Thus, $J_{ij}^{-} \in [0, 1]$, and is "1" if one of the three measures is at maximum, and is "0" is all three measures are at minimum.

We partition the set of all simulated plans into clusters using the Potts spin model [4] in such a way as to minimize the overall sum of all interactions J_{ij}^{-} within each cluster.

The Potts spin problem consists of minimizing an energy function

$$E = \frac{1}{2}\sum_{i,j=1}^{N}\sum_{a=1}^{q} J_{ij}^{-} W_{ia} W_{ja} \quad (10)$$

by changing the states of the spins W_{ia}'s, where $W_{ia} \in \{0, 1\}$ and $W_{ia} = 1$ means that plan P_i is in cluster a. This model serves as a clustering method if J_{ij}^{-} is used as a penalty factor when plan P_i and P_j are in the same cluster.

For computational reasons we use a mean field model, where spins are deterministic with $V_{ia} = \langle W_{ia} \rangle$, $V_{ia} \in [0, 1]$, in order to find the minimum of the energy function. The Potts mean field equations are formulated [5] as

$$V_{ia} = \frac{e^{-H_{ia}[V]/T}}{\sum_{b=1}^{K} e^{-H_{ib}[V]/T}} \quad (11)$$

where

$$H_{ia}[V] = \sum_{j=1}^{N} J_{ij} V_{ja} - \gamma V_{ia} \quad (12)$$

and T is a parameter called the temperature that is used to control the influence of the interaction. This is a system parameter initialized to

$$\frac{1}{K} \cdot max(-\lambda_{min}, \lambda_{max}), \quad (13)$$

where K is the number of clusters, and λ_{min} and λ_{max} are the extreme eigenvalues of M, where

$$M_{ij} = J_{ij}^{-} - \gamma\delta_{ij}. \quad (14)$$

In order to minimize the energy function (11) and (12) are iterated until a stationary equilibrium state has been reached for each temperature. Then, the temperature is lowered step-by-step by a constant factor until $\forall i, a.\ V_{ia} = \{0, 1\}$ in the stationary equilibrium state, Fig. 3 [6, 7].

VI. Evidential Model Screening of Actions

In this section we investigate which actions of the plan have most differentiating capacity for each pair of clusters using Dempster-Shafer theory. These are the actions that should be part of an indicator projected from $(\mathbb{Z}^{+})^{|\{A_k\}|-1}$ to a lower dimension, onto the set of these actions. This will reduce, by the same factor, the dimensionality of the support vectors and hyperplanes that are learned from all plan instances of reduced dimensionality (section VII) with only the most differentiating actions remaining.

D. Maximum differentiating capacity

The most differentiating actions are found by investigating the maximum differentiating capacity for one action A_k of two disjoint subsets of the frame of discernment $\Theta_k = \{P_i.A_k|\chi_j, A_k\}_i$ one for each cluster, i.e., the set of possible values of A_k over all clusters χ_j, where i, j and k are indices for different plan instances, clusters and actions, respectively. Note that Θ_k is an element from $\{\Theta_k\}_k$ that is not dependent on cluster, but varies for each action A_k.

We develop a method, which for each cluster χ_j calculate histograms for all actions A_k over all plan instances that we receive from the simulation-based decision support system.

From all plan instances P_i in each cluster χ_j we build the histogram over all alternatives for A_k.

INITIALIZE

K (number of clusters); N (number of plans);

J_{ij}^{-} $\forall i, j$;

$s = 0$; $t = 0$; $\varepsilon = 0.001$; $\tau = 0.9$; $\gamma = 0.5$;

$T^0 = T_c$ (a critical temperature) $= \frac{1}{K} \cdot max(-\lambda_{min}, \lambda_{max})$, where λ_{min} and λ_{max} are the extreme eigenvalues of M,

where $M_{ij} = J_{ij}^{-} - \gamma\delta_{ij}$;

$V_{ia}^0 = \frac{1}{K} + \varepsilon \cdot rand[0,1]$ $\forall i, a$;

REPEAT

• REPEAT–2

$\forall i$ Do:

• $H_{ia}^s = \sum_{j=1}^{N} J_{ij}^{-} V_{ja}^{\begin{cases} s+1, & j < i \\ s, & j \geq i \end{cases}} - \gamma V_{ia}^s$ $\forall a$;

• $F_i^s = \sum_{a=1}^{K} e^{-H_{ia}^s / T^t}$;

• $V_{ia}^{s+1} = \frac{e^{-H_{ia}^s / T^t}}{F_i^s} + \varepsilon \cdot rand[0,1]$ $\forall a$;

• $s = s + 1$;

UNTIL–2

$\frac{1}{N} \sum_{i,a} \left| V_{ia}^s - V_{ia}^{s-1} \right| \leq 0.01$;

• $T^{t+1} = \tau \cdot T^t$;

• $t = t + 1$;

UNTIL

$\frac{1}{N} \sum_{i,a} (V_{ia}^s)^2 \geq 0.99$;

RETURN

$\left\{ \chi_a \middle| \forall S_i \in \chi_a .\ \forall b \neq a\ V_{ia}^s > V_{ib}^s \right\}$;

Fig. 3. Cluster algorithm.

We have,

$$h_{\chi_j}^{A_k}(l) = \sum_i \begin{cases} 1, & P_i.A_k = l \\ 0, & P_i.A_k \neq l \end{cases} \qquad (15)$$

where $l \in \{P_i.A_k | \chi_j, A_k\}_i$ and the summation is made over all plans P_i in χ_j, and $l = 0$ is a missing value due to action A_k not being performed in P_i.

What we are looking for are actions where there are alternatives with very different frequencies for two clusters χ_i and χ_j, where some alternatives have much higher frequency for one cluster, and other alternatives have much higher frequency for the other cluster. When this is the case we have an action with high discriminating capacity.

However, our interest is in finding different subsets of alternatives with maximum differentiating capacity. In order to handle this situation we need to represent the histograms as basic belief assignments within Dempster-Shafer theory.

From each histogram we construct a basic belief assignment where the frequency of "0" is assigned to Θ_k. This is a mass function where all focal elements except one are singleton subsets of the frame $[\{l\}, m_{\chi_j}(\{l\})]$ (i.e., actions of the plan). The exception being the support of Θ_k $[\Theta_k, m_{\chi_j}(\Theta_k)]$ as the only non-singleton focal element.

For all subsets of Θ_k (i.e., regarding action A_k) we construct m_{χ_j} for χ_j. We get,

$$\begin{aligned} m_{\chi_j}^{A_k}(\{l\}) &= \frac{1}{|\{P_i\}|} \cdot h_{\chi_j}^{A_k}(l), \qquad l \in \{P_i.A_k | \chi_j, A_k\} \\ m_{\chi_j}^{A_k}(\Theta_k) &= 1 - \sum_{k=1}^{|\Theta_k|} m_{\chi_j}^{A_k}(\{k\}), \quad l = 0 \\ m_{\chi_j}^{A_k}(B) &= 0, \qquad 1 < |B| < |\Theta_k|, \end{aligned} \qquad (16)$$

where $|\{P_i\}|$ is the number of plan instances. Note, that all subsets B with cardinality $1 < |B| < |\Theta_k|$ receive zero support.

In order to evaluate the discriminating capacity of a particular action A_k for a pair of clusters χ_i and χ_j we investigate the separation of all disjoint subsets. We find the maximum separation for the two disjoint subsets where we measure the difference in belief in a subset X of two different belief functions from χ_i and χ_j, respectively, for A_k, and the difference in belief for another disjoint subset Y between χ_j and χ_i. Here, X and Y are disjoint subsets $X \cap Y = \varnothing$ of the frame of discernment where $X \cup Y \subseteq \Theta_k$, i.e., *not* necessarily $X \cup Y = \Theta_k$.

We calculate the discriminating capacity $DC(A_k)$ of action A_k as the maximum difference of belief between two clusters χ_i and χ_j for each of two disjoint subsets X and Y of the frame Θ_k. We have,

$$DC(A_k) = \max_{\substack{X, Y \subseteq \Theta_k \\ X \cap Y = \varnothing}} \left[\mathrm{Bel}_{\chi_i}(X) - \mathrm{Bel}_{\chi_j}(X) + \mathrm{Bel}_{\chi_j}(Y) - \mathrm{Bel}_{\chi_i}(Y) \right] \qquad (17)$$

where $0 \leq DC(A_k) \leq 2$. The maximum in (17) is found by evaluating $DC(A_k)$ for all $X, Y \subseteq \Theta_k$ where $X \cap Y = \varnothing$. This is of course a problem of exponential computational complexity, but easy to do since Θ_k is usually small; $1 \leq |\Theta_k| \leq 8$ in the example we study.

From $DC(A_k)$ we can calculate the average discriminating capacity $ADC(A_k)$ or each A_k over all pairs of clusters χ_i and χ_j. We have,

$$ADC(A_k) = \frac{1}{n(n-1)} \sum_{\chi_i} \sum_{\chi_j | j>i} DC(A_k) \qquad (18)$$

where $n = |\{\chi_i\}|$ is the number of clusters.

With this measure we can rank all actions of the plan as to their average discriminating capacity for all clusters. Using a threshold we can project all plan instances onto a smaller number of screened factors with high discriminating capacity.

In Fig. 4 we show the calculation of $ADC(A_k)$ by (18) for all 43 actions of the plan in the example. From this result we can select a subset of actions that has the highest discriminating capacity ranked by $ADC(A_k)$ as a lower dimension projection to be used for the SVM-learning of boundaries between clusters.

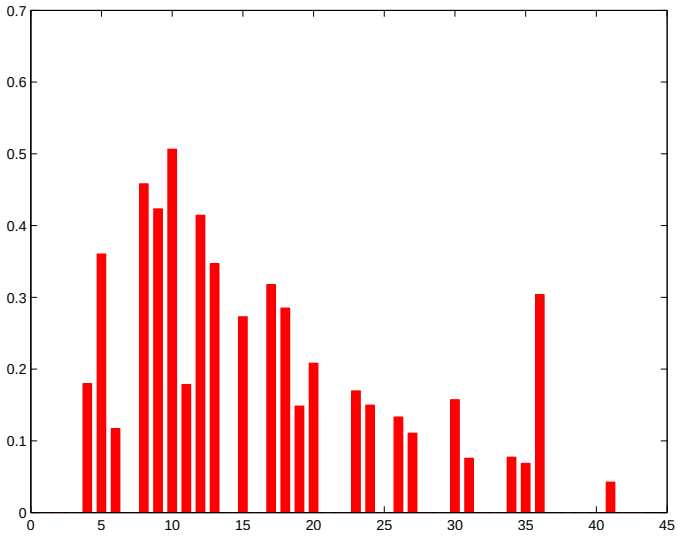


Fig. 4. Discriminating capacity for each action.

In our example we use a cut off of 0.25, focusing on the ten actions with the highest discriminating capacity of the 43 actions available in the plan; {A_5, A_8, A_9, A_{10}, A_{12}, A_{13}, A_{15}, A_{17}, A_{18}, A_{36}}. These are the ten actions that are best in discriminating between plans with different characteristics and outcomes that belong to different clusters.

In addition to the alternatives for all actions of plans, each plan instance also consist of the three real values (f, g, h) describing the consequence of the plan instance as evaluated by the simulation-based decision support system, these three values are always included in the projected plan instance.

VII. Learning Support Vectors and Hyperplanes

We summarize the information contained in a cluster by using a hyperplane created by an SVM. We are mainly interested in the distances from a chosen plan to its boundary with classes other than its own. Several stages are needed to achieve the result. First, we need to find the best way to represent the training data for use in the SVM, this includes normalization. Secondly, we must analyze the problem of finding optimal SVM-parameters and a kernel. Finally, we analyze the distances. An SVM analysis finds the hyperplane that is oriented so that the margin between the support vectors of different classes is maximized.

The concept of treating the objects to be classified as points in a high-dimensional space and finding a line that separates them is not unique to the SVM. The SVM, however, is different from other hyperplane-based classifiers in how the hyperplane is chosen. If we define the distance from the separating hyperplane to the nearest data point as the margin of the hyperplane, then the SVM selects the maximum margin separating hyperplane. Selecting this hyperplane maximizes the SVM's capability to calculate the correct classification of up to that time unseen plan instances. When representing the classification boundary by the SVM optimal hyperplane, each dimension has a bound for the corresponding action in the plan. Using the SVM decision function, each action can be evaluated by its presence in the tested plans presented to the decision function. This way we can correct our bad plans to be good plans by simply changing the bad actions.

The first step is to adapt the plans to the SVM machinery. SVM requires that each data instance is represented as a vector of real numbers. Let a plan contain R actions which can take any value representing a valid alternative for this action. We generate N number of R-dimensional vectors for training. The plans are clustered into different classes to be used as training targets y_i. Training plans are represented by vectors $\boldsymbol{x}_i = \{x_{i1}, \ldots, x_{il}\}$. The plan vectors $\boldsymbol{x}_i$ are all normalized. Scaling them before applying the SVM is very important. This is done to avoid that attributes in greater numeric ranges dominate those in smaller numeric ranges.

The basic idea of SVM is to find a function $f(x)$ that has the highest deviation from the actually obtained targets y_i for all training data $\{(x_1, y_1), \ldots, (x_l, y_l)\} \subset \boldsymbol{X} \times \boldsymbol{R}$, where $\boldsymbol{X}$ denotes the space of the input plans.

In the case of linear functions f, a separating hyperplane, written in terms of a weight vector $\boldsymbol{w}$ and a threshold b takes the form $f(x) = (\boldsymbol{x},\boldsymbol{w}) + b$ with $\boldsymbol{w} \in \boldsymbol{X}, b \in \boldsymbol{R}$, where $(,)$ denotes the dot product. We want to minimize the norm $\|\boldsymbol{w}\|^2 = (\boldsymbol{w},\boldsymbol{w})$ as shown in Fig. 5.

This can be formulated as a convex optimization problem:

Minimize

$$\frac{1}{2}\|\boldsymbol{w}\|^2, \qquad (19)$$

subject to

$$y_i - (\boldsymbol{x}_i,\boldsymbol{w}) - b \geq 1 \qquad i = 1, \ldots, l. \qquad (20)$$

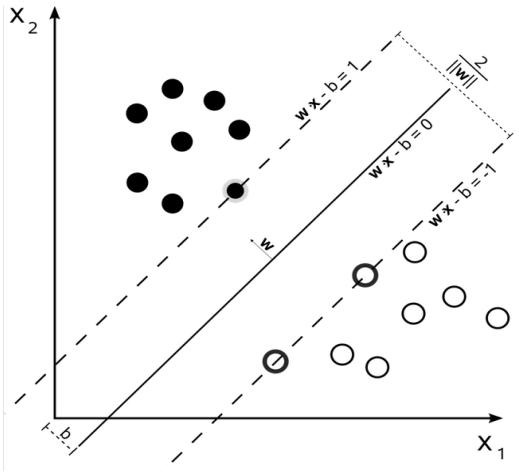


Fig. 5. Optimal linear divider of two separate classes.

The support vectors lie on the boundary of the convex hulls of the two classes, thus they possess supporting hyperplanes. The support vector optimal hyperplane is the hyperplane which lies in the middle of the two parallel supporting hyperplanes (of the two classes) with maximum distance. We have the decision function

$$sign(\boldsymbol{w}\boldsymbol{x} + b)\,. \tag{21}$$

The complexity of a function's representation by support vectors is independent of the dimensionality of the input space *X*, and depends only on the number of support vectors.

For all points from the hyperplane $HP[(\boldsymbol{x}_i,\boldsymbol{w}) + b = 0]$, the distance between the origin and the hyperplane *HP* is $b/\|\boldsymbol{w}\|$. This is the distance measure used.

Using the kernel trick [19] we can represent the decision function in higher dimensions. The choice of kernel used in this work is a Gaussian, which has a single parameter

$$\gamma = \frac{1}{2\sigma^2} \tag{22}$$

to be decided.

The accuracy of an SVM model is largely dependent on the selection of model parameters. Some flexibility in separating the categories is needed. SVM implementations have a cost parameter *C,* which controls the trade off between allowing training errors and forcing rigid margins. This parameter gives the model a soft margin that permits some misclassifications [10]. Increasing *C* increases the cost of misclassification of plans and forces a more accurate model to be crated. A search is used to find the optimal value of *C* and γ.

Using a hyperplane to separate the feature vectors into two classes works when there are only two target categories, but how do we handle the case where we have more than two classes? The two most used methods are: (*i*) "*one against many*" where each category is split out and all of the other categories are merged and (*ii*) "*one against one*" where $k(k-1)/2$ models are constructed where *k* is the number of categories. In this work we use the second approach and we evaluate classes against each other.

VIII. Implementation of SVM

We study an experiment of 1000 evaluated plan instances that are clustered by Potts spin clustering into eleven different clusters based on their characteristics and outcomes. Each action of the plan holds a unique integer number representing the alternative performed for that action. A training matrix of the 1000 different plans of length 46 is normalized with respect to each action. The eleven clusters are represented as classes which in turn are represented by any integer between 1 and 11.

We use the LIBSVM library [20] in this work. Important in LIBSVM is the kernel function and the choice of if its parameters. Parameter optimization is done by a full search out of a pre-defined parameter set. Cross validation is used for selection of best parameters for this training set, meaning that each combination of parameter choices is checked using cross validation, and the parameters with best cross-validation accuracy are chosen. Using the selected parameters the final model is trained on the whole training set. We use the optimal hyperplane defined by the SVM for determining the distance from any plan to the boundary of the classes for the other plans.

Since LIBSVM only delivers output for calculating the distance to the support vectors, the plans nearest the hyperplane of each class, we use an extra class for the plan under execution. This is (by definition) the only support vector of this class, and the distance from any specific plan of interest to the hyperplane can be calculated. Most interesting is how the distance for a specific plan under execution changes depending on how many of the actions has been performed. To be able to calculate this, the SVM needs to be re-trained for each new number of performed actions.

The primal variable $\boldsymbol{w}$ is not a direct output of LIBSVM. Instead we use the provided support vectors SVs and the coefficients for the support vectors sv_coef;

$$\boldsymbol{w} = \text{SVs} * \text{sv\_coef}. \tag{23}$$

The parameter *b* is a direct output from the trained model. It includes result from every combination of all classes. We are, however, only interested in the part of *b* regarding class 12 of the plan under execution. We need to select these by a selection vector for *b*.

The model is trained for twelve classes, eleven classes from pre-calculated Potts spin clustering and one class containing the plan under execution. Training is done 45 times for each investigation, each training with successive longer plans, from plan length of 2 actions to training on the full matrix with 43 actions, see TABLE I.

TABLE I: Pseudo code for the investigation.

1. Select the plan to be investigated and put it in a separate class. Update input label vector and selection vector for *b*.
2. For length of plan = 2 to 46:
   - 2.1 Select optimal parameters for training.
   - 2.2 Train the model.
   - 2.3 Calculate distances $b/\|\boldsymbol{w}\|$ using $\boldsymbol{w}$ = SVs * sv_coef.
3. End.
4. Plot distances.

IX. Using Hyperplanes as Decision Support

Single plans are tested against all the other plans and the result is plotted in Fig. 6–Fig. 11. The length of the plans is on the *x*-axis and the distances on the *y*-axis. The distances from the tested plan to the boundary of another class varies with the length of the plan. First, the best plan is tested against all the other plans; the best plan is the one with lowest value of *h*. The distance from the best *h*-plan to nearest hyperplane of all other classes using successive longer plans is shown in Fig. 6.

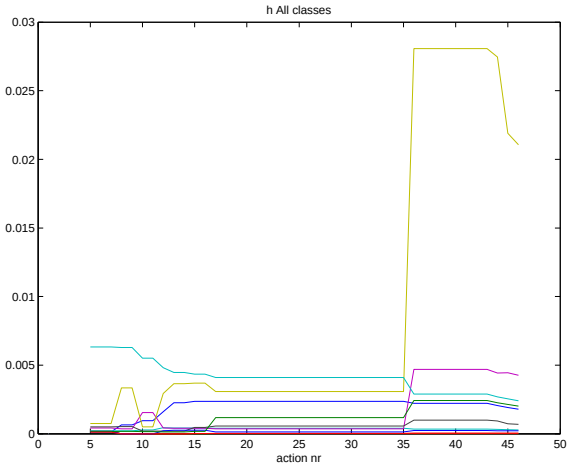


Fig. 6. Distance of the best plan (by measure *h*) during execution towards the eleven hyperplanes.

In Fig. 7 we show another view of the same result, by taking the minimum distance of all eleven classes in Fig. 6 at each length of plan.

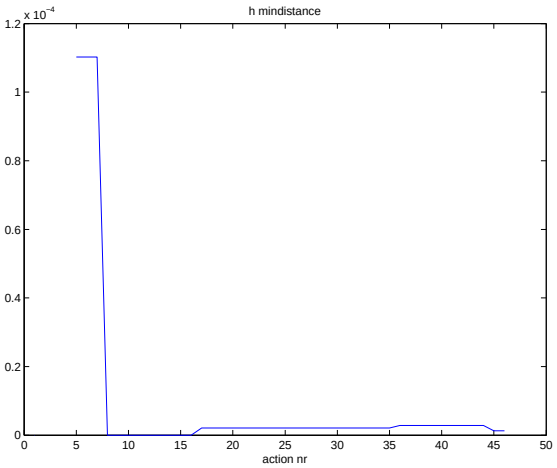


Fig. 7. Minimum distance of the best plan (by measure *h*) during execution to the closest hyperplane.

The same procedure is performed for the best plan by the measure *g*. The distance from the best *g*-plan to all other eleven classes using successive longer plans is shown in Fig. 8.

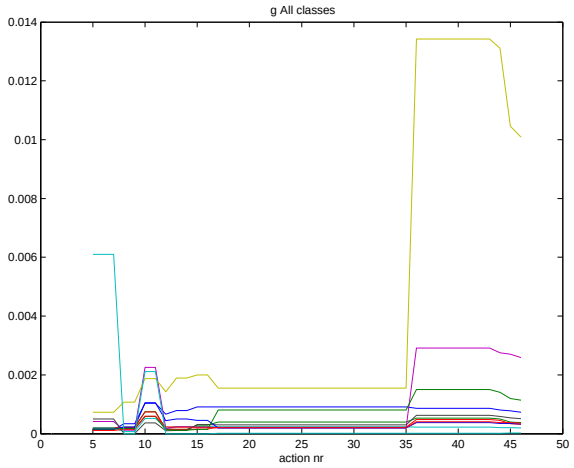


Fig. 8. Distance of the best plan (by measure *g*) during execution towards the eleven hyperplanes.

Taking the minimum distance of all classes in Fig. 8 at each length of plan gives us the result in Fig. 9.

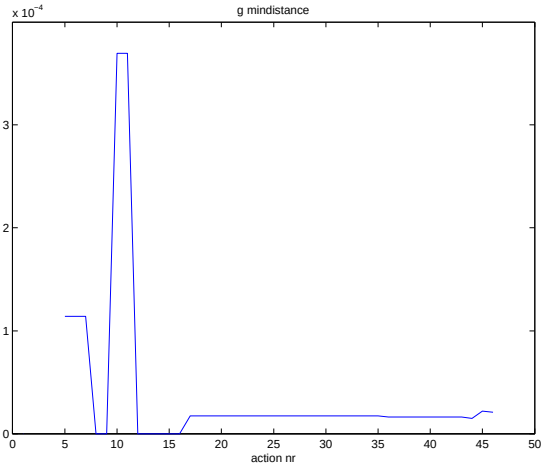


Fig. 9. Minimum distance of the best plan (by measure *g*) during execution to the closest hyperplane.

Since most of the plans are "good" we take a look at the ten best plans regarding *h*.

In Fig. 10, minimum distances created in the same way as in Fig. 7 are plotted for the ten best plans regarding *h*. The best plan is plotted in red for comparison.

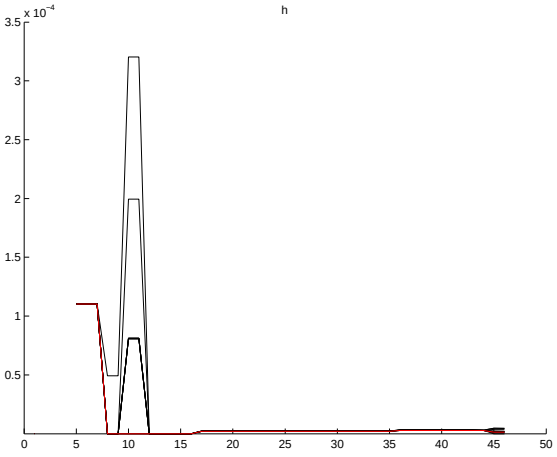


Fig. 10. Minimum distance for each of the ten best plans (by measure *h*) to the closest hyperplane during execution.

Fig. 11 is showing the same view as Fig. 10 but for the ten best plans regarding *g*.

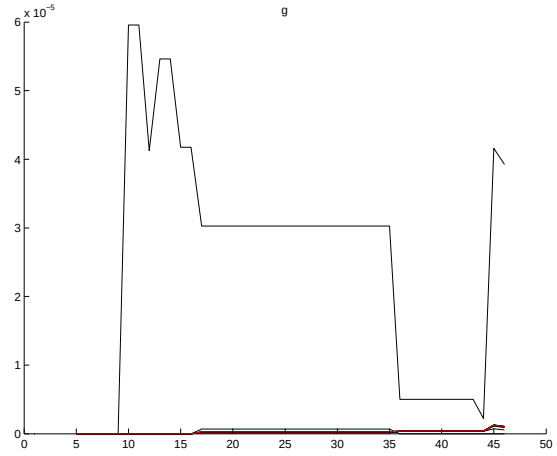


Fig. 11. Minimum distance for each of the ten best plans (by measure *g*) to the closest hyperplane during execution.

By using views as in Fig. 7 and Fig. 9 we may provide decision support during execution of a military operational plan. During the execution we observe in these figures the distance towards the closest (of eleven different) boundaries for the plan under execution as we progress down the sequence of actions.

In Fig. 6 and Fig. 8 the analyst observe a more refined view and may observe which other cluster of plans we might be approaching. The difference in outcomes by the current plan and the plans in the other cluster can then be observed by comparing the current plan with the best plan of that other cluster.

X. Conclusions

We conclude that it is possible to provide decision support regarding the robustness of military operational plans by learning boundaries from simulated and evaluated plan instances from a simulation-based decision support system.

Using a series of computational processing steps, such as calculating distances between all plans based on their characteristics and outcomes, clustering all plans with Potts spin neural clustering using those distances, performing evidential model screening of all actions as to their differentiating capacity between different clusters of plans, using SVM to learn support vectors from clusters of screened plan instances, and deriving hyperplanes from these support vectors, we provide decision support regarding the risk of failure as measured by the distance from a plan during execution towards the closest hyperplane.

XI. References

[1] J. Schubert, F. Moradi, H. Asadi, L. Luotsinen, E. Sjöberg, P. Hörling, A. Linderhed, F. Hinshaw, and D. Oskarsson, "Simulation-based decision support evaluating operational plans – Final report," Division of Information and Aeronautical Systems, Swedish Defence Research Agency, Stockholm, Tech. Rep. FOI-R--3635--SE, January 2013.

[2] J. Schubert, F. Moradi, H. Asadi, P. Hörling, and E. Sjöberg, "Simulation-based decision support for effects-based planning," in Proceedings of the 2010 IEEE International Conference on Systems, Man and Cybernetics, October 2010. Piscataway, NJ: IEEE, 2010, pp. 636–645.

[3] F. Moradi and J. Schubert, "Modelling a simulation-based decision support system for effects-based planning," in Proceedings of the NATO Symposium on Use of M&S in: Support to Operations, Irregular Warfare, Defence Against Terrorism and Coalition Tactical Force Integration (MSG-069), October 2009. Neuilly-sur-Seine: NATO Research and Technology Organisation, 2009, paper 11, pp. 1–14.

[4] F. Y. Wu, "The Potts model," Reviews of Modern Physics, vol. 54, pp. 235–268, January 1982.

[5] C. Peterson and B. Söderberg, "A new method for mapping optimization problems onto neural networks," International Journal of Neural Systems, vol. 1, pp. 3–22, May 1989.

[6] M. Bengtsson and J. Schubert, "Dempster-Shafer clustering using Potts spin mean field theory," Soft Computing, vol. 5, pp. 215–228, June 2001.

[7] J. Schubert, "Clustering belief functions based on attracting and conflicting metalevel evidence using Potts spin mean field theory," Information Fusion, vol. 5, pp. 309–318, December 2004.

[8] J. Schubert and A. Linderhed, "Learning boundaries on military operational plans from simulation data," in Proceedings of the 2011 IEEE International Conference on Systems, Man and Cybernetics, October 2011. Piscataway, NJ: IEEE, 2011, pp. 1325–1332.

[9] V. Vapnik, The Nature of Statistical Learning Theory. New York: Springer, 1995.

[10] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, pp. 273–297, September 1995.

[11] A. P. Dempster, "A generalization of Bayesian inference," Journal of the Royal Statistical Society Series B, Vol. 30, pp. 205–247, 1968.

[12] A. P. Dempster, "The Dempster-Shafer calculus for statisticians," International Journal of Approximate Reasoning, Vol. 48, pp. 365–377, June 2008.

[13] G. Shafer, A Mathematical Theory of Evidence. Princeton, NJ: Princeton University Press, 1976.

[14] E. A. Smith, Complexity, Networking, and Effects-based Approaches to Operations. Washington, DC: Department of Defense CCRP, 2006.

[15] J. P. Hunerwadel, "The effects-based approach to operations: Questions and answers," Air & Space Power Journal, vol. 20, pp. 53–62, Spring 2006.

[16] Effects-based Approach to multinational operations, Concept of operations (CONOPS) with implementation procedures, Version 1.0. Suffolk, VA: Unites States Joint Forces Command, 2006.

[17] H. Asadi and J. Schubert, “A stochastic discrete event simulator for effects-based planning,” unpublished.

[18] R. W. Hamming, “Error detecting and error correcting codes,” The Bell Systems Technical Journal, vol. 29, pp. 147–160, April 1950.

[19] M. A. Aizerman, É. M. Braverman, and L. I. Rozonoér, “Theoretical foundations of the potential function method in pattern recognition learning,” Automation and Remote Control, vol. 25, pp. 821–837, June 1964.

[20] C.-C. Chang and C.-J. Lin, “LIBSVM: a library for support vector machines,” ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 1–27, April 2011. [Online]. Software available: http://www.csie.ntu.edu.tw/~cjlin/libsvm