Real-Time Simulation-Based Planning for Computer Generated Force Simulation

Jin Joo Lee , Paul A. Fishwick
Department of Computer and Information Sciences
University of Florida

Abstract

Automated Planning has been an active research topic for more than thirty years but only recently has it started to move in the direction of combining planning and execution to achieve what is sometimes called as *Intelligent Reactive Planning*. We propose Simulation-Based Planning (SBP) as a new way to perform intelligent reactive planning. SBP —unlike most other planning systems—integrates simulation into the planning process. Once a set of plans is generated, simulations are used to test and evaluate the plans to choose the most applicable plan for that current situation. In most planning systems, plan evaluation depends on rules alone and because rules must be designed general enough to cover all possible cases, the evaluation is not specific enough for some individual cases. However, when the plan evaluation is done through simulations, the evaluation can be more fine-tuned to individual cases and can allow better plans to be chosen for that individual case. From the military planning perspective, the simulation-based planner is also quite useful due to its ability to perform adversarial and multiagent planning. This is a natural consequence of using simulation in the planning process. By allowing other entities such as the enemy to simulate in parallel with the planner's forces, the planner is able to observe, prior to the actual execution, the effects of adversarial and multiagent actions against its own plans.
[Key Words: Artificial Intelligence, Autonomy, Mission Planning, Computer Generated Forces, Multimodeling]

1 Introduction

In the simulation literature, simulation is defined as " the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output"; Fishwick [8]. In the planning literature, Dean [5] states that the idea of using a model to formulate sequences of actions is central to planning and, given a sequence of actions, a robot can use the model to simulate the future as it would occur if the actions were carried out. Simulation can provide the robot with information that can be used to suggest modifications or to compare the proposed sequence with alternative sequence. And this is in fact, how humans perform planning. Humans have models built and stored in their brain for most objects or systems that exist in the world and these models are used to formulate sequences of actions that would occur in the future if a plan executed. Once simulation models have been built

for a system, simulation can be used as a tool to provide the system with information useful for evaluating its hypothesis. Therefore, it is logical that we employ simulation within the planning process to simulate the results of a proposed plan before the plan is selected for execution.

Planning becomes very complex for any real world planning problems that takes place in an environment over which the planner has no control, such as another agent or an enemy, and when there is uncertainty of available information or uncertainty of another agents reaction. In such cases, accurate prediction of the resulting states of plan execution will be difficult. To overcome this increase in complexity of reasoning, many new approaches have been introduced [20, 4, 10, 9]. Mission planning in the military has all of these properties and more. For planners that have to work in the Computer Generated Force simulation [11] using the Distributed Interactive Simulation Environments, they have to handle these uncertainties in real time since the Computer Generated Forces have to fight against the opposing force(which are normally human trainees) in real time. Recent approaches use some form of rule-based expert systems such as SOAR [12, 13, 1, 19]. Since thousands of rules are involved just to model one object, the complexity of maintaining and reasoning about plans is not easy since rule-based systems are usually centralized. SBP can solve the problem of accurate prediction of uncertain environment by allowing the use of individual simulation models to predict the behavior of individual objects in the world. Second, SBP can produce plans in real time since we can allow simulation of plans at different levels of abstraction. Multimodeling [7, 6, 8, 14] will be used to model processes and agents at multiple abstraction levels. Related work [15] suggests the use of a coordinated set of methods, each method having different scope and performance. Some experimental implementations of this approach has been done on Soar [18].

SBP as a general methodology will be discussed in more depth in the next section. In sections 3, 4, and 5, we describe the design of a sample application in the domain of mission planning for Computer Generated Forces (CGF). Section 6 discusses the demonstration mission that was performed on the prototype. The implementation of the prototype is discussed in section 7. Finally, conclusions and future work are discussed in section 8.

2 Simulation-Based Planning

Simulation-Based Planning refers to the use of computer simulation to aid in the decision making process. In much the same way that adversarial trees are employed for determining the best course of action in board games, SBP uses the same basic iterative approach with the following items: 1) a model of an action is executed to determine the efficiency of an input or control decision, and 2) different models are employed at different abstraction levels depending on the amount of time remaining for the planner to produce real time decisions. In the first item, board game trees implement a static position evaluation function whereas, in SBP, a model is executed to determine the effects of making a *move*. In the second item, SBP can run more detailed models of a physical phenomenon when there is sufficient planning time or fast computation or parallel methods are instrumented.

Past reasons for using rules exclusively in planners centered around the idea that rules are less expensive to execute, and that they adequately reflect the heuristics of the human decision maker (the company commander, in our case). However, in addition to trying to model human decision making, it is just as important (if not more) to create planners which yield the best adversaries

whether or not the adversaries conform to a specific doctrine. In actuality, some combination of 1) capturing human decision making heuristics and 2) creating automated planners capable of near-optimal decisions (based on objective functions) is the ideal solution, and it is toward this solution that our research is directed.

The military has been using simulation-based planning for many decades in the form of *constructive model simulation*. A constructive model is one based on equations of attrition and, possibly, square or hexagon-tiled maps using discrete jumps in both space and time. To decide whether to accept a course of action, one can use a constructive model (a "wargame") to evaluate the alternatives. Related work by Czigler et al. [2] demonstrates the usefulness of simulation as a decision tool. Our extension in SBP is one where we permit many levels of abstraction for a model, not just the aggregate abstraction level characterized by Lanchester equations and combat result tables. The idea is to simulate at the level of abstraction permitted by the remaining time left to the planner during real time decision making. The notion that simulation can be used for decision making is covered in several disciplines, such as for discrete event based models [21].

3 Application: Mission Planning for Computer Generated Forces

The mission planner is an integral part of a larger project of the Institute for Simulation and Training (IST) called "Intelligent Autonomous Behavior by Semi-Automated Forces in Distributed Interactive Simulation" which was funded by the U.S. Army Simulation Training and Instrumentation Command (STRICOM). The goal of the planner is to automatically derive plans for a semi-automated force (CGF), at the company level initially, so that the force will provide an Army trainee with an effective training experience. Therefore, the planner represents a decision making algorithm at the level of a company commander. Planning is only a small part of the overall project, which includes efficient line of site (LOS) determination, terrain reasoning, intelligent target acquisition and behavior representation for CGF entities. The planner takes orders from the battalion level and translates these orders, with a tight coupling with the terrain analyzer, into efficient plans for the CGF platoon entities. In addition to planning for its subordinate units, the planner must also also be able to monitor the execution of the plan, react to unexpected situations and replan if necessary.

3.1 Planner Design

Figure 1 displays the architecture of our planner in relation to the IST CGF Testbed [11]. Each commander in the IST testbed is simulated by a Command Entity (CE). The Planner performs major functions of this entity. The planner has two phases: the Reactive phase and the Planning phase–where a phase is a group of states that collectively display a behavior. Only one phase is active at any given time and the starting phase is the Reactive Behavior. The decision as to which phase becomes active is made by the current active phase based on the inputs. There is no single 'main' algorithm that controls the whole process. Thus, the decision is made in a distributive manner. In particular, the decision to give up planning and report to higher level unit is made by the planning behavior. The inputs are either OPORDs or SITREPs, and depending on the

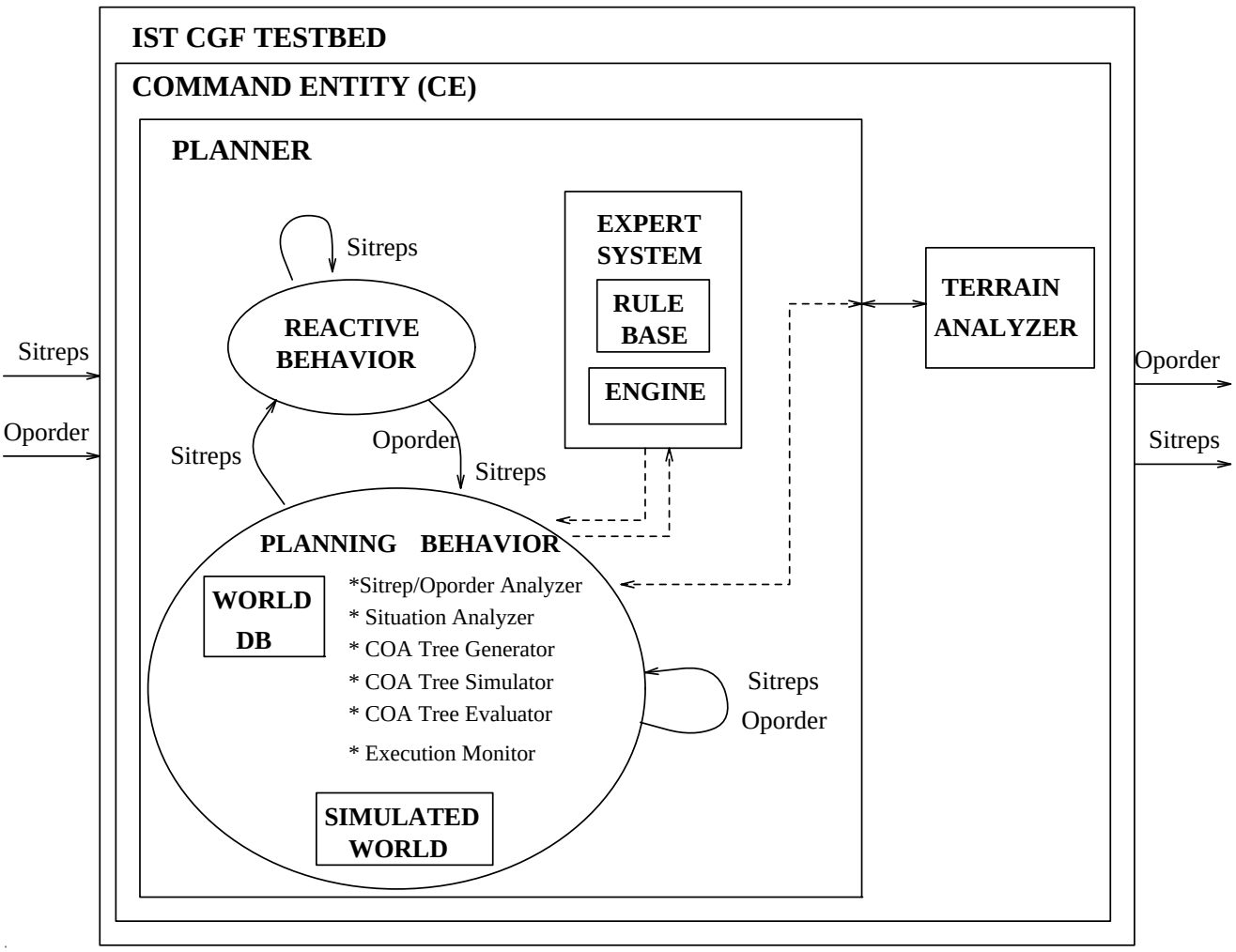


Figure 1: Planner Architecture

type of SITREPs and OPORDs, different decisions will result. Each Planner in the CE is made up of the following components.

3.1.1 World Database(DB)

The World Database contains information about the battlefield. This is not a complete spatial representation of the battlefield (the Terrain Analyzer (TA) has this information) but a simplified database which mainly contains information that is known to the CE (and not known to the TA). Since the TA does not have any information regarding the location of enemy or friendly units and does not keep track of the locations, the planner needs to keep track of these locations and the status of the units in the World Database. This database is created as soon as the CE starts to exist. Initially it contains its own location and will be updated with new information as it becomes available to the CE via SITREPs or OPORDs.

The battlefield is divided into rectangular regions and represented by the elements of a matrix. This is a very low resolution of the battlefield because the purpose of this matrix is mainly to speed up the look up time of unit locations and other information by organizing the linked

lists into regions. Each element of the matrix is linked to a linked list that contains all the information the CE has about that region. Each node will have more exact locations along with other available information such as status of the unit.

3.1.2 Reactive Behavior

The Reactive Behavior module displays reactive behavior necessary for survival when immediate action is required. This behavior may be different for different types of entities. The module is initialized with a generic set of behaviors at the start and may be modified with any reactive behaviors provided by an OPORD.

3.1.3 Planning Behavior

The Planning Behavior module has the ability to generate orders for its subordinate entities from an OPORD given by a higher level entity. This module is made up of the following smaller modules where the order in which they are presented actually coincides with the algorithm steps of a typical planning process.

1. **SITREP/OPORD Analyzer** parses the SITREP/OPORD to update the World DB.
   - **OPORD**: is further parsed to generate a list of task(s) to be achieved. The Situation Analyzer is called next with this list.
   - **SITREP**: SITREP is analyzed to decide if any immediate action is required, if any replanning is required, or if any SITREP needs to be generated. The Execution Monitor is called with the decision.
2. **Situation Analyzer(SA)** is a collection of rules that analyze the given situation using the World DB to generate the appropriate constraints for the ROUTE or the DEF_TACT_POS call to the TA. The decision as to which of the two calls to call first depends on problem size reduction. In other words, in a given situation, the call to ROUTE may produce many routes whereas a call to DEF_TACT_POS may have produced few tactical positions. In such a case, we can first call DEF_TACT_POS to acquire a set of tactical positions and then call ROUTE with these tactical positions which will reduce the number of returned routes due to the given constraints. Thus, the SA will perform some alternate calls of ROUTE and DEF_TACT_POS to produce an appropriate number of alternate routes. The COA Tree Generator is called with these alternate routes.
3. **Course of Action(COA) Tree Generator** Using the set of alternate routes produced by the SA, the COA Tree Generator generates a COA Tree where the 1st level contains alternative subunit combinations. and 2nd level contains alternative route combinations. The following levels can contain other alternatives such as varying the role of platoons in different formations. We can extend the tree as much as we want with any other possible alternatives at each level. Also note that some alternatives may be omitted at this stage and later generated during the Simulation/Evaluation step.

   Once such a tree is generated, the tree is pruned using various methods and rules before it is passed onto the COA Simulator. Many alternatives can be pruned away by using a

military expert knowledge system. However, we must not prune away too much since many alternatives should be left to be explored via simulation. The purpose of using simulation may be lost if the choices have been made already. Next, the COA Tree Simulator is called with the COA Tree.

4. **The COA Tree Simulator** is invoked to simulate the set of COA trees that have been generated. This is done by creating a Simulated World (SIMDB) and performing the simulation of friendly and enemy units by time slicing between actions (move, look, fire) and observation by each unit. It is also time sliced between friendly and enemy units. Different methods can be used in simulating friendly and enemy units. One method is to allow the enemy units to have the same planning capabilities as the friendly units but with different tactics. This method would be quite realistic, but it can be quite time consuming. In general, a complete simulation will be more time consuming than a temporal projection using rules. If computing capabilities are limited, we can perform simulation at different levels of abstraction [14, 6] where each higher level will use less computational power. Another solution is to let the enemy units follow a less sophisticated planning process allowing limited intelligence. This is the approach we are taking for this particular application.

   Thus, when the two opposite forces are inside the circle, engagement is likely when the enemy has line of sight to our unit. The actual simulation algorithm is as follows:

While (planner active) do
    Update entity state variables
    Perform line of sight (LOS) check
    Engagement check
    Update current clock time by ΔT
End While

For each course of action, the simulator operates as follows. State variables defining an entity's position and orientation are updated at each time slice. In low mobility areas or areas with a steep terrain gradient, the movement is slower. Also, for some terrain features, as with fords or chokepoints, a simple queuing model can be executed to keep track of entities that must wait for entities that are blocking the path. Service times and speed values are obtained by sampling from a probability distribution appropriate for the blocked area. A line of sight (LOS) check and range calculation is done between the entity being simulated and known enemy locations. If the enemy is within range of certain weapons (such as a HEAT or Sabot round), an engagement will ensue. We are unsure as to the level of detail required to simulate the engagement for planning purposes. However, these may be extended as behaviors such as "seeking cover" are integrated into the planner. The simulation proceeds, while updating the simulation time by ΔT until either the plan has been fully simulated, or the planner is interrupted.

There are several advantages to using Simulation to predict the results of plan execution.

(a) Simulation provides a uniform method without resorting to adhoc solutions. In simulation, each entity in the environment is simulated in a uniform and consistent manner by using models that represent both the physical and behavioral properties. Thus, simulating a plan is a natural consequence of simulating each of the entities by itself without having to worry about the global state change as a result of each entities action. In some ways, simulation can be viewed corresponding to object-oriented programming methods. Thus, each object is simulated using its own model.

(b) Because there is no central reasoning node for the simulation but many individual simulation models for different entities, scalability is a natural consequence. Extendibility is another advantage simulation provides. For example, the effects of adding a new type of entity will be clear, only the behavior models of each entity must be updated to recognize and reason about this new entity.

(c) Similar to how simulation is used for visualization, simulation can be easily used to perform visual playback of how a plan was simulated to explain the planner's decision.

Initially, the Simulated World is created from the World DB and then the status of the world is updated as entities are being simulated. The simulator uses the TA to update the Simulated World. The outcome of the simulation is then fed into the COA Tree evaluator.

5. The **Execution Monitor**

The Execution Monitor is the main driver of the Planning Behavior module.

(a) Issue the set of chosen subtasks in the plan to each units in an OPORD format.

(b) Execute its own subtask if any.

(c) If any SITREP is received,

i. Call the SITREP/OPORD Analyzer.

ii. If the decision returned calls for

- immediate action, it is handled by the REACTIVE behavior module which accesses the mini Expert System to react accordingly. In doing so, the REACTIVE module also takes into account any Engagement Criteria given in the OPORD.
- replanning, the SA is called to start a planning process with the newly updated World DB.
- giving up planning at the current level, the CE sends SITREP to its higher unit reporting of its current status and waits for further orders.

3.1.4 Expert System

The mini Expert System module contains rules to aid the planning process in making decisions such as choosing routes, choosing best COA tree, performing analysis of situations, OPORDs and SITREPs.

4 Interface between Planner and other Command Entities

Similar to how Operations Order (OPORD) are sent as directives to subordinates in military Command and Control (C^2), the CGF Command Entity is also expected to receive and send OPORDs. The standard military format for OPORDs contain considerable amount of overlap throughout the different sections. Since our OPORDs will be sent as messages within the program and each message has a limited size of 300 bytes, there was need to modify the military standard format to make it more concise. A standard military OPORD consists of five paragraphs: Situation, Mission, Execution, Service Support and Command and Signal [16]. For our application, we are mainly concerned with the first three where Situation describes the situation and missions of enemy forces (if known) and friendly forces, Mission states the mission the unit issuing the OPORD is trying to achieve, and finally Execution expands on the Mission statement by describing the specific tasks to be accomplished by each subordinate units. The resulting format containing the OPORD used in our demonstration mission is included in appendix A. In our OPORD format, the Execution paragraph takes the form of a Synchronization Matrix where the rows represent the tasks for each subunit and the columns represent different phases of the mission. The Synchronization matrix is issued by the commander to its subordinate units where individual unit locates the row that contains the orders for that specific unit and executes the tasks issued to them. Each unit transitions to the next phase of the mission based upon its own transition code. It can initiate its own transition: On Own Initiative or it can only transition after receiving an order from its commanding unit: On Order. A corresponding matrix to our demonstration order is shown in figure 2. This same matrix is sent to all three companys and each company extracts the applicable row of tasks to be accomplished. In our case, the orders for company 1 is contained in the first row. The second and third rows do not directly affect the planner in our demonstration. It can affect the planner, however, if some type of coordination was required among the three companies. Since the prototype assumes there are only one phase to a mission, phase 2 is ignored in the current version of the prototype.

Situation Reports (SITREPs) are another type of order that is sent to the superior unit by a subordinate unit to report either scheduled situations (e.g. when a mission has been accomplished, when a unit hits a check point, etc.) or unscheduled situations(e.g. when a threat is identified, when overall combat strength falls below a predetermined level, etc.). The SITREP format is also described in appendix A.

5 Interface between Terrain Analyzer and Planner

As mentioned earlier in section 1, the mission planner is a part of a larger project at IST. The planner must integrate with many other components in the IST project, but for the most part it must work with the IST's Terrain Analyzer.

5.1 Terrain Analyzer

The Terrain Analyzer is the planner's only source of information where terrain is concerned and thus the planner uses the TA quite extensively during the planning process. The TA is

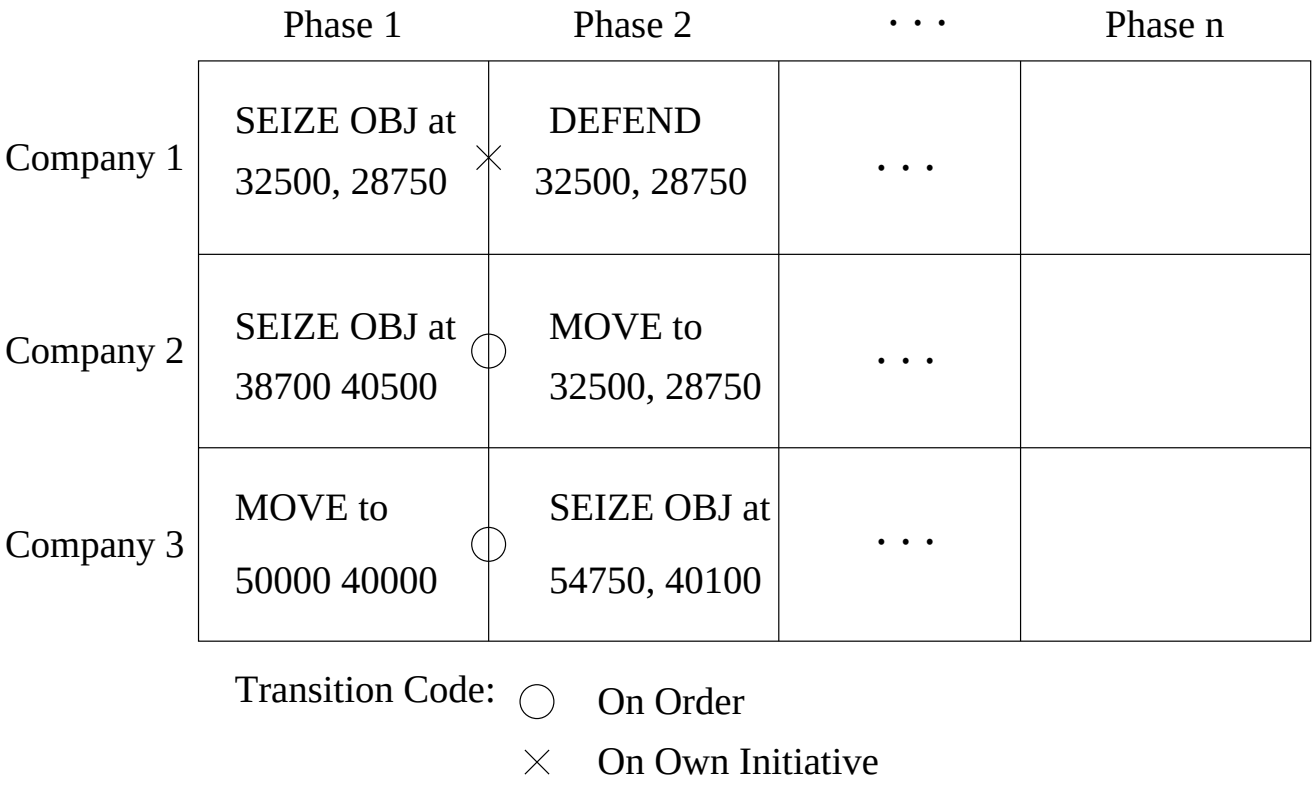


Figure 2: Synchronization Matrix for Demo OPORD

responsible for route planning, finding tactical positions, computing Line of Sight and answering questions about terrain features. The interface between the TA and the planner is established by four types of calls; ROUTE, DEF_TACT_POS, LOS and TERRAIN_FEATURE.

- ROUTE: Given the maximum number of routes, start and end position, the unit boundary, the minimum percentage of concealment, the mission type and the direction of approach to the OBJ (located at end position), the TA returns multiple routes that satisfy the given constraints. Note that the direction of approach does not apply to cases when the end position does not have an enemy unit and the mission type is not a SEIZE mission. Any intermediate enemy position that needs to be avoided can also be given along with a radius and the TA will generate routes avoiding these locations radius distance away from the avoid locations. Each returned route has a route id, length and percentage of concealment relative to the route. The actual route is represented as a piecewise linear curve made up of a set of line segments. Each line segment contains not only it's begin and end points but also the percentage of mobility, passable width, probability of LOS to the OBJ and the terrain_type (ground, road, ford).

- DEF_TACT_POS: Given the type of mission, this call requires the TA to provide locations for a given type of tactical position(s). The current position of the unit and the OBJ must also be given. There are three different types of position: SUPPORT_BY_FIRE, DEFENSE, and ATTACK. We have two types of mission : SEIZE and DEFEND. Finally, the enemy location must also be provided to the TA for SUPPORT_BY_FIRE and ATTACK positions.

- ALOS: This call directs the TA to perform an area to area Line_of_sight determination. Locations 1 and 2 are given, each with Radius1 and Radius2. The Radius1 describes the circular area centered at 1 and Radius2 describes the circular area centered at 2. 1_#_PTS constrains the number of points where LOS should be tested within circle centered at 1. 2_#_PTS constrains circle at location 2 in the same manner. If Radius is 0 for both locations then a simple point to point LOS is performed. The returned data is the probability of sightings over the #_PTS tested within the two areas.

- TERRAIN_FEATURE: This call requires the TA to return all the terrain features that are included within the radius of a given point. The planner supplies the TA with a center point and a radius and the TA returns one or more of the following types as a terrain_feature: ROAD, FORD, GROUND, CHOKE_POINT.

6 Demonstration Mission

6.1 Demonstration scenario

Figure 3 illustrates the demonstration scenario. The friendly company unit 1 (the company entity receiving the OPORD) is situated at Assembly Area(AA) located at (50000, 52500). Company unit 1 is made up of 3 platoons: platoon A is made up of 4 M1 Abrams Main Battle Tanks and each of the remaining 2 platoons B and C are made up of 4 Bradley Infantry Fighting Vehicles. There are two enemy platoons: opfor platoon A is made up of 4 M1 tanks located at (32500,

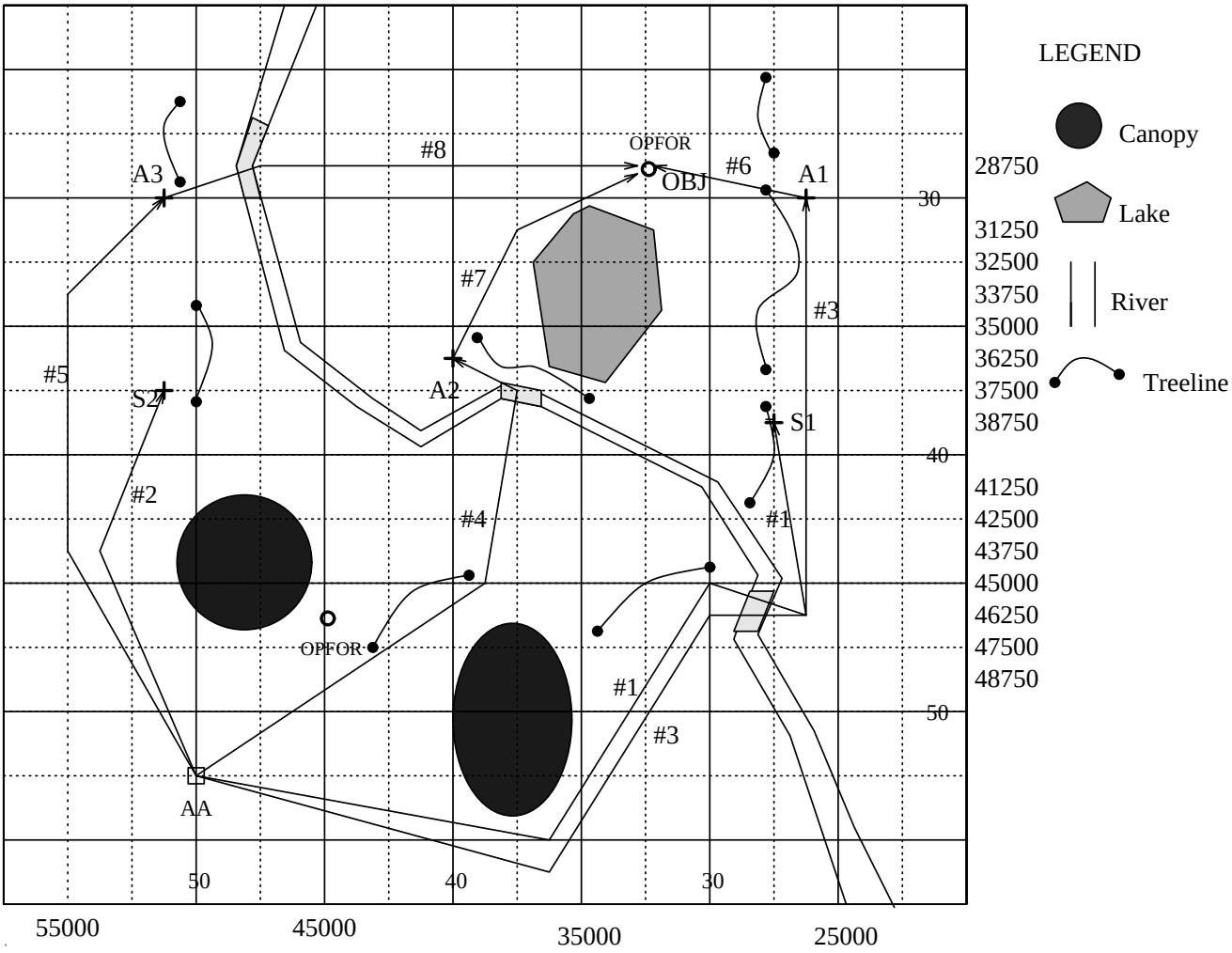


Figure 3: Company Mission and Routes

28750) and opfor platoon B is made up of 4 M2 fighting vehicles located at (45000, 46250). The OPORD given to company unit 1 is to "SEIZE the Objective at (32500, 28750)". The company unit boundaries are given as the rectangular area in the figure 3.

The goal of the command entity is to accomplish the mission with minimal loss of strength. In order to find a mission plan that will satisfy this criteria, the planner simulates several alternative plans, compares the results and chooses the best one. However, these alternative plans need to be generated by the planner first. The planner has to be careful not to generate too few or too many alternative plans since too few may not include a potentially good plan and too many may take too much computation time. Figure 3 also shows the various tactical positions and routes that can be obtained through calls to IST's Terrain Analyzer. Thus, the alternative plans are mainly based on two alternatives: different subunit combinations and different route set combinations. In other words, the alternative plans depend on which unit or units go on which route.

6.2 Demonstration Mission Planning Results

From Figure 3, one should be able to observe that any friendly units traveling on route 4 is likely to engage in combat with the opfor unit stationed at (45000, 46250). The friendly unit may not be totally destroyed but considerable amount of strength may be lost during combat and therefore will result in a lower score. Thus, any plan that includes route 4 will have lower scores than other plans. The evaluation scores produced by the mission planner for the demonstration mission are listed below in the order of decreasing scores. The scoring formula is discussed in depth in section 7.5.

For the alternative plans where the company stays together as a company and travels on a single route to the objective: (Note that routes 1 and 2 are not considered since they end at Support_By_Fire positions.)

```
Route 3 -> 6 : 128.0
Route 5 -> 8 : 128.0
Route 4 -> 7 : 89.8068
```

When the company splits up into two groups and travels on two different routes, there are also different subunit combination alternatives. For example, platoons 1 and 3 can travel together on one route and platoon 2 can travel alone in the other.

Following are the evaluation scores for such a case:

```
Route 3,1 -> 6,- : 158.000000
Route 3,2 -> 6,- : 158.000000
Route 5,1 -> 8,- : 158.000000
Route 5,2 -> 8,- : 158.000000
Route 3,4 -> 6,7 : 152.525711
Route 3,5 -> 6,8 : 144.205093
Route 4,5 -> 7,8 : 116.941719
```

```
Route 4,1 -> 7,- : 112.500000
Route 4,2 -> 7,- : 112.500000
```

7 Implementation

This section describes the implementation of the mission planner prototype. Any assumptions that have been made are also discussed. Each of the significant modules are described along with the functions that are defined in them. Because this prototype is not connected to either of the IST's TA or Simulator Testbed, the REACTIVE BEHAVIOR module has not been implemented. As a result, the EXECUTION MONITOR has not been implemented since it has little meaning if no dynamic SITREPs can be received from the Testbed. The following sections discuss the individual components that has been implemented.

7.1 OPORD

In this prototype, the following assumptions exist for the OPORD:

1. All unit_ids are consecutive from 1 to number of friendly units or number of enemy unit.
2. Only one phase and only one task within a phase.
3. The format is as described in A.

Currently, the planner requests only one route per ROUTE_REQ call and will assume this is always the case. The tactical positions that start with A are ASSAULT_POS and those that start with an S are SUPPORT_BY_FIRE position.

7.2 Data structures

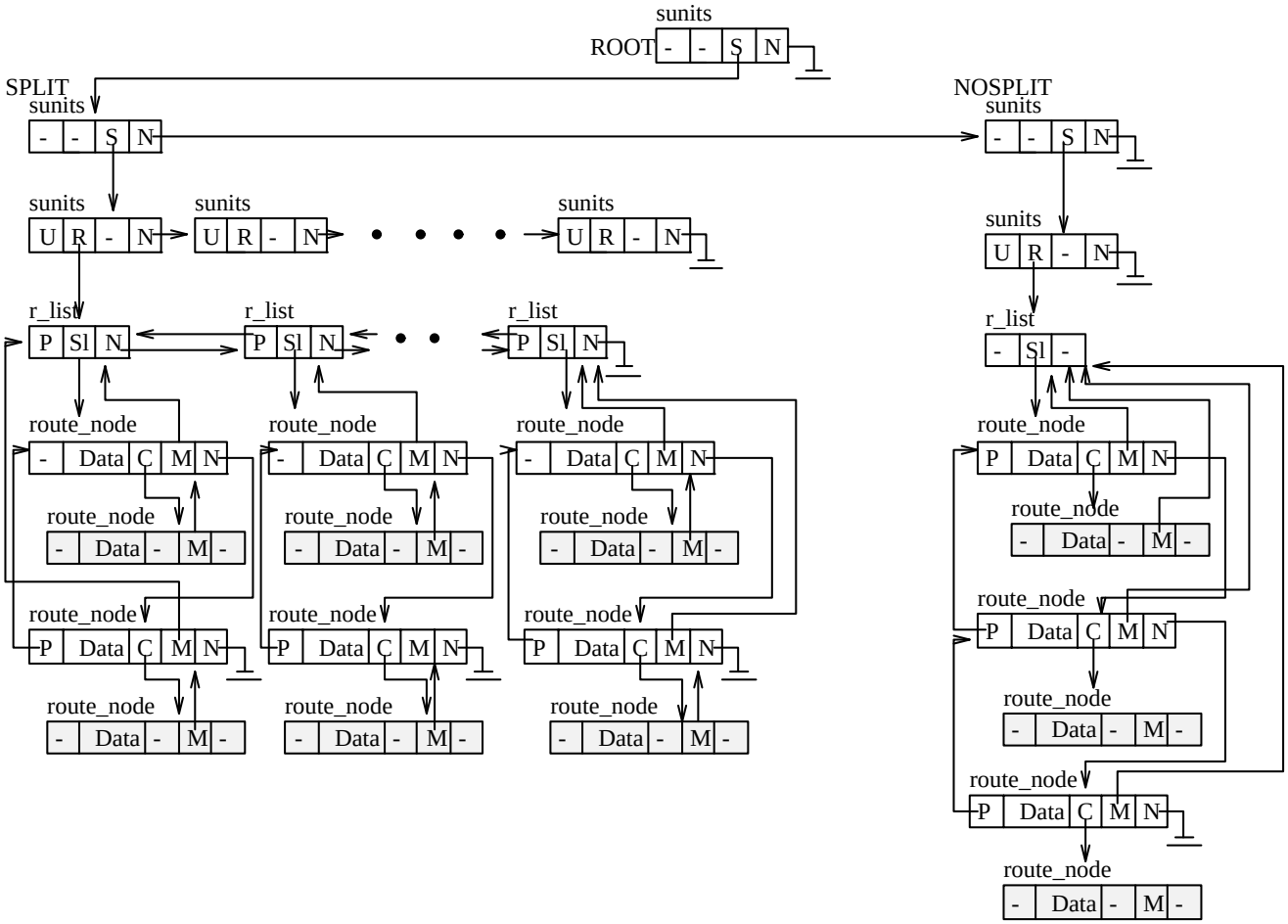
In the prototype, the COA tree lists two types of alternatives: alternative subunit combinations and alternative route combinations. Subunit combinations are represented by a linked list of `sunits` nodes and a set of route combination is represented by a linked list of `route_nodes`. Section 7.4 describes the tree in more detail.

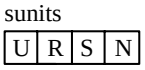
7.3 SITREP/OPORD Analyzer and Situation Analyzer (SOA module)

The SOA module contains the following functions:

1. `OP_ORD *Read_OPORD()` reads in the OPORD from a data file called "OPORD" and initializes an instance of the OP_ORD data structure and returns the pointer to this instance.
2. `Init_WorldDB(OP_ORD *op)` takes the initialized OP_ORD structure op and initializes the global data structure WDB. The underlying data structure for WDB is w_grid and is defined in the header file "plan.h".

COA Tree Structure and Types





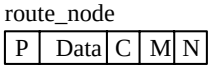
STRUCT SUNITS

U : unit_comb

N : sunits *next

R : r_list *child

S : sunits *schild



STRUCT ROUTE_NODE

Ri : route_idx

Sc : score

Ct : cut

Fu : friendly unit_strength

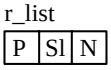
Ou : op_unit_strength

N : route_node *next

P : route_node *prev

C : route_node *child

M : route_node *parent



STRUCT R_LIST

Sl : single_list *route_node

P : r_list *prev

N : r_list *next

Figure 4: COA Tree Structure and Types

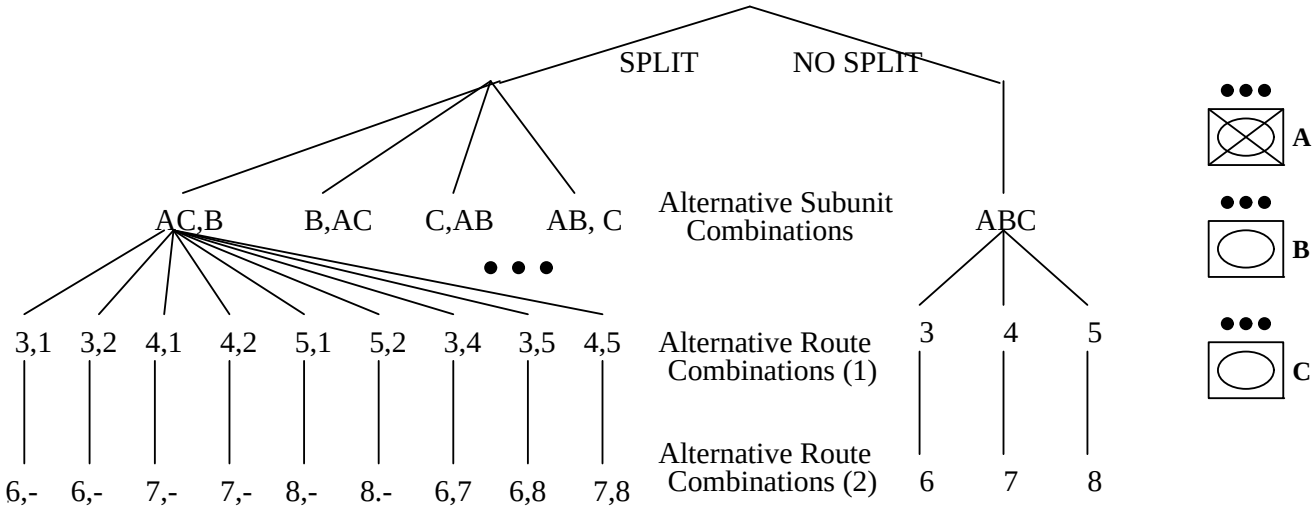


Figure 5: COA Tree of the Demonstration Mission

3. `Situation_AnalyzerI(OP_ORD *op, int phase_num, int *num_tpos, int *num_rts)` takes op and phase_num to find the current mission given in the oporder op to the planner. Depending on the type of mission (one of MISSION_TYPE_ENUM), the appropriate calls are made to the TA. Currently, SEIZE and DEFEND missions have been implemented. For a SEIZE mission, the SA will first make a set of DEF_TACT_POS calls. Using the returned TACT_POSs , the SA requests routes leading to these tactical positions via ROUTE_REQ calls. Finally, the SA receives routes that are returned from the TA. As stated in the beginning of this report, the planner is not connected to the actual TA and therefore what would have been actual calls to the TA are written out to a file called "calltota". Also, the data that would have been returned from the TA are read from an input file called "fromta".

7.4 COA Tree Generator (COA module)

The COA module contains the function :

1. `struct sunits *Create_COA_TREE (RET_ROUTE routes[], int num_rt_calls, int num_phases, RET_TACT_POS Tpos[], int n_tpos_calls , OP_ORD *op, int phase_num)`. This function returns the pointer to the root of the generated COA tree. Due to the dynamic nature of the tree, it is implemented using linked lists. The tree has two main subtrees: the SPLIT subtree and the NO_SPLIT subtree.

   Figure 5 shows the COA tree generated by the prototype. The SPLIT subtree describes the course of action for a company where the company will be split up. Given that a company has 3 platoons, the number of routes needed is always greater than or equal 1. If two platoons go one route and one platoon go another, two routes are needed in total. If each platoon goes on a different route, three different routes will be necessary. The 1st level of this subtree will contain all the possible combinations of splitting a 3 platoon company. Currently, the set of combinations are read from a data file called "Subunits". The justification is that some heuristics must apply in dividing up the company

Variable	Meaning
A	Attacker strength (equivalent divisions, EDs)
D	Defender strength (EDs)
F	Attacker to defender force ration: $F = A/D$
ALR	Attacker loss rate (fractional loss per unit time): $ALR = \frac{dA/dt}{A}$
DLR	Defender loss rate: $DLR = \frac{dD/dt}{D}$
RLR	Ratio of (relative) loss rate: $RLR = \frac{ALR}{DLR}$
Parameter	**Meaning**
K_d	Kills per day of attacker EDs per ED of defender
K_a	Kills per day of defender EDs per ED of attacker

Table 1: Variables of the Aggregate Model

and since there are no expert systems to aid this process in the prototype, an expert can be consulted in creating this "Subunits" file using his expertise and knowledge of the Company's composition. The heuristic used in this scenario is not to allow Platoon A to travel alone at any time since M1s are considerably slower and lower power than M2s. This restricts the SPLIT combination to 4 sets: (AC,B) (B,AC) (C,AB) (AB,C).

From these combinations, the Create_COA_TREE generates possible combinations of route sets. Since the mission is SEIZE, at least one route should lead the platoons to an ASSAULT_POS. These routes are 3,4,5 according to 3. Thus, the possible set of route combinations are (3,1), (3,2), (4,1) (4,2), (5,1), (5,2), (3,4), (3,5), (4,5). The second level of route combinations in the COA tree contains the routes that connect each of the 1st level routes to OBJ if possible. For example, route 6 extends route 3 to OBJ. However, route 1 is not extended to the objective because it's a SUPPORT_BY_FIRE position.

The NO_SPLIT subtree has a single alternative subunit combination (ABC) since no split up is allowed in the unit combination. Therefore only a single route set alternatives that lead to an assault positions (3,4,5) are possible. The second level routes are (6,7,8) respectively. No pruning is being done in the current implementation.

7.5 Simulator (SIM module)

The SIM module contains the function :

1. `int Simulate (struct sunits *node, int level, int SPLIT, RET_ROUTE routes[], OP_ORD *op)`. This takes each level of the COA subtree and simulates each route and calculates a score for each friendly subunit per each route. In the current version, the enemy unit is simulated in a very limited manner. The enemy unit is assumed to remain stationary and only engage in combat when an opposing force unit has been sighted. Currently, our method employs the Aggregate Combat Model [3]. Table 1 lists and defines the variables of the model.

The model takes as inputs the initial values of attacker and defender strengths, A_0 and D_0, and values of the attrition coefficients K_a and K_d. There is also $F = A/D$, the attacker to defender force ratio and $RLR = (K_d/K_a)/F^2$, the ratio of relative loss rates. The only process in this model is that of attrition, governed by Lanchester square law:

$$\frac{dA}{dt} = -K_d D\,, \quad \frac{dD}{dt} = -K_a A$$

An important assumption made in general is the 3 to 1 rule, which says that $RLR = 1$ when $F = 3$. This means the breakeven point is at a force ratio of 3 to 1 which is a reasonable assumption in most cases since a defender is prepared to defend in a favorable terrain. The reference assumes $K_d = 0.18$ which makes $K_a = 0.2$. And that is also how K_d is initialized in the prototype. Thus, at a force ration of 3, the attacker would be losing 6% of its strength per unit time. Another assumption is that the units are equivalent in size and that is what is assumed in the prototype for now. The enemy unit is single platoon defending OBJ and 3 friendly platoons trying to seize the OBJ. According to the route and tactical position combination in the COA tree, the simulation result will be different since the number of friendly platoons actively involved in combat will vary.

The simulation result is recorded in the form of an integer number, the evaluation score, which is calculated using the following formula:

$$score = strength\ of\ unit + proximity\ to\ OBJ(\%)$$

If at any point in time during the simulation the strength of a unit is below a certain threshold (5 %), it is considered to be destroyed and no further simulation will be run on that particular branch of the COA tree. Thus, the overall simulation strategy is branch and bound. Depending on the order of the calls, however, it is possible to simulate the COA tree in a somewhat depth-first manner. It is possible to simulate the 2nd level of the SPLIT tree before the 1st level of the NO_SPLIT tree. The each unit's score is stored in every route_node of the COA tree representing the simulation result of that particular branch.

Since the planner does not currently have access to the TA, the Line of Sight check is being done by a simple function LOS_check which checks the distance between two units. At present time, the terrain is assumed to be flat and open, not affecting the line of sight calculation. When the TA is connected, the planner will be able to have access to a more accurate LOS check which will account for different types of terrain that may obscure the view. Within the function `long update_loc()`, the speed of the units are currently determined only by the terrain type. ROAD has 100% mobility, GROUND has 80% and FORD has 50% mobility.

7.6 Input and Output data

There are basically four different types of information that is given as input to the planner at different stages of the planning process. When the planner is first started, it is given an

OPORD, a set of subunit combinations to be used in the generation of the COA tree, and the initial strength of the individual subunits. During the plan generation process, the planner will make various calls to the TA and get back from the TA information such as tactical positions and routes. Once these plans are generated, the planner simulates them and outputs the final evaluation scores of alternative plans in decreasing order.

7.7 Expert System

In the prototype version, very simple rules have been used and they exist as if-then statements inside the implementation. As more depth military expert knowledge are acquired, it may become reasonable to implement the expert knowledge as a separate Expert System. As the next step, we plan to use Soar to implement the military expert part of our planning system. The Soar architecture provides a stratified approach to specifying, designing, and building Knowledge-Based systems. Soar is well known in the military simulation field through the Soar/IFOR project [17] where Soar is being used to generate the behavior of an automated agent for the Tactical Air Simulation. No extensive mission planning is involved in the project; the goal is to simulate the behavior of a single pilot.

8 Conclusions and Future work

We have shown how we are able to perform planning with fewer rules by using simulation to project and evaluate potential plans. Simulation allows the planner to project a broader class of results in a uniform manner. In the mission planning aspect, SBP is useful because it is easily scalable, extendible and explainable.

The simulation-based planner prototype runs in real time on an IBM 486 PC and is currently being transported to run on the Sun Unix Workstations. Adding and testing different strategies of choosing the evaluated plans is our immediate future work. To follow the military's operational concepts of acting with an *initiative*, incorporating unpredictability of actions into the planners is a major task. A possible solution is to allow the planner to choose nondeterministically among those plans that have evaluation scores above some threshold. Another important extension is to allow the enemy to react nondeterministically during simulations so that the evaluation scores will come out differently at different runs.

Since the concept of SBP is fairly new, we plan to create experiments to further analyze the methodology thoroughly. We will build two planning systems; one that employs the simulation-based models and one that is entirely rule-based. For the rule-base part, we intend to use the Soar architecture as our expert system framework. Then, we will compare the results through several problem domains. First, we will experiment with classical AI problems such as the blocks world problem and the machine shop scheduling problem. We will then experiment with the mission planning problem. The comparison criteria between the two systems are: 1) Speed - number of plans generated per unit time. 2) Success - the rate of success of plans. 3) Model complexity - how easy is it to design, build and comprehend the system model? 4) Maintenance - extensibility and modifiability. 5) Reactiveness - ability to react during or after the planning process. 6) Adaptability - ability to adapt planning to dynamic changes of the environment during planning.

In order for the experiment to be valid, we must maintain several variables such as 1) Knowledge: the set of knowledge that is used in both the planners must come from the same source, 2) Data: the data provided to the planner such as Terrain Analysis data must be the same, and 3) Evaluation function: the same objective function must be used to choose the best plan. By successfully maintaining the 3 variables as above, we can ensure the validity of the experiments—allowing us to observe the strong and weak points of the systems. to thoroughly evaluate the SBP method.

In the long-term, we plan to apply the simulation-based approach to other areas of planning such as traffic control.

References

[1] W. Braudaway. A Blackboard Approach to Computer Generated Forces. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 11–20, Orlando, FL., 1993.

[2] M. Czigler, S. Downes-Martin, and D. Panagos. Fast Futures Contingency Simulation: A "What If" Tool for Exploring Alternative Plans. In *Proceedings of the 1994 SCS Simulation MultiConference*, San Diego, CA, 1994.

[3] Paul Davis. Aggregate Combat Models. Technical report, RAND Corporation.

[4] T.L. Dean and K. Kanazawa. Persistence and probablistic inference. Technical Report CS-87-23, Brown University Department of Computer Science, 1987.

[5] T.L. Dean and M.P. Wellman. *Planning and Control*. Morgan Kaufmann, 1991.

[6] P. A. Fishwick. An Integrated Approach to System Modelling using a Synthesis of Artificial Intelligence, Software Engineering and Simulation Methodologies. *ACM Transactions on Modeling and Computer Simulation*, 2(4):307 – 330, 1992.

[7] P. A. Fishwick and B.P. Zeigler. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation*, 2(1):52–81, 1992.

[8] P.A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Prentice-Hall, Inc, 1994.

[9] K. Hammond. Cased-based planning. In *Perspectives in Aritificial Intelligence*, volume 1. Academic Press, 1989.

[10] K. Kanazawa and T. Dean. A Model for Projection and Action. In *Proceedings of IJCAI-89*, pages 985–999, Detroit, MI, 1989.

[11] C.R. Karr, R.W. Franceschini, K.R.S. Perumalla, and M.D. Petty. Integrating Aggregate and Vehicle Level Simulations. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 231–239, Orlando, FL., 1993.

[12] J. Laird, P. Rosenbloom, and A. Newell. *Universal Subgoaling and Chunking.* Kluwer Academic Publishers, 1986.

[13] J. Laird, P. Rosenbloom, and A. Newell. Soar: An Architecture for General Intelligence. *Artificial Intelligence*, 33, 1987.

[14] J.J. Lee, W.D. Norris, and P.A. Fishwick. An Object-Oriented Multimodeling Design for Integrating Simulation and Planning Tasks. *Journal of Systems Engineering*, 3:220–235, 1993.

[15] Soowon Lee. *Multi-Method Planning.* PhD thesis, Department of Computer Science, University of Southern California, 1994.

[16] Dept. of the Army. *The Tank and Mechanized Infantry Battalion Task Force, FM 71-2.* Dept. of the Army, 1987.

[17] P.S. Rosenbloom, W.L. Johnson, K.B. Schwamb, and M. Tambe. Intelligent Automated Agents for Tactical Air Simulation: A Progress Rep ort. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 69–78, Orlando, FL., 1994.

[18] P.S. Rosenbloom, J.E. Laird, and A. Newell. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Aritificial Intelligence*, 47:289–325, 1991.

[19] M. Salisbury and H. Tallis. Automated Planning and Replanning for Battlefield Simulation. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 243–254, Orlando,FL., 1993.

[20] M. Schoppers. Universal Plans for Reactive Robots in Unpredictable Domains. In *Int. Joint Conference on Artificial Intelligence*, 1987.

[21] Arne Thesen and Laurel E. Travis. *Simulation Model for Decision Making.* West Publishing Co., 1992.

A Appendix

OPORD format:

- NUM(1 byte int: n) : number of ENEMY units.
- NUM(1 byte int: m) : number of FRIENDLY units.
- MISSION (1 byte int enumerate: DEFEND, ASSAULT) : mission to be accomplished by *issuing* unit. This is similar to mission of superior. Mission of adjacent unit has been omitted since it is included in the synchronization matrix (each row of the matrix describes the mission of different companies).
- NUM(1 byte int: s) : number of FRIENDLY subunits = rows in synchronization matrix.
- NUM(1 byte int: p) : number of phases = columns in synchronization matrix.
- BOUNDRY : area of unit's boundary.
- UNIT_DESCRIPTION : ENEMY unit description.
  - for each ENEMY unit i, where $1 \leq i \leq n$.
    - UNIT_ID (1 byte int: i) : enemy unit i.
    - LOCATION(4 byte int: X_i, Y_i, Z_i) : position of enemy unit i .
    - ECHELON_LEVEL (1 byte int enumerate: COMPANY, PLATOON): level of ENEMY unit.
    - ECHELON_TYPE (1 byte int enumerate: ARMOR, MECH_INFANTRY): type of ENEMY unit.
    - STRENGTH(1 byte int: %) : strength of enemy unit in terms of personnel and equipment.
    - OPERATIONAL ACTIVITY(1 byte int enumerate: DEFEND, ASSAULT) : current activity of enemy unit.
  - for each friendly unit j, where $1 \leq j \leq m$.
    - UNIT_ID (1 byte int: j) : friendly unit j.
    - LOCATION(4 byte int: X_j, Y_j, Z_j) : position of friendly unit j.
    - ECHELON_LEVEL (1 byte int enumerate: COMPANY, PLATOON): level of FRIENDLY unit.
    - ECHELON_TYPE (1 byte int enumerate: ARMOR, MECH_INFANTRY): type of FRIEN DLY unit.
    - STRENGTH(1 byte int: %) : strength of friendly unit in terms of personnel and equipment.
    - OPERATIONAL ACTIVITY(1 byte int enumerate: DEFEND, ASSAULT) : current activity of friendly unit.
  - the synchronization matrix (S x P).

* NUM(1 byte int: tk): number of tasks for subunit in phase.
* NUM(1 byte int: tn): number of next phases available.
* for each task k, where $1 \leq k \leq tk$.
  - NUM(1 byte int) : task identifier.
  - NUM(1 byte int enumerate: DEFEND, SEIZE, MOVE, FIRE, ASSAULT) : task ident ifier.
  - LOCATION(4 byte int: X_i, Y_i, Z_i) : e.g. destination of MOVE. .
* for each phase l, where $1 \leq l \leq tn$.
  - NUM(1 byte int) : next phase for subunit.
  - NUM(1 byte int enumerate: ON_ORDER, ON_OWN_INITIATIVE): condition for transitioning to the next phase.

SITREP format:

1. SITREP_TYPE (1 byte int enum) - conditions when SITREP is generated.
   - **Sighting** - when another *unit* has been sighted.
   - **Under_Fire** - when a *unit* is under fire.
   - **Damaged** - when a *unit* has been damaged (hit).
   - **Engaging** - when engaging a *target*.
   - **Moving** - when a *unit* has started to move to location.
   - **Arrival** - when a *unit* has arrived at a check point.
2. LOC (4 byte int: X,Y) - location of *unit* or *target*.
3. TIME (2 byte int) - the time when the condition occurred.
4. UNIT_LEVEL (ENUM_EGL_ECHELON_LEVELS) - level of *unit*.
5. UNIT_TYPE (ENUM_EGL_ECHELON_TYPES) - type of *unit*.
6. TIMESTAMP (2 byte int) - time when SITREP is generated.
7. DISTRBTN () - destination units to which this SITREP is being sent.
8. STRENGTH (1 byte float) - current remaining strength of unit damaged.
9. MY_LOC (4 byte int: X,Y)- location of the unit sending the SITREP.
10. ID (1 byte int enum) - ID of unit sending the SITREP.

If LOC and MY_LOC are identical, then the unit generating the SITREP is describing the situation of itself, otherwise the SITREP contains information about status of other units. The different types of SITREPs should be quite self-explanatory from the short description given above. The ID of unit sending the SITREP corresponds to the codewords used by individual units to identify themselves in actual combat. If STRENGTH is less than 5 % for a unit that

has been hit, it is considered to be killed.

Sample OPORD:

```
Number of Opfor Units : 2
Number of Friendly Forces Units : 1
Type of mission: SEIZE
Number of Friendly Subunits : 3
Number of Phases : 1 (Current version will only handle 1 phase)
Left Unit Boundary pt1 x,y : 55000 20000
Left Unit Boundary pt2 x,y : 55000 57500
Right Unit Boundary pt1 x,y : 20000 20000
Right Unit Boundary pt2 x,y : 20000 57500
Front Unit Boundary pt1 x,y : 55000 20000
Front Unit Boundary pt2 x,y : 20000 20000
Rear Unit Boundary pt1 x,y : 55000 57500
Rear Unit Boundary pt2 x,y : 20000 57500

Opfor Unit id : 1 ( Unit id will always start from 1 and increment by 1)
Opfor Unit Location x,y : 32500 28750
Opfor Unit Echelon-level : 7 (corresponds to PLATOON in EGL_ECHELON_ENUM)
Opfor Unit Echelon-type : 2 (corresponds to ARMOR in ECHELON_TYPE_ENUM)
Opfor Unit Strength : 70
Opfor Unit Operational Activity : DEFEND

Opfor Unit id : 2
Opfor Unit Location x,y :45000 46250
Opfor Unit Echelon-level : 7 (corresponds to PLATOON in EGL_ECHELON_ENUM)
Opfor Unit Echelon-type : 2 (corresponds to MECH_INF in ECHELON_TYPE_ENUM)
Opfor Unit Strength : 60
Opfor Unit Operational Activity : DEFEND

Friendly Unit_id : 1
Friendly Unit Location x,y : 32500 28750
Friendly Unit Echelon-level : 7 (corresponds to COMPANY in ECHELON_TYPE_ENUM)
Friendly Unit Echelon-type : 1 ( 1 MECH_INF and 2 ARMOR )
Friendly Unit Strength : 85
Friendly Unit Operational Activity : DEFEND

[ SYNCHRONIZATION MATRIX ] : Missions to Company Entities 1,2,3
SubUnit 1, Subphases 1, Num of tasks : 1
SubUnit 1, Subphases 1, Num of transitions : 1

SubUnit 1, Subphases 1, Task id : 1 (this is the order for the current CE)
SubUnit 1, Subphases 1, Task  : SEIZE
```

```
SubUnit 1, Subphases 1, Task loc x,y : 32500 28750
SubUnit 1, Subphases 1, Transition next phase : 1
SubUnit 1, Subphases 1, Transition code : ON_INITIATIVE

SubUnit 2, Subphases 1, Task id : 1 ( Task for other Company CE )
SubUnit 2, Subphases 1, Task  : SEIZE
SubUnit 2, Subphases 1, Task loc x,y : 38700 40500
SubUnit 2, Subphases 1, Transition next phase : 1

SubUnit 3, Subphases 1, Task id : 1 ( Task for other Company CE )
SubUnit 3, Subphases 1, Task  : MOVE
SubUnit 3, Subphases 1, Task loc x,y : 50000 40000
SubUnit 3, Subphases 1, Transition next phase : 1
SubUnit 3, Subphases 1, Transition code : ON_ORDER
```

B Biographies

Jin Joo Lee received a B.S. degree in Computer Science from Ewha University, Korea in 1988 and a M.S. degree in Computer Science from Brown University in 1991. After receiving the M.S. degree, she was a research engineer at Human Computers Inc., Korea until 1992. She is a currently a PhD student at the Computer and Information Sciences department at University of Florida. Her research interests are in AI planning, simulation and control.

Paul A. Fishwick is an associate professor in the Department of Computer and Information Sciences at the University of Florida. He received the BS in Mathematics from the Pennsylvania State University, MS in Applied Science from the College of William and Mary, and PhD in Computer and Information Science from the University of Pennsylvania in 1986. He also has six years of industrial/government production and research experience working at Newport News Shipbuilding and Dry Dock Co. (doing CAD/CAM parts definition research) and at NASA Langley Research Center (studying engineering data base models for structural engineering). His research interests are in computer simulation modeling and analysis methods for complex systems. He is a senior member of the IEEE and the Society for Computer Simulation. He is also a member of the IEEE Society for Systems, Man and Cybernetics, ACM and AAAI. Dr. Fishwick was chairman of the IEEE Computer Society technical committee on simulation (TCSIM) for two years (1988-1990) and he is on the editorial boards of several journals including the ACM Transactions on Modeling and Computer Simulation, IEEE Transactions on Systems, Man and Cybernetics, The Transactions of the Society for Computer Simulation, International Journal of Computer Simulation, and the Journal of Systems Engineering.