Automated Fire Support Planning for Combat Simulations

Byron R. Harder and Chris Darken

Naval Postgraduate School, MOVES Institute,
1 University Cir., Monterey, CA 93943, USA
{brharder1,cjdarken}@nps.edu
http://www.movesinstitute.net

Abstract. Today's constructive combat simulations are not capable of automatically generating realistic battle plans. A contributing cause for this is the lack of a fire support planning algorithm to support a given maneuver plan. We present a methodology for modeling the fire support planning problem, which assumes the availability of a numerical definition of tactical risk. The solution space appears very hard to search exhaustively, so we offer a fire support planning algorithm that can generate a reasonable although sub-optimal plan in polynomial time.

Keywords: Automated planning · Combat modeling · Military simulations

1 Introduction

Today's constructive combat simulations are not capable of automatically generating realistic battle plans. This makes them quite non-adaptive in the human decision-centric arena of warfare. Furthermore, the need to use human experts for this task is a bottleneck for many simulation applications. We propose here an incremental improvement to the status quo with the potential to improve the robustness of analytical and training simulations. The method is informed by tactical science as embodied in military doctrine [1], [2].

1.1 The Combat Simulation Domain

We focus here on tactical ground combat simulations for brigade and smaller units. These units must travel across the ground; their movement, visibility, and weapons are affected by terrain elevation and type. Decisions of an entity or unit should represent not just instincts, such as aggressiveness and survival, but also the tasks assigned to it by its commanding unit's plan [3].

The focus of this paper is *deliberate attack* operations, a subset of offensive operations where the attacker has reasonable knowledge (military intelligence) of enemy defensive positions and capabilities. Good information about the enemy should allow the attacker to devise an effective plan, but actually generating such a plan involves some work. Human commanders and staffs generate plans using

spatial and temporal reasoning as well as heuristics developed through military experience. It turns out that mimicking this in software is quite a challenge.

1.2 Fire Support

We distinguish between *maneuver* and *fire support* units. Maneuver units are those whose main purpose is to move in relation to the enemy to gain tactical advantage. An attack often involves an *assault*, where a maneuver unit forcefully occupies a position held by an opposing unit, forcing its retreat or defeat. Fire support units are those meant to engage at longer range with special-purpose weapons such as machine guns, artillery, or anti-armor missiles. We consider support units with the capability for either direct fire (such as machine guns) or indirect fire (such as mortars). Maneuver units can provide fire support when not given a maneuver task.

A good attack plan takes advantage of military intelligence and terrain to avoid enemy fields of fire, but a good defensive plan will force the attacker to expose units. At some point, the attacker must depart cover. The fire support plan limits the enemy capability during those exposed times through suppression fires.[1] Some fire support units may need to move to new positions, which takes time, in order to fire on their assigned targets.

We assume that the maneuver portions of the plan have already been developed down to specific routes across the terrain—a nontrivial task, but separate from this work. Our problem is to generate the fire support plan: a set of movement and firing orders that sufficiently supports the overall mission objective.

2 Related Work

A small number of commercial combat simulations (used for both training and games) can produce terrain- and enemy-aware movement orders to achieve the kind of maneuver planning described above. These include the Command Ops 2 game engine [4], which models from the division down to the maneuver company and support platoon level, and the PlannedAssault scenario generation tool [5], which plans from the approximate company level down to squads or smaller teams. Both of these systems generate complementary maneuvers for multiple subordinate units and call artillery missions on opposing units once sighted during execution. However, they do not appear to automatically plan suppression missions in support of specific maneuvers in the manner that we propose.

A related research thrust seeks to generate or analyze courses of action for real-world tactical missions. Battlespace Terrain Reasoning and Analysis [6] and the Computational Military Tactical Planning System [7] use evolutionary algorithms to optimize maneuver choices according to a mission-specific objective function. SimPath [8] uses qualitative process theory to analyze given courses of

[1] Fire support can accomplish other objectives, such as obscuration and destruction, but the focus here is on suppression.

action. These types of systems deal with fire support in a manner that is too abstract or implicit to integrate with entity-level simulations.

Tactical pathfinding is foundational for the techniques described below and is used in several of the aforementioned systems. Path planning while avoiding enemy visibility is described in the work of Rowe and Lewis [9]. Additional techniques, such as accounting for target acquisition time and observer mobility, are explained by van der Sterren [10]. Cooperative pathfinding [11] is a closely related problem, useful for keeping units well-dispersed. More recently, the MECH framework is used to reason about risk to a route based on parameters such as visibility [12]. The problem at hand differs from these approaches by allowing the risk calculation to be modified by fire support tasks.

We build upon a method for planning a single suppression task [13], extending to a collection of fire support providers and allowing movement to new positions. Our approach aims to be more holistic by reasoning about the reduction of risk to the friendly force's mission.

3 Problem Formulation

We use the term *unit* to refer to any group of one or more simulated combatants that moves and fights together as a single formation. A unit moves by following a *route*, which is defined by a series of waypoints, or locations in three-dimensional space. Units travel in straight lines along the terrain skin toward their next waypoint at a realistic, known speed for the terrain type. For this problem, the routes for tasked maneuver units are predetermined and fixed. Untasked maneuver units are considered fire support units.

3.1 Segments and Intervals

We will mark each unit's route with enemy visibility regions. We define a *route segment* as the portion of a route between any pair of location coordinate vectors $(\boldsymbol{c}, \boldsymbol{c'})$ that lie on it. A *risk segment* is annotated as the tuple $(e, f, \pi, \boldsymbol{c}, \boldsymbol{c'})$, where enemy unit e is capable of damaging friendly unit f as it traverses the segment of route π from $\boldsymbol{c}$ to $\boldsymbol{c'}$ (see Fig. 1). Each risk segment r has an associated value $\Psi(r)$ equal to the expected losses to f caused by e during f's traversal of r. We must have some way to calculate or estimate this expected value.

As Fig. 1 illustrates, risk segment endpoints do not have to coincide with the route's waypoints. If precision is needed, we can calculate the intersections of routes with visibility volumes following the approach of Rowe and Lewis [9].

It is often convenient to refer to time intervals rather than geometric segments. We assume that combat tasks are scheduled for specific times, so we can map between risk segments $(e, f, \pi, \boldsymbol{c}, \boldsymbol{c'})$ and risk intervals (e, f, π, t, t').

If enemy unit e has a choice of several targets during some time interval, we will plan as if e equally distributes firepower among all simultaneous targets. This avoids separate consideration of all possible combinations of target choices, which might require an exponential number of scenarios. Even if we have access to the

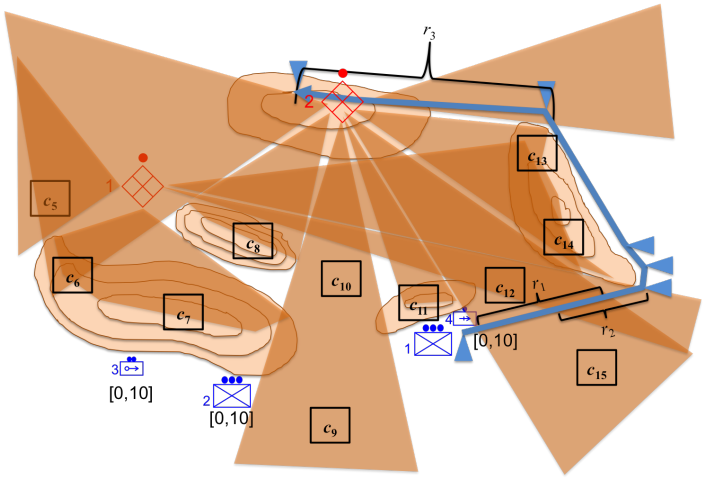

Fig. 1. Friendly Unit 1 is tasked to follow a route consisting of six waypoints, marked with triangles. Enemy Unit 1 (diamond-shaped symbol) can damage Friendly Unit 1 during risk segment r_2. Enemy Unit 2 can damage it during r_1 and r_3. The [0,10] markers show availability intervals for fire support resources, and the boxes marked $\boldsymbol{c_i}$ are possible fire support locations. Both are described later in the text.

enemy targeting logic, and even if it can be simply described, planning against such logic can become quite complex. For example, it often depends on target range, which varies as friendly units move in different directions along nontrivial routes. Our enemy targeting model causes the planner to simultaneously consider multiple possibilities, just as many human military planners tend to think about threats.

3.2 Applying Fire Support Tasks

A *fire support task* is an order for friendly unit f to move and emplace (during $[t_1, t_2]$), and then fire ($[t_2, t_3]$) from position $\boldsymbol{c}$ on enemy unit e. We denote the complete task as the tuple $(f, \boldsymbol{c}, e, t_1, t_2, t_3)$.[2]

Let W be the (achievable) set of fire support tasks assigned to friendly units, and define $\Phi(r, W)$ as the *residual risk value* for risk segment r given W. This function must be implemented for the specific combat model under study. Note that Φ and Ψ are related; it is always the case that $\Phi(r, \emptyset) = \Psi(r)$, but additional relationships depend on the combat model's rules for interacting tasks. For a particular task $w \in W$, we will need its *total risk reduction* value:

$$\Delta_w(W) = \sum_r (\Phi(r, W - \{w\}) - \Phi(r, W)) \tag{1}$$

[2] The route for f to get to $\boldsymbol{c}$ is left implicit for brevity.

Intuitively, Δ_w gives us the "partial derivative" with respect to w. But note that the benefits of fire support tasks may not be independent. For example, suppressing a single target with two identical assets is probably not twice as effective as suppression with just one of them. Suppression effects are likely to be nonlinear and discontinuous: they involve models of fear and perception. Furthermore, since each fire support task can introduce additional movement orders, we could easily introduce new risk segments and make the plan worse.

3.3 Fire Support as a Resource

If the unit f is initially available (no task assigned) over the interval $[t_0, t_4]$, the assignment of a fire support task w makes that unit unavailable for a subinterval $[t_1, t_3]$. But f is still available during $[t_0, t_1]$ and $[t_3, t_4]$. To represent an available fire support resource—a unit with no task at some location for some time—we use a special "empty" fire support task called an *availability task* of the form $(f, \boldsymbol{c}, 0, t_0, t_0, t_4)$. The initial plan W_0 has only availability tasks. Each time we assign a task $w = (f, \boldsymbol{d}, e, t_1, t_2, t_3)$ to a resource $a = (f, \boldsymbol{c}, 0, t_0, t_0, t_4) \in W$, we update W using the following operator:

$$\gamma(W, a, w) = W - \{a\} \cup \{w, (f, \boldsymbol{c}, 0, t_0, t_0, t_1), (f, \boldsymbol{d}, 0, t_3, t_3, t_4)\} \tag{2}$$

If $t_0 = t_1$ or $t_3 = t_4$, the corresponding resource can be ignored. Note that we are not restricted to forward or backward planning, and tasks for different units can overlap in time or share the same target.

3.4 An Optimization Problem

The above notation allows us to state fire support planning as an plan optimization problem: what set W^* of fire support task tuples, formed by γ operations from initial state W_0, best minimizes the total risk value? Or, if we have a maximum acceptable risk value, we can ask whether a goal plan W_g exists.

Searching this optimization space is hard. Even with finite but practical bit lengths for locations and times, the search space is vast, so we turn to a heuristic approach using modern tactical intuition.

Unfortunately, the best fire support plan may consist of several tasks that work poorly alone ($\Delta_w(W_0 \cup \{w\}) < 0$), but that provide mutual support once they are all added to the plan. We deal with this by aggressively adding tasks even when $\Delta_w(W) < 0$, expecting eventual benefits from cooperating tasks.

4 A Greedy Algorithm

We offer the following greedy algorithm to generate a fire support plan. It is, of course, neither optimal nor complete given the optimization and satisfaction interpretations above, but it can produce a reasonable plan efficiently.

For each resource, this algorithm maintains on its search frontier (the sets N and M) tasks to suppress each enemy unit. On each pass through the main loop,

it chooses the task in $N \cup M$ that best reduces the total risk. To limit complexity, potential tasks requiring no movement (those in N) are preferred. Each time a task w is added to the current plan W, we remove from N and M all of the siblings of w generated from the same availability task. We then generate new options for N and M from the new (shorter) availability tasks given by γ.

The algorithm terminates either when the total risk is sufficiently small or when no task options remain. If we assume that fire support tasks have a minimum useful duration ϵ, such as the time to fire a single shot, then each choice of a task removes a resource of duration t and replaces it with two whose summed duration is $t - \epsilon$ or less. Since the sum of all resource durations decreases by at least ϵ per loop iteration, we will eventually have a sum less than ϵ, which can only be true if there are no tasks in the frontier, terminating execution.

Algorithm 1 PlanFireSupport (R : set of maneuver plan risk intervals)

Require: targetValue is the maximum acceptable total risk value, defaulting to 0.0
  $(M, N) \leftarrow (\emptyset, \emptyset)$ //will hold tentative tasks (requiring/not requiring movement)
  $W \leftarrow$ set of availability tasks for all fire support units
  for all $a \in W$ **do**
    GenerateTaskOptions(a) and add them to (M, N) as appropriate
  loop
    if $N \neq \emptyset$ **then**
      $w \leftarrow \text{argmax}_{n \in N}(\Delta_n(W \cup \{n\}))$
    else if $M \neq \emptyset$ **then**
      $w \leftarrow \text{argmax}_{m \in M}(\Delta_m(W \cup \{m\}))$
    else
      return W //a non-satisficing but reasonable plan
    $a \leftarrow$ the availability task in W from which w was generated
    $N \leftarrow N - \{n \in N | n$ was generated from availability task $a\}$
    $M \leftarrow M - \{m \in M | m$ was generated from availability task $a\}$
    $W \leftarrow \gamma(W, a, w)$
    a_1, a_2 are the (possibly null) new resources added by γ
    GenerateTaskOptions(a_1) and add them to (M, N)
    GenerateTaskOptions(a_2) and add them to (M, N)
    $R_w \leftarrow$ set of all risk intervals on the new routes for w
    for all (b, r) where $b \in W$ is an availability task **and** $r \in R_w$ **do**
      GenerateTask(b, r) and add it to M or N as appropriate
    $R \leftarrow R \cup R_w$
    if $\sum_{r \in R}(\Phi(r, W)) \leq$ targetValue **then**
      return W //a satisficing plan

The most important job of the GenerateTaskOptions and GenerateTask subroutines is finding good firing positions. The goodness of a position is measured both by the ability to engage a potential target from it and the quality (both speed and protection) of paths to get in and out. Our approach to this is based mainly on the A* tactical pathfinding algorithm [10] with the following modi-

fication. Rather than search for a path to a single, predetermined point, each fire support resource (represented by an availability task) seeks paths to enough different positions so that it can potentially engage any threat. If that unit has a subsequent fire support task $(f, \boldsymbol{c}, e, t_1, t_2, t_3) \in W$, it must also find a point-to-point path from each new position to $\boldsymbol{c}$. Only one position and path (or pair of paths) may eventually be selected per availability task, but the variety of positions gives different tactical options to evaluate. This approach is exhaustive in the sense that every resource has an option to engage every threat if possible, given its movement speed and the durations of the maneuver plan's risk intervals. However, it is also minimal in the sense that pathfinding stops once we have at least one position from which to engage each threat. This means that the starting positions of the fire support resources make a difference in the planning results.

Pathfinding tends to be the most costly operation in this kind of planning, so we employ a few efficiency techniques. Path costs are a combination of time and risk, a tradeoff that must be experimentally tuned. Given fixed enemy positions, we can perform a preprocessing step to tag all locations in the navigation graph with their valid targets. This allows us to use an A* heuristic even though we are searching for multiple positions. We avoid recalculating paths when W is updated. Each update can only decrease the risk of such a route, so this shortcut is relatively safe. We can also gain some efficiency by employing the coroutine pattern [14], saving the state of the pathfinder each time it yields a route.

An upper bound for the time complexity of PlanFireSupport[3] is $O(k^5 s^3 v^3)$, where k is the number of friendly and enemy units in the simulation, s is the maximum number of tasks per fire support unit, and v is the number of vertices in the navigation graph. If T is the duration of the maneuver plan in simulated time, and ϵ is the minimum task duration (as above), then $s \leq \frac{T}{\epsilon}$. Most realistic plans only need a small number for s—even if jagged obstacles or terrain create many small risk intervals, a plan with many fire support tasks of short duration would not be a reasonable model of a human-generated fire support plan. Since the formulation of the problem resembles general continuous time planning [15], we can be somewhat satisfied with a polynomial running time. Our initial implementation completes planning in no more than few seconds when run with seven units.

5 Example of Execution

We illustrate the algorithm using the scenario in Fig. 1. The friendly force consists of maneuver units 1 and 2 and two fire support units: mortar section 3 and machine gun squad 4. They are tasked with seizing the hill occupied by enemy unit 2. The pre-computed maneuver plan has unit 1 attempting a right flank attack, which takes 10 units of time and has 3 risk segments (r_1, r_2, r_3). The other friendly units are untasked and therefore considered fire support assets

[3] We have remained agnostic about the combat model, but the complexity analysis requires some implementation assumptions not detailed here.

with availability intervals [0,10]. $\boldsymbol{c_i}$ is the initial location of each friendly unit i, and we have generated additional locations $\boldsymbol{c_5}$ through $\boldsymbol{c_{15}}$. The starting state is:

$N = M = \emptyset$
$W = \{(\text{unit2}, \boldsymbol{c_2}, 0, 0, 0, 10),\ (\text{unit3}, \boldsymbol{c_3}, 0, 0, 0, 10), (\text{unit4}, \boldsymbol{c_4}, 0, 0, 0, 10)\}$

First, PlanFireSupport calls GenerateTaskOptions on each member of W (this does not change W):

$N = \{(\text{unit3},\ \boldsymbol{c_3},\ \text{enemy2},\ 0.2, 0.2, 1.5),\ (\text{unit3},\ \boldsymbol{c_3},\ \text{enemy1},\ 1, 1, 2),$
$(\text{unit3},\ \boldsymbol{c_3},\ \text{enemy2},\ 5, 5, 10)\}$
$M = \{(\text{unit2},\ \boldsymbol{c_9},\ \text{enemy2},\ 0, 1.2, 1.5),\ (\text{unit2},\ \boldsymbol{c_9},\ \text{enemy2},\ 3.7, 5, 10),$
$(\text{unit2},\ \boldsymbol{c_7},\ \text{enemy1},\ 0, 1.5, 2),\ (\text{unit4},\ \boldsymbol{c_{11}},\ \text{enemy2},\ 0, 1, 1.5),$
$(\text{unit4},\ \boldsymbol{c_{11}},\ \text{enemy2},\ 4, 5, 10),\ (\text{unit4},\ \boldsymbol{c_{12}},\ \text{enemy1},\ 0.3, 1, 2)\}$

In the first iteration of the main PlanFireSupport loop, N is nonempty, so the algorithm selects its member with the best Δ score: (unit3, $\boldsymbol{c_3}$, enemy2, 5, 5, 10). γ inserts this task in W, removes the original availability task for unit3, and creates two new availability tasks, a_1 and a_2, with the remaining time intervals. Finally, GenerateTaskOptions(a_1) produces new possible tasks, which turn out to be the same as they were before. Now we have:

$W = \{(\text{unit2},\ \boldsymbol{c_2}, 0, 0, 0, 10),\ (\text{unit3},\ \boldsymbol{c_3}, 0, 0, 0, 5),\ (\text{unit4},\ \boldsymbol{c_4}, 0, 0, 0, 10),$
$(\text{unit3},\ \boldsymbol{c_3},\ \text{enemy2},\ 5, 5, 10),\ (\text{unit3},\ \boldsymbol{c_3}, 0, 10, 10, 10)\}$
$N = \{(\text{unit3},\ \boldsymbol{c_3},\ \text{enemy2},\ 0.2, 0.2, 1.5),\ (\text{unit3},\ \boldsymbol{c_3},\ \text{enemy1},\ 1, 1, 2)\}$
M : no change

Since N is still nonempty, the next iteration works similarly, moving (unit3, $\boldsymbol{c_3}$, enemy2, 0.2, 0.2, 1.5) to W. But then, since only the interval [1.5, 2] is available, GenerateTaskOptions produces a shorter-length task. We now have $N = \{(\text{unit3},\ \boldsymbol{c_3},\ \text{enemy3},\ 1.5, 1.5, 2)\}$. This final task in N is chosen on the next iteration, leaving the set empty.

The algorithm now considers M and finds that (unit4, $\boldsymbol{c_{12}}$, enemy1, 0.3, 1, 2) has the highest Δ value. This task has introduced two new risk segments, r_4 and r_5: the portions of unit4's route to $\boldsymbol{c_{12}}$ where each enemy unit has a field of fire. Tasks to deal with them are now added to the frontier for each available asset—in this case, only unit2. Here is the planner state after this iteration (also see Fig. 2):

$W = \{(\text{unit4},\ \boldsymbol{c_4}, 0, 0, 0, 0.3),\ (\text{unit2},\ \boldsymbol{c_2}, 0, 0, 0, 10),$
$(\text{unit3},\ \boldsymbol{c_3},\ \text{enemy2},\ 0.2, 0.2, 1.5),\ (\text{unit4},\ \boldsymbol{c_{12}},\ \text{enemy1},\ 0.3, 1, 2),$
$(\text{unit4},\ \boldsymbol{c_{12}}, 0, 0, 2, 10),\ (\text{unit3},\ \boldsymbol{c_3},\ \text{enemy1},\ 1.5, 1.5, 2),$
$(\text{unit3},\ \boldsymbol{c_3},\ \text{enemy2},\ 5, 5, 10)\}$
$N = \{(\text{unit4},\ \boldsymbol{c_{12}},\ \text{enemy2},\ 5, 5, 10)\}$

$$M = \{(\text{unit2},\ \boldsymbol{c_9},\ \text{enemy2},\ 0, 1.2, 1.5),\ (\text{unit2},\ \boldsymbol{c_9},\ \text{enemy1},\ 0, 1.2, 2),$$
$$(\text{unit2},\ \boldsymbol{c_9},\ \text{enemy2},\ 0, 1.2, 2),\ (\text{unit2},\ \boldsymbol{c_9},\ \text{enemy2},\ 3.7, 5, 10),$$
$$(\text{unit2},\ \boldsymbol{c_7},\ \text{enemy1},\ 0, 1.5, 2)\}$$

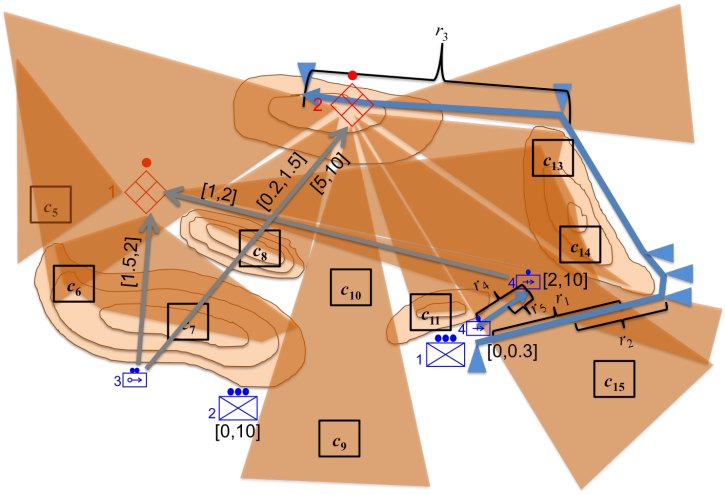

Fig. 2. Partial fire support plan generated by a few iterations of the algorithm

6 Conclusion and Future Work

We have presented a methodology for modeling the fire support planning problem using a numerical definition of tactical risk to a supported maneuver plan. The formulation is with respect to time and position, and the solution space appears very hard to search exhaustively. To deal with this, we have developed a greedy fire support planning algorithm that seeks a balance between utility and running time; it completes in time polynomial in the number of units, maximum number of tasks per unit, and terrain vertices.

As of this writing, we have just completed the initial implementation of this algorithm in a combat simulation environment. Limited experimentation has already demonstrated that the algorithm can generate reasonable plans, and the effort has provided more insight about aspects such as pathfinding parameters and the numerical risk and risk reduction functions (Ψ and Φ). We plan to test a wider variety of terrains, force compositions and sizes, and starting conditions. We are interested in both the computational efficiency of the implementation and the validity of its output, since both are critical in an analyst's or trainer's decision to use this kind of tool.

A general battle planning algorithm needs more than the fire support component we offer here. It must generate the maneuver plan (we assumed it already

existed), operate at multiple command echelons, reason about terrain with respect to large and small units, and react to unpredictable combat results, just to name a few. We hope to advance the state of the art for those tasks as well.

References

1. Department of Defense. Marine Corps Doctrinal Publication 1-3: Tactics (1997)
2. Department of Defense: Army Doctrinal Publication 3-90: Offense and Defense (2012)
3. Pittman, D.: Command Hierarchies using Goal-Oriented Action Planning. AI Game Programming Wisdom 4, ed. Rabin, S., Course Technology, pp. 383–391. Boston, MA (2008)
4. Panther Games. Lock 'n Load Publishing, `http://www.panthergames.com/`
5. van der Sterren, W.: PlannedAssault. `http://www.plannedassault.com`
6. Schlabach, J.: Cognitive Amplification for Contextual Game-Theoretic Analysis of Courses of Action Addressing Physical Engagements. U.S. Patent Application 12/504,077 (2010)
7. Kewley, R. H., Embrechts, M. J.: Computational Military Tactical Planning System. IEEE Transactions on Systems, Man, and Cybernetics 32, no. 2, pp. 161–171 (2002)
8. Hinrichs, T., Forbus, K., de Kleer, J., Yoon, S., Jones, E., Hyland, R, Wilson, J. Hybrid Qualitative Simulation of Military Operations. In: Proceedings of the Twenty-Third Innovative Applications of Artificial Intelligence Conference, pp. 1655-1661. San Francisco, CA (2011)
9. Rowe, N. C. and Lewis, D. H.: Vehicle Path-Planning in Three Dimensions using Optics Analogs for Optimizing Visibility and Energy Cost. In: Proceedings of the NASA Conference on Space Telerobotics. Available: `http://faculty.nps.edu/ncrowe/spacesymp2.htm`. Pasadena, CA (1989)
10. van der Sterren, W. Tactical Path-Finding with A*. Game Programming Gems 3, ed. DeLoura, M., Course Technology, pp. 294-306. Boston, MA (2002)
11. Silver, D.: Cooperative Pathfinding. In: Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 117-122. Marina del Rey, CA (2005)
12. Wang, X., George, S., Lin, J., Liu, J.: Quantifying Tactical Risk: A Framework for Statistical Classification Using MECH. SBP 2015, LNCS 9021, pp. 446–451. (2015)
13. Straatman, R., van der Sterren, W., Beij, A.: Killzone's AI: Dynamic Procedural Combat Tactics. In: Proceedings of the 2005 Game Developers Conference. San Francisco, CA (2005)
14. Conway, M.E. 1963: Design of a separable transition-diagram compiler. Communications of the ACM, vol. 6, no. 7, pp. 396-408 (1963)
15. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann Publishers, pp. 281-374. San Francisco, CA (2004)