A Probabilistic Planner for the Combat Power Management Problem

Abder Rezak Benaskeur
Defence R&D Canada - Valcartier
Québec, QC G3J 1X5, Canada
abderrezak.benaskeur@drdc-rddc.gc.ca

Froduald Kabanza and **Eric Beaudry** and **Mathieu Beaudoin**
Menya Solutions
Sherbrooke, QC J1H 47B, Canada
{kabanza, eric.beaudry, mathieu.beaudoin}@menyasolutions.ca

Abstract

We present a planner for the Combat Power Management (CPM) problem. In response to multiple simultaneous or sequential threats, the planner generates a set of local plans, one for each target considered apart, and then merges them by searching the space of global plans. The proposed plan merging solution serves also as an iterative plan repair process that resolves negative interferences (subaddivity) and exploits synergistic effects (superaddivity) among activities. The planner was developed as a component of a naval Command and Control system to support the warship command team in defending against Anti-Ship Missile threats. The planner is particularly suited for domains characterized by durative concurrent actions, with both superadditive and subadditive interactions, and probabilistic effects. It is implemented in a generic way, allowing it to address other application domains. We discuss the CPM application domain, describe the planner, and present experimental results.

Introduction

During military naval operations, warships may be exposed to a variety of threats, including coordinated raids of advanced ballistic and cruise Anti-Ship Missiles (ASMs). To defend against these threats, the ship commanding team relies on different kinds of combat resources. Determining an efficient plan for the application of these resources against threats, particularly in the case of multiple-threat engagements, is a very complex decision-making process, in which a huge amount of imperfect data must be dealt with, under the pressure of time. This defines the problem of Combat Power Management (CPM), also known as Weapon Allocation (WA). This problem is characterized by durative concurrent actions having time-dependent probabilistic effects, competing for limited combat resources. This paper presents the COmbat Resource ALlocation System (CORALS) planner designed to recommend engagement plans to the commanding team of warships during cognitively intense Anti-Ship Missile Defence (ASMD) operations.

Although the motivating application of CORALS is CPM, its generic implementation makes it potentially exploitable in several others domains requiring planning with resources, such as real time strategy games (e.g., Microsoft Age of Empires, Blizzard Entertainment's Starcraft). These video games include combat phases in which player and non-player characters must give orders to offensive and defensive units to confront the opponents. A unit can use different attacks or abilities with varying time costs, and effectiveness which depends on the engaged enemy unit. The implementation of CORALS is generic enough to enable its application in such games to decide the optimal offensive or defensive actions for non-player characters.

Different studies have shown that experienced operators are competent in responding efficiently to multiple sequential threats. However, in situations involving massive raids with threats arriving in quick succession, human decisions tend to be suboptimal after a certain number of threats (Bos et al. 2005; Ousborne 1993). In order to permit operators to optimally decide and engage actions against given targets rather than responding based solely on intuition and rules of thumb, advanced CPM decision support capabilities are required. This is one of the objectives of the Innovative Naval Combat Management and Decision Support (INCOMMANDS) Technology Demonstration Project funded by Canada Department of National Defense. INCOMMANDS aims at demonstrating advanced decision support capabilities to support operators during ASMD operations. One of these capabilities is the CORALS planner, presented in this paper.

The remaining part of the paper is organized as follows. In the next section, we discuss related work. Then we present the CPM application domain. We follow with a description of the planner, experiments, and finally conclude with a perspective on future work.

Related Work

The CPM problem has been abundantly studied, mostly in the operational (OR) research literature and, to a less extent, in the AI planning and scheduling literature. There are generally two formulations of the problem: static and

dynamic. The static formulation consists in finding pairing weapons/targets that maximizes the expected survival value of the defending platform. The exact scheduling of weapons launch is not considered. That is, the sequential decision-making nature of the problem is ignored. Even with this simplifying assumption, the problem is known to be NP-Complete (Lloyd and Witsenhausen 1986).

An interesting survey of early OR approaches, back in the 1970s, is given in (Matlin 1970). Most of these approaches were based on linear integer programming, with few considering heuristic search. More recent approaches still use the same basic techniques but rely on better implementations (Ahuja et al. 2003; Bohachevsky and Johnson 1993; Hadj-Alouane, Bean, and Murty 1999). The fact that the CPM problem has been to date mostly studied in the OR literature is not surprising. Indeed, dealing with resources and durative actions is very common in the OR literature, and there exist efficient scheduling algorithms use heuristics to maximize various types objective functions other than makespan (Brucker and Knust 2006). However, OR approaches have so far been essentially limited to the static formulation of CPM problem. The very few exceptions include (Hosein and Athans 1990), who are among the first authors to consider formally the sequential aspect of the dynamic CPM problems (Hosein and Athans 1990). They used a dynamic programming approach that did not handle durative actions. In a similar vein, AI planning approaches have been proposed to solve the problem as a Markov Decision Planning (MDP) problem (Bertsekas et al. 1999; Meuleau et al. 1998). These approaches did not either handle durative actions.

An in-depth analysis of the dynamic CPM problem suggested to us that one of the keys to being able to solve it efficiently would be to reason explicitly about actions, that is, about their preconditions and effects and how they interfere with each other. Action reasoning capability is more a feature of artificial intelligence (AI) planning approaches than OR approaches, hence we opted for an AI planning approach to the dynamic CPM problem. There exist few probabilistic planners in the AI planning litterature that can handle concurrent durative actions, and which may a priori be applicable to the CPM problem. Three of the most renowned are Tempastic (Younes and Simmons 2004), CPTP (Mausam and Weld 2006), and FPG (Buffet and Aberdeen 2006). Tempastic follows an event-based modeling that does not naturally fit the CPM domain. CPTP is limited to a small number of actions.[1] FPG presents a planner that, among the three, best meets the requirements of the CPM domain. However it does not permit probabilistic effects that depend on time, which is a crucial requirement in the CPM domain. For instance, the probability of a missile hitting its target depends on the distance at which interception will happen. To allow for comparison of CORALS and FPG, probabilistic effects were represented by constants (this assumption makes sense, for example, in war games that do not have to emulate naval warfare perfectly). With this simplified version of the CPM problem, CORALS outperformed FPG.

[1]We contacted the authors of CPTP, the implementation of which was not cleaned for public delivery. Its cousin CoMDP was available, but does not handle interwoven decision epochs.

The Combat Power Management Domain

State Variables

A state in the CPM domain (also called a tactical picture) describes the position of the defending ship, a list of targets (including their range, bearing, speed, class, and identification), and the availability and the status of own combat resources. The information about targets is acquired though various dissimilar and imperfect sources and is subject to uncertainty and deception. The uncertainty inherent to the state is omitted during the planning phase and dealt with during execution monitoring and replanning. However, uncertainty related to action outcome is directly accounted for during the planning phase.

Given the uncertainty characterizing the state, a planner that searches through a belief space would have been a natural choice. Instead, we opted for a planning approach that reflects the line of thinking of human planners and considers a completely defined initial state. Figure 1 illustrates an example of an initial state where three threats (with different types, bearings and ranges) are attacking a defending ship.

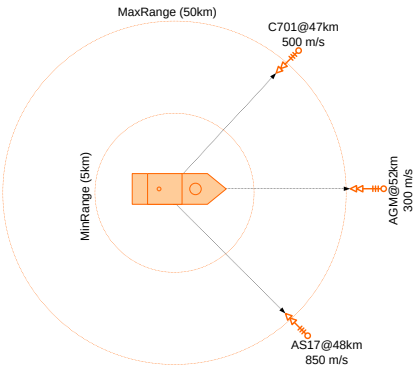

Figure 1: State (Tactical Picture)

Adopting a planning approach that reflects the line of thinking of military experts was one of the key arguments in convincing them of the relevance of an automated planning solution. Another key argument, and possibly the most convincing, was the experimental results demonstrating the capability of the planner to generate timely and efficient plans. Based on this, a recommendation was made to integrate a modified version of CORALS into a naval ASMD tool. Given the sensitive nature of the tool and the integration, we cannot give any detail on this integration. Here we report on the planning algorithm and its performance as a simple standalone and unclassified application.

Resources

Combat resources can be consumable (*e.g.*, each action consumes a fix amount out of a limited quantity) or renewable (*e.g.*, when being used, it is temporarily unavailable, but is released afterwards). The scenarios used in experiments reported here are based on the following resources:

- Hardkill weapons are used to intercept targets and actively destroy them through direct impact or explosive detonation in the proximity. They include Evolved Sea Sparrow Missile (ESSM) launchers; Intermediate Range Gun (IRG); Close-In Weapon System (CIWS).
- Softkill resources use different techniques to deceive or disorient the target to cause it to destroy itself, or at least to lose its lock on the ship. Chaff decoy launchers are used to seduce or distract targets, while radar jammers are used to perturb the enemy's radar.
- Separate Tracking and Illuminating Radars (STIRs) are fire control radars used to track targets being engaged by ESSMs or IRG.
- Ammunitions: ESSMs for the missile launchers, rounds for the guns and chaff.

The launchers and the STIRs are renewable resources. The ESSMs, chaffs and gun rounds are consumable resources. In more complex scenarios, a warship can in addition use deterrence measures to dissuade a target (*e.g.*, warning it by radio or locking tracking radar on it) or employ navigation (*e.g.*, to minimize radar signature, or to reduce the expected damage in case of likely hits).

Constraints

There are different kinds of constraints on deployment of combat resources. For instance, the number of launchers for ESSM or chaffs, as well as the number of guns, is limited. Resources also have different ranges and engagement regions, with the possibility of blind zones (e.g., the ship may have to be oriented in a certain direction to be able to use a certain resource against a specific target). Consumable resources are in limited quantities. On the other hand, some resources have to be used in combination with others (e.g., when an ESSM is launched, there must be a STIR tracking the target during the missile flight). Hence, the number of concurrent engagements by ESSMs is constrained both by the number of available ESSM launchers and the number of available supporting STIRs.

Actions

An action is defined as the application of one or a combination of resources against one target. For instance, firing an ESSM against a target is an action. Deploying a STIR to track a target is another action. Alternatively, one could model both actions as one of firing a missile, which requires both the ESSM and STIR resources. For this particular case, conflicts are better handled by separating them because in general missile and target track radars are not deployed simultaneously; their deployments rather overlap.

Associated to each action is a Probability of Successful Engagement (P_{se}). For a specific resource, the P_{se} may depend on several factors, including interception range, target type, target speed and weather condition. Figure 2 shows an example of an unclassified P_{se} (as a function of the target range) for an AGM84-type ASM target engaged by the ESSM. The graphics (small vertical dashed lines) also show

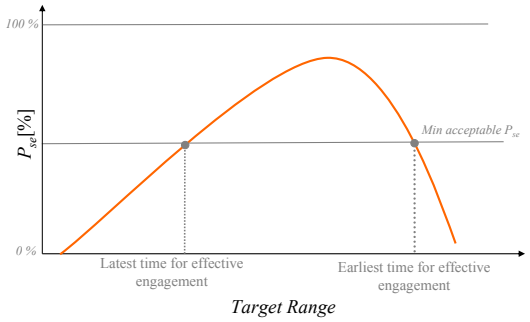

Figure 2: Example of P_{se} for ESSM

the minimum and maximum range for an effective engagement. Other factors besides the range may affect the action outcome (and the P_{se} curve). Those factors are not modeled explicitly and are treated as contingencies by plan execution and monitoring. For the experiments we used unclassified P_{se} data provided by naval officers, emulating the main features of actual data as far as the algorithmic aspects of the planner are concerned.

To illustrate, the action 'Fire-ESSM' along with the problem definition is specified in PDDL as follows:

```
(:domain
 (:durative-action Fire-ESSM
  :parameter(?target - t)
  :duration(=
    (/ (-(detectrange t) (*(starttime)(speed t)))
     (+ (speed t) 900)))
  :condition(and
    (at start (not (killed t))))
  :effect(and
    (at starttime (consume (AvailableESSM) 1))
    (during[starttime,endtime] (reserve (STIR) 1)))
  :probabilistic-effect(
    (:probability (call ESSM-Killprobability (t
      (-(detectrange t)(*(endtime)(speed t))))))
    (:effect (killed t)))
...))
(:problem
  (:objects  AGM84 C701 AS17 - target)
  (:init
    (= (detectrange AGM84) 52000)
    (= (speed AGM84) 300)
    ...)
  (:goal (and
     (killed AGM84) (killed C701) (killed AS17))))
```

In this case, the domain expert specifies the preconditions, resources utilization and probabilistic effects. For the Fire-ESSM action, the duration is computed by estimating the interception time that depends on the target detection range, the action starting time, the target speed and the ESSM speed (assumed 900 m/s). Since the P_{se} depends on the interception range, a table that summarizes the ESSM P_{se} is provided.

Goals

The goal is given by specifying an objective function to optimize. For instance, in the CPM domain, a goal may be to maximize the probability of destroying all the threats. Alternatively, given the threat level for each target, the goal may be to minimize the expected damage. Furthermore, by defining values for different regions of potential impact, the goal may be to maximize the expected survival value. Additionally, one may want to optimize the use of available resources (e.g., minimization of the monetary cost of the battle). This could be done by optimizing a score which is a linear combination of the ship survival probability and resource costs.

Local and Global Plans

A plan is a list of timed actions, each associated with corresponding resources. Figure 3 illustrates a plan that maximizes the defending ship survival probability. The plan corresponds to the initial state depicted in Figure 1. Actions are grouped by targets and represented by solid bars on the time line. An plan against a single target is also called a local plan. A global plan is defined as a combination of several local plans and allows countering multiple simultaneous targets.

Figure 3 shows three local plans, respectively for AGM84, AS17, and C701 type targets. The interpretation of the first plan is as follows: 10 seconds after the engagement begins, Chaff is launched to distract the target (avoid detection of ownship) and will remain effective until time 58. At 116 seconds, an ESSM is launched and is expected to intercept the AGM84 ASM target at time 129. The two other local plans can be interpreted in a similar way.

Note that contingencies are implicitly represented in the list of actions. When the current action in a local plan fails to neutralize the corresponding target, the next action is executed. Once the target is neutralized, subsequent actions are not executed, and the associated resources become available so the planner can consider using them against other targets during the replanning phase. Each action increases the cumulative success probability of the plan. Actions in Figure 3 are displayed with the corresponding P_{se}. The cumulative P_{se} of the global plan is displayed on the first line. This probability represents also the ship's final survival probability to all the threats. This probability is given under the assumption that a non-destroyed threat is sufficient to sink the ship. As mentioned before, the planner allows for the use alternative objective functions, including those taking into account partial damage to the ship by optimizing a ship survival value.

This plan representation is specific to the CPM domain and follows requirements given by human-machine interaction experts. The experts ruled out explicit branching contingencies as they found them more difficult to digest in situations requiring short decision times. Otherwise, CORALS could be quite easily extended to add branching contingencies. This way, new threats would be handled via conditional branches, rather than replanning as it is currently the case.

Planning Method

Overview

CORALS algorithm performs a search in the space of plans. Traditional plan-space approaches (such as UCPOP (Penberthy and Weld 1994) or SNLP (McAllester and Rosenblitt 1991)) begin with an empty plan and then iteratively add new actions. As the actions are added to the plan, conflicts introduced by these actions are resolved. CORALS planner does the opposite. It starts by generating local plans, one for each target, and then merges them into one global plan. Since the local plans are generated independently, their union may create conflicts. Conflicts resolution is performed in a second step, with the three following possible modifications applied iteratively: 1) shift actions along the time axis; 2) remove actions; and/or 3) add new actions. This plan-merging approach can also be understood as a plan-repair approach.

Algorithm 1 shows the entry point of the planner, which takes as input a list of activities and produces a conflict-free plan. The 'OK' function returns $true$ if a stop condition is met. This can be based on a quality (P_{se}) threshold for the global plan and/or a timeout for the planning process.

Algorithm 1 Plan Merging Algorithm

```
1.  Algorithm GENERATEPLAN(Activity activities[1...n])
2.    Plan globalplan = ∅, bestglobalplan = ∅
3.    while (not OK(globalplan) )
4.      Plan localplans = GenerateNextLocalPlans(activities)
5.      if (localplans = ∅)
6.            return bestglobalplan
7.      globalplan= ResolveConflicts(localplans)
8.      if Better(globalplan, bestglobalplan)
9.         bestglobalplan=globalplan
10.   return bestglobalplan
```

The function 'GenerateNextLocalPlan(activities)' generates a set of new local plans, which are different from the previous calls. Iteratively, local plans are generated and merged until a stop condition is met or there are no more new local plans to merge.

The function 'GenerateNextLocalPlan' is defined depending on the domain. In the CPM domain, activities correspond to targets (*i.e.*, the activity is to neutralize a target), and we have a local plan per target. The local plan for a given target is given simply by assigning to each target all available resources. No search is involved in this case. This way, 'GenerateNextLocalPlan' generates only one plan (i.e., one set of potentially conflicting local plans)[2]; the next time it is called, it generates an empty plan since there is only one possible set of local plans obtained by assigning all resources to each target. Hence, in the CPM domain the 'while' loop at line 3 requires only one iteration. In other domains, 'GenerateNextLocalPlan' may be more elaborate and involve search. Each time it is called it must generate

[2]The actual implementations involves some trivial removal of resources that are obviously not applicable, such as removing a missile launch with a blind zone containing the target position

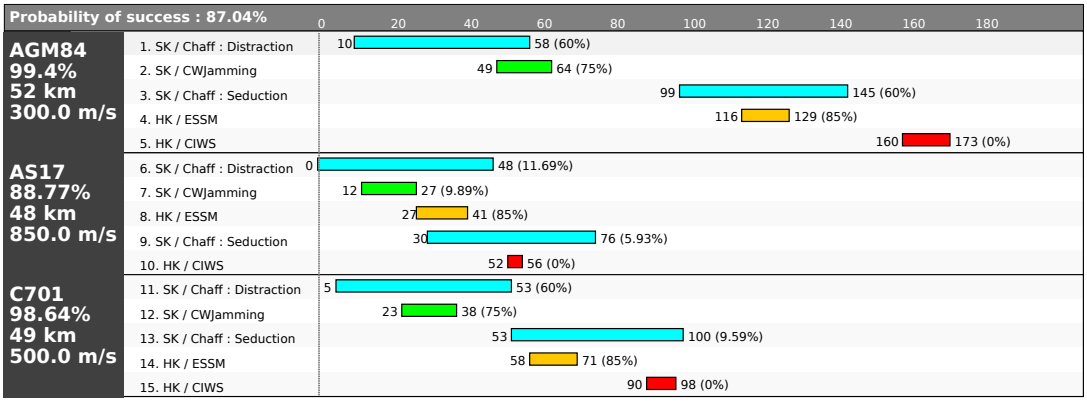

Figure 3: Example of engagement plan

a plan (i.e., again, a set of local plans, potentially with conflicts), and so on, until the plans are exhausted. In this case, 'while' loop at line 3 will iterate over the plans.

Detailed Algorithms

Algorithm 2 finds a conflict-free plan by performing a heuristic search in the space of all possible plans. The algorithm manages two lists. The open list contains plans to be processed and the close list tracks the plans that have already been visited in order to avoid infinite cycling. At beginning, $inputglobalplan$ is added to the 'open list' (Line 4).

Algorithm 2 conflict resolver Algorithm

```
1.  Algorithm RESOLVECONFLICTS(Plan inputglobalplan)
2.    Variable open, close, plan, bestplan
3.    inputglobalplan.hvalue = EvaluateHeuristic(inputglobalplan)
4.    open.add(node)
5.    while (! open.isEmpty)
6.      plan = open.removeFirst() // returns node highest hvalue
7.        close.add(plan)
8.      conflicts = DetectConflicts(plan)
9.      if (conflicts.isEmpty())
10.       if EvalutateMetric(plan) > EvaluateMetric(bestplan)
11.         bestplan = plan
12.     else
13.       sucessors = GenerateSuccessors(plan, conflicts)
14.       for each s in sucessors
15.         if newplan not in close
16.           newplan.hvalue = EvaluateHeuristic(newplan)
17.           open.add(newplan)
18.     return bestplan
```

When processing a new plan, it is first added to the 'close list' to visit it only once. Conflicts are then detected ('DetectConflicts') and the list of conflicting actions is returned. If the newly generated plan does not contain any conflicts, it becomes a candidate for execution. It is then compared (using 'EvaluateMetric') to the best conflict-free plan found so far. If the quality of the new plan is higher, it will be kept as the new best plan. This anytime feature of the planner guarantees that a conflict-free plan is always available for execution and that the quality of the current best plan improves over time.

If the newly generated plan contains conflicts, these will have to be resolved. This resolution is performed incrementally, handling one conflict at a time, by invoking The 'GenerateSuccessor' procedure. The latter takes the plan and the set of conflicting actions as parameters and returns a list of plans with exactly one modification of a conflicting action. A modification may consist in either rescheduling the action at different time or completely removing it.

Plans in the open list are sorted and explored based on their respective value ($hvalue$). This value, which is determined by a heuristic, represents an estimation of how promising the node is on a path to the optimal solution.

Conflict Detector. Prior to their resolution, conflicts must be identified. To find conflicting actions, we have in each visited node (plan) of the search space a resource agenda keeping track of the use of resources (see Algorithm 3). This agenda maintains a list of events keeping track of the reservation and release of resources by actions and these events are sorted chronologically. A conflict is declared if the number of reservations for a resource exceeds its capacity for a specific time interval.

As mentioned before, we distinguish between renewable and consumable resources. Renewable resources have a fixed capacity and can be reserved by an action for a period of time. After its utilization, such resource is released and becomes available again. In contrast, consumable resources are not automatically renewed; each such a resource has a maximum capacity that cannot be exceeded. Addotionaly, we have limit that cannot be exceeded at a specific time. For the sake of conciseness, the management of consumable resources is not reflected in Algorithm 3.

Figure 4 shows an agenda for a simple plan. Each chaff action reserves the chaff launcher for a fixed duration. If there is only one chaff launcher and two chaff actions are scheduled at the same time, the plan will not be feasible. In such a case, actions #1 and #2 are returned as conflicting actions.

Algorithm 3 Conflicts Detector Algorithm

```
1.  Structure Reservation:
2.      Integer quantity
3.  Structure RenewableResource:
4.      Integer capacity, quantity
5.      Reservation reservations[Action]
6.  Structure Event:
7.      Action action
8.      Resource resource
9.      Integer time, quantity
10.
13. Algorithm DetectConflict(Plan plan, Resource[] ress)
14.     Variable: events[time index], conflicts=∅
15.     for each Action a (in) plan :
16.         actionevents = a.getResourceEvents()
17.         for each Event e (in) actionevents :
18.             events[e.time].add(e)
19.     for t = 0 to maxtime :
20.         for each e in events[t] :
21.             e.resource.quantity += e.quantity
21.             e.resource[e.action].quatity += e.quantity
22.         for each r in ress :
23.             if(r.quantity > r.capacity)
24.                 for each a in plan:
25.                     if r.reservations[a].quantity > 0
26.                         conflicts.add(a)
27.                 return conflicts
28.     return ∅
```

Heuristics. We have seen that most promising nodes (plans) are visited first, according to the heuristically sorted open list in Algorithm 2. The next subsections discuss the heuristics we use.

Number of Conflicts Heuristic. A *hill-climbing* approach offers a good strategy for efficiently computing a solution. Using this approach and a heuristic based on the number of conflicts, the nodes in the open list are sorted by their number of conflicting actions. The search algorithm always selects the node closest to a conflict-free solution, *i.e.*, one with the smallest number of conflicts.

Lower Bound Heuristic. The previous heuristic has the advantage of producing conflict-free plans rapidly, but it does not generate high quality plans. A better heuristic should include the plan quality (based on the given objective function) in its estimate.

To do that, each node in the open list is associated with a potential branch in the search space; that is, its successors. The idea consists in selecting the nodes that will lead to branches having the highest expected quality. This is achieved by using the lower bound estimate of the quality of a branch. In other words, the nodes in the open list are sorted using a lower bound estimate of the quality that can be obtained from a sub-search in that branch. The next node to be selected and explored will be the one with the highest lower bound quality.

There are many ways to compute the lower bound of a branching. The higher the lower bound is and the closer it is to the real maximum value, the better it will be. But the lower bound of a given branch should never overestimate the real minimum expected value of the branch. For a given node n, a lower bound of value q always guarantees to reach a conflict-free node n' with at least a quality of q. Consequently, if S_n is the set of successors of n, this means that there exists at least one node $s \in S_n$ where its lower bound is q', with $q' \geq q$.

The method used to compute the lower bound is simple and fast. It consists in evaluating the quality (objective function) of the plan on its sub-plan obtained by removing all conflicting actions.

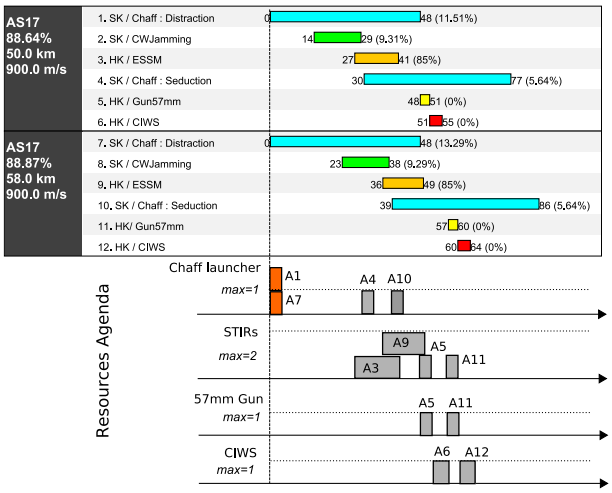

Figure 4: Example of a resources agenda built from a plan

Mixed Heuristic The two previous heuristics are efficient at finding either a plan rapidly or a good plan. But none of them is effective in finding a good plan rapidly. By computing a linear combination of these two heuristics, we obtain a tradefoff between higher performance and higher plan quality. This mixed heuristic is computed by the following simple equation, where c is an empirical parameter, with $0 \leq 1$.

$$H_{mixed}(p) = H_{LB}(p) - c \times H_{NbConflicts}(p) \quad (1)$$

Completeness and Optimality

In the worst case, CORALS will explore the space of possible plans exhaustively. Therefore, under the assumption that the function 'GenerateNextLocalPlans' is able to generate exhaustively all local plans, if an optimal plan exists, CORALS will find it. However, the search space is potentially huge even for a small number of targets. The heuristics ensure that the exploration begins with the most promising plans first. In practice, the plan quality increases with the time allocated to planning and optimality can only be guaranteed after the search space is exhausted.

Execution Monitoring and Replanning

The CPM domain is very dynamic. The tactical situation can change drastically during the plan elaboration and/or execu-

tion. New threats can appear, while others may disappear. Combat systems and the underlying resources can have failures too. Therefore the execution of a plan is monitored to detect changes that are not accounted for in the planned actions (*i.e.*, contingencies). For such changes, the current plan must be revised. CORALS planner is good not just for generating plans from scratch. It can also be used to revise existing plans when new information is collected from the environment. For instance, as a new target is detected, the planner could be invoked to create a new local plan for this target, to merge it into the current global plan, and resolve any resulting conflicts. This technique has two advantages: 1) it speeds up the planning process; and 2) it reduces the number of modifications required compared to replanning. The last feature means that CORALS to some extent avoids unnecessary changes to already committed or soon to be committed actions.

Experiments

We tested CORALS planner on the CPM domain. Random scenarios with targets of different types and speeds, and at ranges and bearings, were generated. The plan quality was measured by the probability of a successful engagement of all the targets. The experiments were conducted on a Pentium 4 Compouter with 1.8 GHZ CPU and 1GB RAM.

For scenarios involving less than five targets, CORALS produced high quality plans in less than one second. Figure 5 shows the performance curves of CORALS with the three above-described heuristics. The x-axis represents the planning time, and the y-axis is the quality of the best plan found up to this time. A longer search allows visiting more nodes, and therefore is generally associated with a higher final plan quality. The quality measures displayed on this chart are averages computed from the plan qualities generated from 20 random scenarios of 5 different targets.

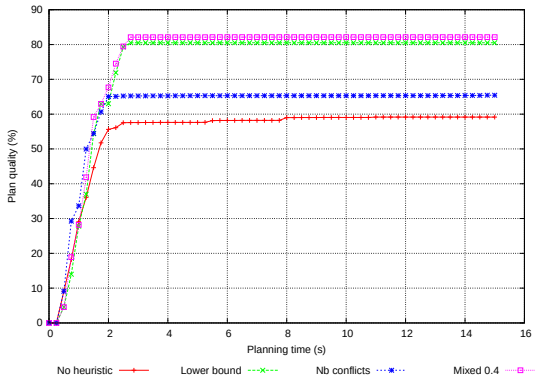

Figure 5: Comparison of different heuristics

The current doctrine in ASMD consists in applying predefined defence plans based on the characteristics of the target. These plans, called ZIPPO tables, are generated off-line based on the analysis of data from previous engagements and using war-gaming tools. ZIPPO tables handle only one target at a time and are used as support tools by human operators during operations. In situations involving multiple targets, the ZIPPO tables are considered as local plans, and the operators rely on their own experience and know-how to merge them, with all associated cognitive overhead and risk of errors. To estimate the benefits of using CORALS compared to the currently used ZIPPO-based scheme, we compared CORALS against a planner that simulates the ZIPPO tables. Figure 6 shows the overall improvement brought by CORALS.

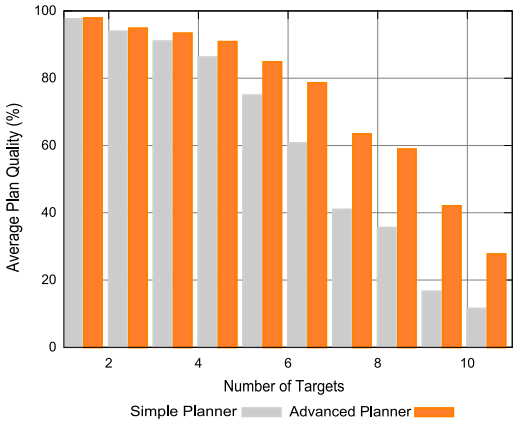

Figure 6: Improvement test

In the introduction of this paper, we mentioned FPG (Buffet and Aberdeen 2006) as the only planner publicly available for comparison. As a matter of fact, since FPG does not allow actions with probabilistic effects that depend on their execution times (probabilities are specified by numerical constants), we made comparisons on a simpler version of the CPM domain in which probabilities are set to be constant. Note that without time-dependent success probabilities, the problem becomes much easier since the planner does not have to optimize the actions along the timeline. On several such problems, the CORALS planner returns the final solution practically instantaneously, because no rescheduling of action was necessary to resolve conflicts. In contrast, Figure 7 shows the progression of FPG planner on a scenario involving three targets. FPG takes about 20 seconds to reach a plan quality of 80 %, while CORALS reaches it within the first second.

This result shows the superiority of CORALS in a simplified version of the CPM domain. However, we must keep in mind that although both planners are generic, the CPM domain is particularly well suited to CORALS in that local plans can be generated quickly and conflicts checked efficiently. When it comes to applications of planning, there is a large spectrum of possibilities between domain-specific planners and general-purpose planners. CORALS planner lies somewhere in-between, as it is relies on domain-independent heuristics and does not make use of any domain knowledge (other than action specifications), but has an algorithm that assumes that local plans can be generated quickly and they involve conflicts that can be checked and resolved efficiently. This line of inquiry is complementary to approaches that use search control knowledge in planning to

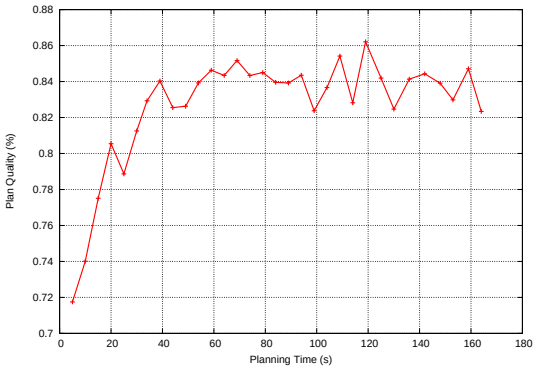

Figure 7: FPG performance

constrain more general-purpose planning engines to a more specific planning domain, such as TLPLAN (Bacchus and Kabanza 2000).

Conclusion

CORALS, a planner for CPM applications in naval warfare operations was presented. This planner is well suited to a particular class of probabilistic domains with concurrent durative actions, where concurrency is observed between localized processes requiring some coordination in the use of shared resources. CORALS planner exploits a modeling of local plan for each activity and searches in a space of global plans to merge these local plans efficiently. The planner was evaluated on realistic CPM scenarios and its performance was very conclusive.

The planner development was motivated by the CPM domain; hence its design is to a large extent biased towards similar domains. However, the planner is generic and could be used, for example, to generate plans in war games, or other similar applications. It may also be used for problems that do not share all the characteristics of the CPM domain, but its performance might be affected.

Work is in progress to integrate CORALS into a prototype of a naval command and control system and test it at sea in a real life setting. Work has already been done on the human factors to determine the actual display of plans to ship operators to minimize cognitive information overload. Future work will include search into a belief space, search control knowledge, and applications to other domains.

Acknowledgements

This work was funded by the Canada Department of National Defense. We thank Taek-Sueng Jang for his contribution to the implementation. We are also grateful to the anonyomous reviewers for comments that helped improving the final version of the paper.

References

Ahuja, K.; Kumar, A.; Jha, K.; and Orlin, J. 2003. Exact and heuristic algorithms for the weapon target assignment problem. *MIT Sloan Working Paper No. 4464-03*.

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Bertsekas, D.; Homer, M.; Logan, D.; Patek, S.; and Sandell, N. 1999. Missile defence and interceptor allocation by neuro-dynamic programming. *IEEE Transactions on Systems, Man and Cybernetics, Part A* 30(1):42–51.

Bohachevsky, I., and Johnson, M. 1993. Optimal deployment of missile interceptors. *American Journal of Mathematical and Management Sciences* 13(1-2):53–82.

Bos, J.; Rehak, L.; Keeble, A.; and Lamoureaux, T. 2005. Tactical planning and response management: Investigating a cognitive work analysis approach to the development of support concepts. Technical report, DRDC Atlantic.

Brucker, P., and Knust, S. 2006. *Complex Scheduling*. Springer.

Buffet, O., and Aberdeen, D. 2006. The factored policy gradient planner. In *Fifth International Planning Competition*.

Hadj-Alouane, A.; Bean, J.; and Murty, K. 1999. A hybrid genetic/optimization algorithm for a task allocation problem. *Journal of Scheduling* 2:189–201.

Hosein, P., and Athans, M. 1990. Some analytical results for the dynamic weapon-target allocation problem. *Naval Research Logistics Journal*.

Lloyd, S., and Witsenhausen, H. 1986. Weapons allocation is np-complete. In *IEEE Summer Conference on Simulation*.

Matlin, S. 1970. A review of the literature of the missile-allocation problem. In *Operations Research*, 334–373.

Mausam, and Weld, D. S. 2006. Probabilistic temporal planning with uncertain durations. In *National Conference on Artificial Intelligence (AAAI)*.

McAllester, D., and Rosenblitt, D. 1991. D. mcallester and d. rosenblitt. In *National Conference on Artificial Intelligence (AAAI)*, 634–639.

Meuleau, N.; Hauskrecht, M.; Kim, K.-E.; Peshkin, L.; Kaelbling, L.; Dean, T.; and Boutilier, C. 1998. 15^{th} national conference on artificial intelligence (AAAI). In *Operations Research*, 165–172.

Ousborne, D. 1993. Ship self-defence against air threats. *Johns Hopkins APL Technical Digest. 12(2):125-140.*

Penberthy, J., and Weld, D. 1994. Ucpop: A sound, complete, partial order planner for adl. *AI Magazine* 15(4):27–61.

Younes, H., and Simmons, R. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 325–334.