Automated Adaptation of Strategic Guidance in Multiagent Coordination*

Rajiv T. Maheswaran[1], Pedro Szekely[1], and Romeo Sanchez[2]

[1] University of Southern California
Information Sciences Institute and Department of Computer Science
{maheswar, pszekely}@isi.edu,

[2] Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica
nigenda.romeosn@uanl.edu.mx

Abstract. We address multi-agent planning problems in dynamic environments motivated by assisting human teams in disaster emergency response. It is challenging because most goals are revealed during execution, where uncertainty in the duration and outcome of actions plays a significant role, and where unexpected events can cause large disruptions to existing plans. The key to our approach is giving human planners a rich strategy language to constrain the assignment of agents to goals and allow the system to instantiate the strategy during execution, tuning the assignment to the evolving execution state. Our approach outperformed an extensively-trained team coordinating with radios and a traditional command-center organization, and an agent-assisted team using a different approach.

Keywords: Multi-Agent Systems, Real-Time Coordination, Human-Agent Collaboration, Mixed-Initiative Approaches, Disaster / Emergency Response

1 Introduction

Recent disasters in Haiti, Chile, Christchurch and Japan caution us that emergency response is a significant practical global issue. They draw attention to the need for technologies that assist human responders to perform more effectively. We address a fundamental challenge in these domains: most important tasks (e.g., rescuing injured, restoring services) are only revealed during execution, i.e., the severity, type and locations of injured people or the damage done are revealed *while* the rescue operation is being performed. Humans form high-level strategies that do not fully specify actions

* The work presented here is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S. Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them. Approved for Public Release, Distribution Unlimited.

and dynamically implement them with information revealed during execution. Additional complexities include managing teams with heterogeneous capabilities, tasks that require coordination of multiple skills and resources, task failure, loss of agent capabilities and uncertain durations.

There are several relevant multi-agent system approaches. However, most require a specification of the problem that is infeasible under conditions where most tasks are revealed during execution. Considering all evolutions (e.g., all possibilities of injured and damage) scales the problem beyond the capabilities of current technologies. It is prudent and often necessary to allow humans to outline a high-level strategy. This necessitates having a formalization that captures how humans describe strategies, while maintaining sufficient richness to enable computational assistance for execution. The challenge is devising a strategy specification language amenable to human strategic thinking and algorithms to execute such strategies flexibly and efficiently.

Our contribution is an approach, STaC, composed of a strategy specification language that captures human-generated high-level strategies and corresponding algorithms that execute them in dynamic and uncertain settings. This partitions the problem into *strategy generation*, designed by humans and understood by the system, and *tactics*, orchestrated by the system with information to and from responders on the ground. STaC gives the ability to create changing *subteams* with *task* threads under *constraints* (e.g., focus on injured). The connection between a STaC strategy and the STaC execution algorithm is the notion of *capabilities*: agents have capabilities; tasks require capabilities. STaC dynamically updates the *total capability requirements* (TCRs) for the tasks in the strategy and assigns agents to tasks during execution following the human guidance.

STaC was evaluated in three high-fidelity independently-conducted real-world field exercises. We outperformed a traditional human approach by a significant margin in all three exercises, where a third team using software-assisted agents did not succeed. Additional analyses show STaC robustness with respect to strategy and automated planners that manage goals.

2 Motivation

Fig. 1. Agent activities in the field exercises (left) and map of park and pathways (right).

We were challenged to create a multi-agent system that improved performance in simulated disaster-rescue field exercises. Each exercise was conducted in a park. Different parts were designated as sites with an unknown number of injured in *serious* or

critical condition as well as *gas*, *power* and *water* substations which may have been damaged. Park pathways were roads that agents walked to travel between sites (see Fig. 1). Teams earned points by rescuing injured to hospitals or operational clinics (before a deadline associated with each injured person) and restoring damaged substations. The goal was to maximize points in 90 minutes.

The points per rescue and repair differed by site, injury type and service type. The number of injured or type of damage at each site was unknown ahead of time. To discover this information, agents with capabilities to *survey for injured* or *survey for damage* needed to visit the sites. Before performing any survey, the site first had to be *isolated* to ensure safety for those conducting the surveys. Isolation, survey, repair and rescue tasks all required different combinations of capabilities.

Each team had 8 field agents and 2 commander agents. Each field agent had different capabilities, which included performing major or minor repairs for gas, power or water and saving certain types of injured. Commander agents stayed at a base and helped to coordinate activities. The baseline *Radio Team* operated under a traditional command structure and communicated only with radios. Our *STaC Team* and a third team with a different approach had radios and ruggedized tablet computers with cell modem and GPS capabilities running agents.

Consider an agent who completed surveys for damage and injured at a site. They can (1) start a low-probability repair alone, (2) wait for another agent to start a medium-probability repair, (3) get a repair kit from the warehouse for a high-probability repair, (4) rescue seriously injured, (5) wait for another agent to rescue critically injured, or (6) go to another isolated site and perform surveys. The right choice depends on the availability of other agents. Making good decisions is difficult without situational awareness.

The Radio Team realized that commanders could not make detailed decisions for agents in the field. Instead, they employed a strategy that formed subteams, defined itineraries for subteams and restricted actions to specific tasks (e.g., isolation, power repairs, critical injured rescue). These structural restrictions enabled commanders to delegate detailed decision-making to agents in the field. One commander assembled up-to-date situational awareness from multiple simultaneous field reports. The other focused on problem solving to execute the strategy by directing agents to locations, where agents would decide how to accomplish tasks. Consider the following exchange:

Agent 1:	Found power problem at Site 2, need power specialist.
Agent 1:	Found 3 seriously injured at Site 2.
Commander:	Power Specialist, go to Site 2.

In the meantime, Agent 1 succeeds with a low probability power repair.

Agent 1:	Power repaired at Site 2.
Commander:	Agent 1 go to Site 3 and do surveys.
Commander:	Power Specialist, skip Site 2 and go to Site 4.

Agent 1 and the Power Specialist are a subteam responsible only for power repairs and surveys for injured. Agent 1 arrives at Site 2 first, surveys for power damage, reports a problem, and, after looking at repair tasks, requests the Power Specialist. Agent 1 then surveys for injured and because the Power Specialist has not arrived, attempts and completes a low-probability repair. Consequently, the Commander redirects the Power

Specialist to Site 4. Agent 1 makes detailed field decisions such as doing the power survey first to discover the needed capabilities and attempting the low-probability repair. The Commander, who has better situational awareness, executes the high-level strategy, i.e., assigning agents to goals. This simple scenario illustrates the management commanders perform. The complexity increases as all 8 agents frequently and often simultaneously report status, and task failures, delays, agent injuries, vehicle breakdowns and road blocks occur. The structural restrictions of the strategy let the Radio Team partition responsibility for decisions in a manner that allowed them to perform well.

Two insights yielded avenues for improvement: (1) humans do not execute their strategies efficiently (e.g., it may take a human commander a few minutes to gather situational awareness, calculate the desired actions and communicate them to the team), and (2) humans sometimes forget to task agents or communicate actions (e.g., forgetting to tell the Power Specialist to skip Site 2) causing waste. If a system could generate the appropriate assignments, it could calculate and communicate them in seconds and would not forget to enact them. The cost is that the formalization that systems require do not have the expressivity to capture the full space of strategies and adaptations that humans can generate. Thus, we must find a scheme and associated algorithms where loss in expressivity and flexibility is overcome by gain in efficiency of execution.

3 Approach

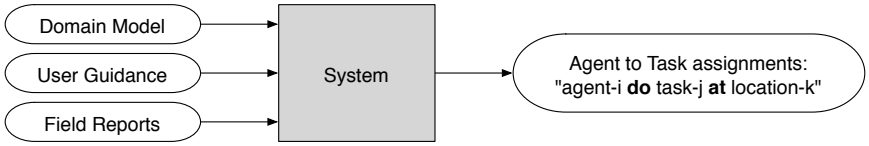

Fig. 2. The problem to be solved.

Figure 2 shows inputs and outputs of a system that addresses our problem. The domain model gives utilities for potential goals (e.g., restore power at Site 2) and tasks that achieve them, agents and capabilities, geography of sites and roads, etc., before execution. Field reports, received during execution, include discovered goals (e.g., 5 critically injured at Site 3), task success/failure, agent locations and availability, etc. User guidance is information provided by a human planner to influence system behavior. The objective is generating actions for human agents, i.e., where to go, what to do.

An "ideal" system could select assignments to maximize utility without user guidance. Current planning technology does not support this. We encoded simplified versions of the field exercise scenarios in PDDL [7]. We made all goals known *a priori* (i.e., gave prescience on what would be discovered), fixed travel durations, and removed failure, deadlines on injured and all dynamic events (e.g., road blocks, agent injuries, vehicle breakdowns). From the deterministic planners we tried, only LPG-TD [8], and SGPLAN [6] could scale up to 5 sites under such restrictions[3]. This verified the need for significant user guidance in this domain.

[3] SGPLAN was the winner of suboptimal tracks in the 4th and 5th ICAPS International Planning Competition, while LPG-TD got a second prize in the 4th competition.

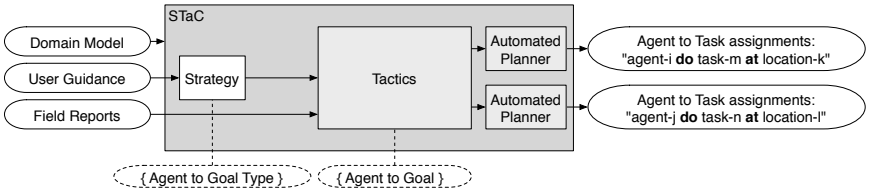

Fig. 3. STaC problem reformulation.

We reformulated the problem as shown in Figure 3. The human planner gives a *strategy* that specifies how agents are assigned to goal types (e.g., rescue, restore, isolate, survey) under various constraints (location, timing, ordering). STaC provides the *tactics*, i.e., assigning agents to discovered goals by fusing the strategy with field reports (e.g., discovered goals, agent locations, availability) during execution. Once agents are assigned to goals, automated planners (one for each goal) handle the agent-to-task assignment. The decomposition allows the automated planners to independently solve significantly smaller problems, i.e., single goals instead of the entire mission.

For this approach to work, we must solve two related problems: (1) We need a strategy specification language that enables human planners to express agent-to-goal-type assignments, and relevant constraints. (2) We need algorithms to efficiently execute the strategy, i.e., make appropriate assignments of agents to discovered goals.

3.1 Problem Formulation

We do not present our full formalization of the general problem (described in Section 2) due to space. However, we discuss a few key ideas that will aid in understanding our approach. Before the exercise begins, there are certain goals (e.g., isolation, surveys) that we know we can achieve if we so choose, because the associated tasks to complete them exist (e.g., turning off power and gas for isolation). But, we cannot know whether there are people to rescue at a site or whether a particular substation needs restoration until we survey that site during execution. Thus, we introduce a concept, *goal type*, to discuss goals we might discover during execution. Utilities are associated with achieving particular goal types (rescues and restorations) at particular sites, if they are discovered during execution. Rescuing critically injured as Site 1 may be twice as valuable as restoring power at Site 7 which in turn may be twice as valuable as restoring gas at Site 7.

If we discover these goals in the exercise, task hierarchies capture the set and sequence of actions needed to achieve them. The leaves represent actions that agents with appropriate capabilities must perform. Each goal may have multiple task hierarchies that could achieve the goal but failure of any task within a hierarchy means that it no longer can achieve the goal. For example, a broken transformer at a power substation has multiple repair options. Each repair option is a task hierarchy that may have enablement or synchronization constraints. Similarly, there are several ways to rescue an injured person. Our task representation is a variant of CTAEMS [3], extended so that the capabilities needed to perform tasks are explicitly represented.

3.2 Strategies

STaC strategies enable human planners to (1) flexibly assign goals or goal types to appropriate subteams, (2) re-form subteams as needed, (3) order goals, while allowing parallelism, and (4) address special situations that require clever use of valuable resources and remaining time. We define a STaC strategy using a grammar:

1. <strategy> := <thread>+
2. <thread> := <agent>+ <goal-constraint>+
3. <goal-constraint> := <goal-spec> <constraints>
4. <goal-spec> := <goal> | <goal-type> <location>
5. <constraints> := [<timing-constraints>] (<capability> <usage-limit>)*

Strategies are composed of *threads* that can operate in parallel during execution based on agent availability. Each thread is composed of a subteam (<agent>+) and a sequence of goal or goal types they should pursue under some constraints (<goal-constraint>+). Threads and goal-constraint tuples are ordered and agents move sequentially through those where they are part of the subteam. Timing constraints prevent agents from attempting goals when there is insufficient time, redirecting them to quicker goals when time is running out. Capability constraints limit how often a capability can be exercised while pursuing a goal. Using these constraints, a human planner can control the amount of work done (e.g., rescue only 4 injured). In a later thread, a subteam can revisit the same location and achieve more goals of the same type. Goals and goal types can appear in multiple threads and multiple times within a single thread. Thus, human planners can create strategies that specify multiple attempts to meet goals at different times, with different agents and different constraints.

The table below shows an example strategy. For compactness, we show it in tabular format rather than the grammar, and encode goal types and locations as $g@l$ where g is a goal or goal type and l is a location:

Thread	**Subteam**	**Goals & Constraints**
1	A_1	*survey@(a, b, c, d)*
2	A_2, A_3	*gas@(a, b)*
3	A_4, A_5	*power@(c, d)*
4	A_1, A_4	*water@(a, b)*
5	A_2, A_3, A_5	*injured@(a, b, c)[drop-off 6]*
6	$A_1, \ldots, A_5$	*critically-injured@(d, e)*

In thread 1, A_1 is to survey locations a, b, c and d. The others are split into subteams optimized for gas (thread 2) and power repairs (thread 3). Subteams are reshuffled after these threads complete. The strategy limits each subteam to specific goals with the expectation that threads 1,2 and 3 complete at roughly the same time.

While the strategy sends both A_2 and A_3 to locations a and b, STaC algorithms will optimize the execution by determining when agents can skip goals, locations and threads. For example, the surveys A_1 performs may reveal no damage in b or reveal goals with tasks requiring a single agent. In the former case, both agents can advance to their next threads without visiting location b. In the latter case, one agent will be sent to location b while the other proceeds to its next thread.

In threads 4 and 5, subteams are reshuffled to optimize for different goal types. A_1 and A_4 focus on water repairs at locations a and b, which by this time should have been surveyed. The other agents focus on rescuing injured at locations a, b and c. Thread 5 limits the agents to rescuing 6 injured people by limiting the *drop-off* capability. This forces agents to move to other locations where they are also needed. In thread 6, all agents come together to rescue critically injured at two locations until time runs out. The sequence in which agents arrive at thread 6 is determined by events and outcomes during execution. The example illustrates how a strategy can exert control while also providing flexibility to accommodate variance during execution. A strategy constrains the space of execution paths for a team. The instantiated path is determined at run-time.

An important issue is to help human planners develop good strategies. We built simulation and visualization tools to help human planners understand the evolution of strategies [10]. Using these visualizations, planners can observe the behavior of subteams and refine the strategies as appropriate.

3.3 Executing Strategies

A strategy puts structure on who can be assigned to what, when and where. Once goals are discovered during execution, the exact who, what, when and where must be decided based on agent availability and location along with the strategy. We must continuously assign and re-assign agents to goals. The choice of which agents remain and which are passed to the next goal depends on the needs of the particular goal being considered.

Our strategy execution algorithms calculate the capability requirements needed to achieve each goal and ensure that the collective capabilities of the agents assigned to the goal are sufficient to meet the capability requirements of the goals. The capability requirements are updated dynamically when agents complete tasks. As these requirements change, unneeded agents are re-assigned to the next goals in the strategy.

We assign agents to goals based on *total capability requirements* (TCR). The TCR of a task characterizes the set of capabilities required to complete it. TCR calculations aggregate capabilities required by atomic actions up the task hierarchy, across enablements and synchronizations that may need to be performed at different locations. TCRs are defined as follows:

1. <tcr> := <req>+
2. <req> := <req-num> <req-type>
3. <req-type> := <req-elem>+
4. <req-elem> := <capability> <location> <amount>

A TCR structure is composed of *requirement elements* (line 4) which denotes that a certain *amount* of a particular *capability* is required at a particular *location*, e.g., two instances of minor power at Site 3. A *requirement type* (line 3) is a collection of these elements. Restoring a power at Site 1 could require simultaneously turning on power at the substation, and a power main located in a different city, Site 2.. The requirement type for this task would be composed of two requirement elements: one instance of minor power at Site 1 and one instance of major power at Site 2. A *requirement* (line 2) states how many instances of a particular requirement type are needed. If there are 5

critically injured at a site, the requirement for rescuing all of them would be 5 instances of the requirement type to rescue one critically injured. Finally, the *TCR* for a task is a collection of requirements. This could represent the requirements for all tasks at a site.

The TCR for a task is computed bottom-up, starting with the actions at the leaves of task structures. The *local* TCR for leaf nodes is a single requirement type, composed of one requirement element for one instance of the capability associated with the action at the location to be performed. The TCRs for all nodes incorporate TCRs from all enabling nodes and children nodes, if they exist. We first discuss how TCRs from children are aggregated and then present the general algorithm for updating TCRs (Fig. 4).

AsyncMerge (Alg. 1) takes in a set of TCRs and outputs a single TCR. This is applied only to tasks that do not require any synchronization of their children. We decompose and organize all the TCRs by requirement types, and then aggregate the associated requirement numbers based on how the planner views execution of that requirement type. A key definition is the function Operator[reqType] which determines how the aggregation occurs. For a requirement type where the planner desires serial execution, the operator is max. This may be used for repairs where a single skill can be used to satisfy multiple requirements. Alternatively, if the planner desires parallel execution, the operator is sum. This may be used to increase resources allocated to rescuing injured.

SyncMerge (Alg. 2) also takes in a set of TCRs and outputs a single TCR. This aggregates the TCRs of children of synchronization tasks which require capabilities to be devoted in a manner such that each child node executes simultaneously. A subtlety of synchronization tasks is that their children may be achieved in multiple ways. The TCR for a synchronization task must consider all ways that the children can be executed simultaneously. Each requirement in the TCR of a child node captures the capabilities that may be needed for it to succeed. A combination of requirements is a set that takes one requirement from the TCR of each child. We gather all such combinations to characterize what may be needed to successfully execute the synchronized task.

Consider a synchronized task with two children having $\text{TCR}_1 = \{1 \cdot (gas, a, 1), 1 \cdot (water, a, 1)\}$, $\text{TCR}_2 = \{1 \cdot (power, b, 1)\}$. Here, the first child has two requirements at location a, the second has one requirement at location b. All requirement types have a single element, representative of primitive actions. The TCR of the synchronized task will have $\text{TCR}_0 = \{1 \cdot [(gas, a, 1)(power, b, 1)], 1 \cdot [(water, a, 2)(power, b, 1)]\}$. This states that the task could require a *gas* capability at a with a *power* capability at b or a *water* capability at a with a *power* capability at b. At least two agents are required.

The *UpdateTCR* procedure (Alg. 3) dynamically updates the TCR of any task, based on the TCRs of both subtasks and enabling tasks. If the task is a leaf node (i.e., action), we use local TCR. Otherwise, we collect the TCRs of the children subtasks and apply the appropriate merge operation based on the task type. We then add the TCRs of the enabling tasks and merge them using *AsyncMerge*. As actions get completed or the deadline associated with any node passes, the local TCR becomes the empty set. These modifications get propagated up to parent nodes and forwarded through enablement relationships so TCRs are an up-to-date picture of capability requirement.

Task TCRs are used to determine when an agent can be released from a goal-constraint tuple in a thread. *AgentSelection* (Alg 4.) takes in the current TCR for a task, the relevant constraints according the the human strategy for the current execution

Alg 1: AsyncMerge(tcrSet)

```
reqSet = ∅; reqTypeSet = ∅
for all tcr ∈ tcrSet do
  for all req ∈ tcr do
    for all reqSet ← reqSet ∪ req do
      reqSet ← reqSet ∪ req
      for all reqType ∈ req do
        reqTypeSet ← reqTypeSet ∪ reqType
for all reqType ∈ reqTypeSet do
  num = 0
  for all req ∈ reqSet do
    if GetType(req) = reqType then
      num =
        Operator[reqType](num, GetNum(req))
  newTcr ← newTcr ∪ NewReq(num, reqType)
return newTcr
```

Alg 2: SyncMerge(tcrSet)

```
reqTypeSet = ∅
for all tcr ∈ tcrSet do
  for all req ∈ tcr do
    for all reqType ∈ reqTypeSet do
    newReqType = reqType
      for all reqElem ∈ GetType(req) do
        newReqType
          ← newReqType ∪ reqElem
      newSet ← newSet ∪ newReqType
  reqTypeSet = newSet
for all reqType ∈ reqTypeSet do
  newTcr ← newTcr ∪ NewReq(1,reqType)
return newTcr
```

Alg 3: UpdateTCR(t)

```
tcrSet = ∅
if isAction(t) then
  tcrSet ← tcrSet ∪ localTCR(t)
else
  for all subTask ∈ ChildrenOf(t)
    tcrSet ← tcrSet ∪ TCR(subTask)
  if TaskType(t) = Synchronization then
    tcrSet = SyncMerge(tcrSet)
  else
    tcrSet = AsyncMerge(tcrSet)
for all enablerTask ∈ EnablersOf(t) do
  tcrSet ← tcrSet ∪ TCR(enablerTask)
return newTcr = AsyncMerge(tcrSet)
```

Alg 4: AgentSelection(agents,tcr,constraints)

```
newTcr = removeDisallowed(tcr,constraints)
remainingSubsets = PowerSet(agents)
for all subset ∈ PowerSet(agents) do
  for all cap ∈ capabilities do
    if NumAgents(subset,cap) <
      Needed(cap,newTcr) then
        remainingSubsets ←
          remainingSubsets \ subset
minSet = GetSmallest(feasibleSet)
return bestSet = GetLowestCost(minSet)
```

Fig. 4. Strategy execution algorithms.

of this task and the available agents. First, we trim the TCR according to the capability constraints from the strategy, e.g., the strategy might disallow performing any repairs at the current time, so we remove requirements related to repair. Then, for each capability, we calculate the needed amount by the maximum <amount> in any <req-elem> with the given <capability> in the modified TCR. We consider all possible subsets of available agents who can meet all the needed amounts of required capability.

We choose the set with the fewest number of agents and break ties using a cost function. Each agent is weighted by the value of their capabilities so that the most valuable agents are not in the selected set. The weights can be customized for each domain. For the field exercises, the value of an agent was the sum of the value of its capabilities where the value of its capability was one over the number of agents possessing the capability. Agents outside the selected set are released from that goal-constraint tuple in the thread. TCRs enable STaC to dynamically manage the execution of the strategy.

Once agents are assigned to goals, the next step is to compute the plans for achieving the goals. These plans specify which tasks should be performed, and which agent should perform them. STaC is agnostic to the methods used to solve these planning problems, so we do not discuss them in detail. We used an MDP planner to construct policies for performing repairs and a custom BDI planner for rescues with simple heuristics for critically injured, which required pairs of agents to load patients into vehicles.

4 Evaluation

Our contribution is a strategy specification language and execution algorithms that enable humans to state a high-level strategy that our system can execute effectively in dynamic and uncertain domains. We evaluate (1) the approach as a whole, (2) the strategy language and (3) the execution algorithms individually.

Effectiveness of the STaC approach. STaC was evaluated in 3 field exercises with 3 teams: a baseline team coordinating using radios, a team assisted by STaC agents, and a third team also assisted by software agents built by a different team using a different approach. Each team received one week of training covering all aspects of the field exercise: the rules, the physical procedures to simulate rescues and perform repairs, etc. They toured the grounds to get familiar with the map and terrain, and conducted three days of full practice scenarios. The teams using agents were trained on the software and used it during the practice scenarios. The radio team was coached by a retired U.S. Marines commander on effective techniques for managing the command center and coordinating using radios.

Each scenario has 15 sites with over 100 repair and injury rescues that required over 400 actions: more than was achievable in the 90 minutes allotted. Disruptions such as road blocks, vehicle breakdowns and agent injuries forced re-planning during execution. The scenarios were significantly more complex than we have space to describe here. The formal description of the field exercise, the scenario specifications and the strategies that the STaC team used for each scenario are provided online[4]. Teams were given scenario specifications only 24 hours before the start of each exercise.

In Figure 5, vertical bars labeled Radio and STaC show scores from the field exercises. The horizontal bars show simulation results discussed later. The results for the third software-agent team were not published but were lower than those of the radio team. We outperformed the radio team in all three scenarios (by 26%, 26%, and 24%). It is difficult to assess whether the scores are good in an absolute sense as calculating an optimal solution, even if centralized, is infeasible for scenarios of such complexity.

The field exercise results provide compelling evidence for the effectiveness of the STaC approach because (1) the radio team represents a reasonable baseline as they were extensively trained and operated with technology often in use today in such scenarios, and (2) the other agent-based team was unable to outperform the radio team in any scenario.

[4] http://www.isi.edu/∼maheswar/prima-2011/

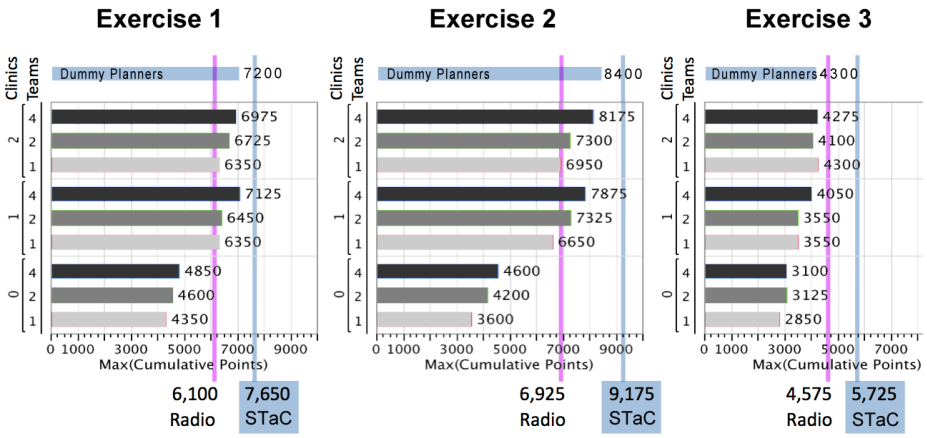

Fig. 5. Field Exercise Evaluation Results

Effectiveness of the STaC strategy specification language. The complexity of the scenarios for the field exercise required sophisticated strategies to optimize performance. A strategy we employed for one exercise first created three subteams for gas, power and water to restore the "mains" for each service. This is a prerequisite for restoring substations and making strategically important clinics operational[5]. One of the remaining agents traveled to perform only isolations and the other began early deadline rescues. The agents then reformed into one large team to make the clinics operational. They then repartitioned into gas, power, water teams that only performed high-value repairs and a medical team that only rescued high-value critically injured. When the deadline was approaching, they became two large teams that performed only injured rescue. This required intricate orchestration to ensure that isolation, surveys, repairs, rescues, and repair kit management which all needed different combinations of skills were satisfied in the right sequence without causing excessive delays.

The goal of the first set of simulations was to assess the importance of the quality of the strategies. In the simulations, we used the logs of the real exercises to compute average values for the duration of all activities, and used the same outcomes and disruptive events that the evaluators had used in the field exercises. Our simulations of the STaC field exercise strategies were within 4% of the real results, so we posit that it is reasonable to expect a similar margin of error in the simulations of the other strategies.

We designed a collection of simple strategies organized along two dimensions: the number of clinics that would be made operational (0, 1 and 2), and the number of independent subteams (1, 2 and 4). Each subteam had enough capabilities to achieve the enabling isolation and survey goals. All repairs for clinics were done first, as both the radio team and the STaC team did in the field exercises. Sites were ordered by total expected utility from rescuing all injured and repairing all substations and assigned to

[5] In each scenario, it was crucial to determine whether to restore any of two damaged clinics. Each required commitment of many agents and capabilities, but once operational, injured could be moved there rather than to the hospital which may be much further away.

subteams in a round-robin manner from most to least valuable. Thus, each subteam had a single thread. When a subteam traveled to a site, it would attempt all the goals for the site, irrespective of utility, i.e., no constraints.

These simpler strategies performed worse than those used by the STaC team on the field (vertical bars labelled STaC). The greater the use of STaC features, e.g., subteaming and reformation (used after making clinics operational), the better the performance. These simulations suggest that exploiting richer strategy structures is valuable.

Effectiveness of the STaC execution algorithms. Several of the simpler, simulated strategies obtained scores that were better or comparable to the scores of the radio team. We attribute these results to the effectiveness of the execution algorithms. The radio team was not required to write down their strategies, however, informal conversations with their team members revealed that their strategies were more sophisticated than our simpler strategies. For example, in one scenario, they designated their least capable agent as a warehouse runner who got repair parts and delivered them wherever they were needed. The cases where the simpler strategies outperformed the radio team suggest that the radio team was not able to effectively carry out their strategies. Their commanders often had to handle multiple radio calls simultaneously and put agents on hold, wasting valuable time, and sometimes made incorrect decisions sending agents to sites where they were not needed. In the third scenario, which emphasized injury rescues, none of the simple strategies would have outperformed the radio team. The simple strategies had agents rescuing all injured at a site before going to a different site, and thus missed the deadlines for many injured. It was important to round-robin multiple rescue teams over sites to ensure saving the injured with urgent deadlines.

To evaluate the effect of the TCR agent-to-goal assignment algorithm, we also ran simulations where we replaced the planners for repairs and rescues with dummy planners. In these simulations, the agent-to-goal assignments were done using the TCR algorithms, but task assignments were done using simple algorithms. For repairs, we randomly selected repair tasks. For injured, we rescued the injured with the most urgent deadline and never considered waiting to have two agents simultaneously ready to load the more valuable critically injured. The results show that in the first two exercises the effect of the automated planner was relatively minor: the STaC strategies executed using the TCR algorithms (and dummy planners) would still have outperformed the radio team. In the third exercise that emphasized rescues, the greedy rescue algorithm did have a very detrimental effect.

5 Related Work

Real-world applications of the type addressed here are challenging because they must work in dynamic and uncertain environments [11, 14, 18]. The requirements for a richer model for actions and time are generally problematic for current planning frameworks based on MDPs, POMDPs, SAT, CSP, planning graphs or state-space encodings [4]. The main obstacle is scale as it is currently infeasible for a fully automated system to effectively reason about all the possible futures that may arise during execution. STaC is an approach that enables partitioning these large problems into smaller subproblems where these approaches can be used effectively.

Mixed-initiative approaches, where humans and systems collaborate in the development and management of solutions, are viable alternatives to tackle complex real world problems [12, 1, 9, 18]. While mixed-initiative planning [5] is an established field, the type of problems we address with STaC bring new challenges. These include agent teams with heterogeneous and dynamic capabilities, tasks with uncertain durations and outcomes, and most importantly, planning over incomplete reward structures. In these problems, the goals that give reward to teams are not revealed *a priori*, but have to be discovered during execution.

Several mixed-initiative planning systems have been developed where users and software interact to construct and refine plans by adding and removing activities during execution while minimizing changes to a reference plan or schedule [1, 9, 12]. In the MAPGEN framework [1], humans control the construction of plans offline, while automated planning and reasoning capabilities are used to actively enforce constraints to generate safe plans for the Mars rovers. The output of the collaboration is a concrete plan. In the STaC approach, the human planner provides a high-level plan (strategy) that the system fleshes out during execution to produce concrete plans to satisfy the particular goals that are revealed during execution.

O-Plan is a knowledge-based mixed-initiative framework, which uses an agenda of outstanding (goal) issues to incrementally refine and repair plans by tightening constraints [16]. In O-Plan, a task assigner or commander specifies the set of tasks to consider; and a planning agent, using multiple classes of constraints, restricts the range of plans generated for the tasks specified [17]. An O-Plan-style planner could be used for dynamic problems by allowing the human planner to select goals as they are revealed and work with the automated planner to construct plans to achieve them. This approach involves human planners during execution, giving them more control than human planners have in the STaC approach. This approach could be effective in less dynamic situations where a human-in-the-loop planner does not become a bottleneck and where human input into the agent to task assignment is worthwhile. In our field exercises, a human planning during execution would have become a bottleneck. In the STaC approach, the role of human planners during execution is to oversee the execution of the strategy, assess its effectiveness and adjust the strategy accordingly. Our current system enables human planners to oversee the execution of the strategy giving commanders full situational awareness. In future work we plan to enable dynamically adjusting the strategy and distributing it to the agents during execution.

STaC can be seen as a particular form of control-knowledge that leverages the structure of domains. Work on using domain control-knowledge to solve complex planning problems has been extensively documented [13, 2, 16], and has been shown to convert intractable planning problems into a tractable ones [2, 13]. Two planning systems that consider human intuition for problem solving are SHOP2 [13] and TLPLAN [2]. While SHOP2's search space consists only of those nodes that are reachable, TLPLAN uses its formulas to tell which part of the search space should be avoided. Similarly, STaC strategies can be viewed as a type of control knowledge that specifies constraints on the goals that should be attempted, the order for attempting them and the resources that can be used to fulfill them. One key difference is that STaC strategies allow expressing this control knowledge over goals that have not yet been discovered.

Systems like DEFACTO [15] study the role of adjustable autonomy to improve human/agent collaboration for disaster management. While adjustable autonomy is not the focus of our work, STaC enables human planners to delegate significant control to the system by imposing fewer constraints on strategies, or to exert more control by imposing additional constraints. It would be interesting to explore a DEFACTO-style approach to adjustable autonomy in STaC whereby humans would be consulted on agent-to-goal assignments in specific contexts (e.g., when the decision is critical and human planners have time to intervene).

6 Conclusions and Future Work

STaC is an approach to address multi-agent planning problems in dynamic environments where most goals are revealed during execution, where uncertainty in the duration and outcome of actions plays a significant role, and where unexpected events can cause large disruptions to existing plans. The key insight of our approach is to (1) give human planners a rich language to control the assignment of agents to goals (strategy language), (2) allow the system to assign agents to goals during execution, tuning the assignment to the evolving execution state (TCR algorithms), and (3) use off-the-shelf planners to select and plan the tasks to achieve these goals. Evaluations in 3 realistic field exercises showed that STaC is a promising approach. The STaC strategy language enabled us to encode sophisticated strategies. Additional evaluations using simulations of the field exercises showed that the TCR algorithms provided significant benefit. These simulations showed how in 2 of the exercises, simple strategies executed using the TCR algorithms produced better scores than more sophisticated strategies used by the radio team, but executed without the help of our algorithms. The evaluations also showed how the STaC approach effectively partitioned the large and complex planning problems to create small subproblems that traditional planning technology (MDP and BDI) could address effectively. The STaC system outperformed an extensively-trained team coordinating with radios and a traditional command-center organization, and an agent-assisted team using a different approach.

The evaluations suggest that the following enhancements would be valuable. First, our agent-to-goal assignment algorithms are based on the capability requirements of the current goals without consideration of capability requirements of future goals in the strategy. This local optimization may lead to situations where agents with important capabilities for future goals are inappropriately assigned to the current goals. Fast algorithms to reason more globally are desirable. Second, strategies are easy to understand from a local point of view, i.e., each strategy thread is easy to understand, but the evolution of a strategy is hard to predict. In the field exercises, we used detailed simulations of strategies and visualizations to understand their evolution and to judge their effectiveness. Each simulation ran for 20 minutes, so it was only possible to run a small number of simulations before an operation. It is infeasible to run simulations of this sort during execution to evaluate potential revisions to a strategy. Less detailed and significantly faster simulations coupled with strategy analysis algorithms would enable rapid evaluation of alternative strategy adjustments under different dynamic evolutions. This would help human planners evaluate and enact strategy adjustments during execution.

References

1. Ai-Chang, M., Bresina, J., Charest, L., Chase, A., jung Hsu, J.C., Jonsson, A., Kanefsky, B., Morris, P., Rajan, K., Yglesias, J., Chafin, B.G., Dias, W.C., Maldague, P.F.: Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission. IEEE Intelligent Systems 19(1), 8–12 (2004)
2. Bacchus, F., Kabanza, F., Sherbrooke, U.D.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence 116, 2000 (1999)
3. Boddy, M., Horling, B., Phelps, J., Goldman, R.P., Vincent, R., Long, A.C., Kohout, B., Maheswaran, R.: CTAEMS language specification: Version 2.04 (2007)
4. Bresina, J., Meuleauy, N., Ramakrishnan, S., Smith, D., Washingtonx, R.: Planning under continuous time and resource uncertainty: A challenge for ai. In: In Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence. pp. 77–84. Morgan Kaufmann (2002)
5. Burstein, M.H., McDermott, D.V.: Issues in the development of human-computer mixed-initiative planning systems. In: Cognitive Technology (1996)
6. Chen, Y., Wah, B., Hsu, C.W.: Temporal planning using subgoal partitioning and resolution in sgplan. Journal of Artificial Intelligence Research (JAIR) 26(1), 323–369 (2006)
7. Fox, M., Long, D.: Modelling mixed discrete-continuous domains for planning. Journal of Artificial Intelligence Research (JAIR) 27 (2006)
8. Gerevini, A., Saetti, A., Serina, I., Toninelli, P.: Fast planning in domains with derived predicates: an approach based on rule-action graphs and local search. In: Proceedings of the 20th national conference on Artificial intelligence (AAAI). pp. 1157–1162. AAAI Press (2005)
9. Hayes, C., Larson, A., Ravinder, U.: Weasel: A mipas system to assist in military planning. In: ICAPS05 MIPAS Workshop (WS3) (2005)
10. Jin, J., Sanchez, R., Maheswaran, R.T., Szekely, P.A.: Vizscript: on the creation of efficient visualizations for understanding complex multi-agent systems. In: Intelligent User Interfaces. pp. 40–49 (2008)
11. Kleiner, A., Freiburg, U.: Wearable computing meets multiagent systems: A real-world interface for the robocuprescue simulation platform. In: First International Workshop on Agent Technology for Disaster Management at AAMAS06 (2006)
12. Myers, K.L., Jarvis, P.A., Mabry, W., Michael, T., Wolverton, J.: A mixed-initiative framework for robust plan sketching. In: Proceedings of the 13th International Conference on Automated Planning and Scheduling (2003)
13. Nau, D., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D., Yaman, F.: Shop2: An htn planning system. Journal of Artificial Intelligence Research 20, 379–404 (2003)
14. Nourbakhsh, I.R., Sycara, K., Koes, M., Yong, M., Lewis, M., Burion, S.: Human-robot teaming for search and rescue. IEEE Pervasive Computing 4(1), 72–78 (2005)
15. Schurr, N., Marecki, J., Scerri, P., Lewis, J., Tambe, M.: The DEFACTO System: Coordinating human-agent teams for the future of disaster response, chap. Programming multi-agent systems: Third international workshop. Springer Press (2005)
16. Tate, A., Drabble, B., Dalton, J.: O-plan: A knowledge-based planner and its application to logistics. In: Advanced Planning Technology, pp. 259–266. AAAI Press (1996)
17. Tate, A.: Multi-agent planning via mutually constraining the space of behaviour. Tech. rep., In Constraints and Agents: Papers from the 1997 AAAI Workshop (1997)
18. Veloso, M.M., Mulvehill, A.M., Cox, M.T.: Rationale-supported mixed-initiative case-based planning. In: in Proceedings of the Ninth Annual Conference on Innovative Applications of Artificial Intelligence. Menlo. pp. 1072–1077. AAAI Press (1997)