Task Assignment in AOSNs: A Constraint-based Approach

Elise H. Turner*
Computer Science Department
University of Maine
Orono, ME 04468

ABSTRACT

To allow multi-component systems, such as Autonomous Oceanographic Sampling Networks (AOSN), to autonomously organize and reorganize, there must be a method for assigning components of the systems to tasks. In this paper, we present a method for making these task assignments. Our method is based on *constrainted heuristic search (CHS)* [1]. We represent the problem of task assignment as a constraint satisfaction problem using the CHS formalism. The method is extended to handle alternative methods available for performing the mission.

1 Introduction

We envision an Adaptive Oceanographic Sampling Network (AOSN) which allows differently-abled Vehicles and Instrument Platforms (VIPs) to be adaptively configured to perform a variety of data collection tasks. The AOSN will autonomously organize and reorganize to adapt to changes in the task and in the composition of the AOSN. By creating or changing organizations autonomously, the system frees the user from possessing the expertise needed to organize the VIPs and allows the system to respond to changes even when no user is present.

We are using a two-level approach to organization and reorganization [2; 3]. The *meta-level organization (MLO)* is composed solely of VIPs that are capable of reasoning about other agents, the task, and the organization. Its purpose is to create the organization of all VIPs that will perform the tasks of the AOSN. The MLO is organized for flexibility. It must be able to create an organization for any task given any number and type of VIPs, whenever possible. As the situation changes, the MLO may be required to create several organizations for several different tasks. The MLO will also need to adapt to a changing situation. The MLO may be expanded or contracted as potential members enter and exit the AOSN.

The *task-level organization (TLO)* performs the work of the AOSN. It should be designed for efficiency. To create the TLO, the MLO must decide on an *organizational structure* that determines the *roles* of members of the organization and the lines of communication between the VIPs filling those roles. These roles are the individual subtasks of the assigned domain task and the managerial tasks required to manage the VIPs. For example, a strict hierarchy is an organizational structure in which VIPs filling roles directly related to the AOSN's task communicate with specific VIPs who

*This work was funded in part by contract N0001–14–96–1–5009 from the Office of Naval Research. Thanks to Dick Blidberg, Steve Chappell, Jim Kadin, Jim Jalbert, Roy Turner and the UMaine CDPS group for many insightful comments on this work. Special thanks to Steve Chappell who is responsible for the programming on this project.

have the task of managing them. The MLO must also assign particular VIPs to the tasks that must be performed. We call this *task assignment*.

In this paper, we will describe a technique for creating a task assignment. Our technique is based on *Constrained Heuristic Search (CHS)* [1], a problem solving method that combines constraint satisfaction and heuristic search. CHS must be extended for task assignment because tasks can often be done in many different ways. In Section 2 we present a brief introduction to CHS and explain why we chose it for task assignment. In Section 3 we discuss the need to extend CHS for our problem and present the algorithm that we are currently implementing. In Section 4 we illustrate our technique with an example. We conclude in Section 5.

2 Constrained Heuristic Search

2.1 Introduction to Constrained Heuristic Search

Many problems can be characterized as a set of variables that must have values assigned to them and a set of constraints on those values. There are many approaches to solving *constraint satisfaction problems (CSPs)*, each appropriate for certain tasks and types of constraints. One approach is to see the CSP as a state space search problem. The initial state is some assignment of sets of potential values to the variables; the goal is an assignment of a single value to each variable so that the value does not violate any constraints. Operators on the states reduce the set of potential values of some particular variable. Applying these operators leads to intermediate states with different partial assignments. The search space is pruned in two ways. First, *constraint propagation* is used to remove values from the sets of potential values that no longer satisfy the constraints. Second, specific, well-studied heuristics can be used to choose the assignment that should be tried next.

Constrained heuristic search (CHS) combines constraint satisfaction and heuristic search. Search states are *constraint graphs*. As in standard CSP's, variables are represented as variable nodes and their domains are their remaining potential values. Constraints are also represented by nodes which are adjacent to the variables whose values they restrict. There are also *satisfiability* nodes which are adjacent to the constraints which they AND or OR together. The topology of the constraint graph can be characterized by a set of *textures*. Heuristics are developed to approximate these textures. The heuristics can then be applied to determine which operator should be applied and what search state can be generated next. Because the operators choose the variables and values that should be restricted to percipitate constraint propagation, these choice points are now treated within the well-studied technique of heuristic search.

The algorithm for CHS appears below. We quote from Fox *et al.'s* paper [1], altering it here only to number the steps for easy reference.

1. An initial state is defined composed of a problem topology.
2. Constraint propagation is performed within the state.
3. Texture measures and the problem objective are evaluated for the state's topology.
4. Operators are matched against the state's topology, and
5. A variable node/operator pair is selected and the operator is applied.

We have selected CHS for task assignment for three reasons:

1. Constraints are handled efficiently through the constraint propagation portion of the technique.
2. A solution will be found if one exists.

3. A distributed form of CHS, DCHS [4], has been developed which can be used to distribute the work of task assignment across the entire MLO.

However, CHS must be extended for use in task assignment. CSP's in general, and CHS in particular, assume that all variables will be assigned a value. For task assignment, this is not the case. The mission is broken into the tasks which must be performed to complete the mission. These tasks are further broken down to find the primitive tasks, or *executable actions*, which the VIPs can perform. This is called the *task decomposition*. Since there may be many ways to achieve any task, the task decomposition includes many alternatives. Only one alternative for each task needs to be performed, so only the executable actions associated with that alternative need to be assigned to a VIP.

Consequently, the CHS algorithm must be extended to handle the fact that all executable actions will not, ultimately, be included in the task assignment. We could extend the formalism for the constraint graph to add satisfiability nodes that are adjacent to variables. This leads to a confusing graph with many superfluous nodes that affect the solution even though they will not be part of the actual task assignment. Instead, we extend the algorithm by adding an operator that selects the appropriate capabilities and adds them to the constraint graph. The capabilities can be selected following the textures that have been suggested for CHS so that the selection will not hurt the efficiency of CHS.

3 CHS for Task Assignment

We can divide our work into two parts: formalizing task assignment in a constraint graph and extending CHS to include the **select-capability** operator. Although we intend to integrate this operator in to the CHS algorithm at Steps 4 and 5, we currently implement it as a preprocessing step. This simplifies the design and allows us to obtain preliminary results.

3.1 Representing Task Assignment as a Constraint Graph

It is natural to view task assignment as a constraint satisfaction problem. The variables of the problem are the roles in the organizational structure that must be filled. These roles can be characterized by the *capabilities* required to fill the roles. The domain of these variables can be defined initially as the VIPs which have the needed capability.

Constraints on the variables restrict the way in which roles associated with particular tasks can interact and capture the resource limitations of the VIPs.[1] Constraints can be easily added to any constraint satisfaction problem, and we expect to find more constraints as we continue this work. For now, we have implemented a single *resource constraint*. The resource constraint checks that a VIP can perform all of the tasks to which it is assigned. We currently use a simple representation of resources and work space to check if the VIP has to work beyond its resources or has to move too far between tasks. The VIP has some number of resource units that can be used during the mission, and each task requires some number of units. We also require that all of the tasks of the VIP be within that VIP's *volume-of-work*. The resource constraint holds between all variables for which the VIP is in the domain.

[1] We can also represent this limitation on the initial domain, that all VIPs in the domain have the required capability, as a unary constraint on the variables. We choose instead to limit the domain because this is useful for the extensions discussed in Section 3 and can be used in a distributed version of CHS.

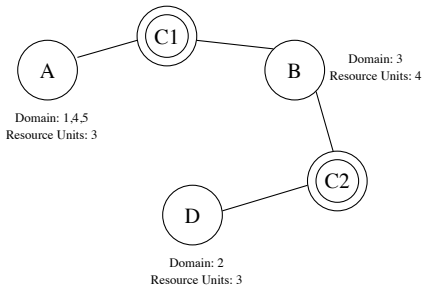

Figure 1: A simple constraint graph for task assignment.

A constraint graph for a task assignment problem is shown in Figure 1. The figure shows three tasks that must be performed by some VIPs. These tasks require the capabilities, A,B and D. The variable nodes are labeled with the capability. The identifying numbers of the VIPs in the variable's domain and the resource units needed to perform the task are also shown. The constraints are the nodes C1 and C2. C1 represents the resource constraints on VIP4. It is adjacent to A and B because VIP4 is in the domain of both of these variables. Similarly, C2 represents the resource constraints on VIP2. There are no constraints for the other VIPs because they are in the domain of only one variable.

We are currently using very simple heuristics which are widely used in constraint satisfaction problems. When necessary, we choose a value for a variable by finding the variable with the fewest values remaining in its domain and then choosing the value that remains in the domain of the fewest variables.

3.2 Extending CHS

We extend CHS by adding a **select-capability** operator that will choose the capabilities that will be included in the constraint graph. The capabilities are selected from the task decomposition. When a capability is selected, it is added to the constraint graph. The process is repeated until all the capabilities required to perform some decomposition of the mission have been added to the constraint graph.

3.2.1 Representing the Task Decomposition

The task decomposition must be created by a planner that can reason about how the mission can be performed. We expect the MLO to construct the task decomposition during a previous phase of TLO creation so that it is available during task assignment.

The task decomposition is represented in an AND-OR tree like the one in Figure 2. A task, such as the mission, is specified in terms of the steps which can be performed to accomplish it. In out example, T1 and T2 must both be performed to accomplish the mission. Since both tasks must be performed, they are ANDed together, as represented by the link between the nodes. Tasks can also have alternative methods for achieving them. For example, T1 can be performed either by performing Alt-T1-1 or Alt-T1-2. The mission is decomposed to executable steps at the leaves of the tree. In our work, executable steps are represented in *generic behaviors*, or *capabilities* [5].

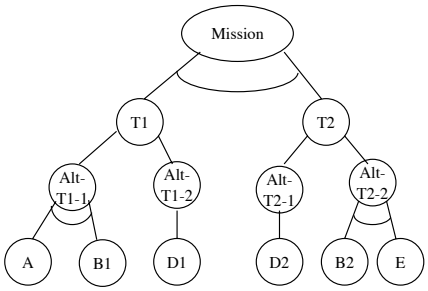

Figure 2: The task decomposition of a mission.

1. If no children for node, at leaf. Return number of VIPs which can perform the task.
2. If the node is the root, return the child with the maximum value as the selected alternative.
3. If node is an AND node, return the minimum value of all of its children
4. If node is an OR node, return the maximum value of all of its children

Figure 3: Selecting an Alternative.

3.3 Selecting the Capability

We would like to select capabilities in a way that takes their future role in the constraint graph into account. For that reason, we choose among the capabilities using measures that reflect textures of the constraint graph.

Selecting capabilities is complicated because the task decomposition is represented in an AND-OR tree. Our algorithm starts at the root mission node and selects between its children. If the children are ORed together, a child is selected using the *select-alternative* algorithm. If the children are ANDed together, a child is selected using the *select-subtask* algorithm. The selected child is then used as the root of the tree and the process continues until a leaf is reached.

Select Alternative

Following the *value goodness* texture [1], we choose the alternative that can be satisfied by the greatest number of values. This gives the constraint algorithm the most flexibility. Figure 3 shows the recursive algorithm used for selecting between alternatives. Since the VIPs that can perform the task give the potential values for the variables, they provide the basis for the selection. These values are propagated up the tree from the leaves so the interdependence of the ANDed siblings can be taken into account. This is standard for AND-OR trees. We assume that the VIPs which can perform the capabilities in the task decomposition are associated with those capabilities before any capability selection is performed.

Select Subtask

Following the *constraint reliance* texture [1], we choose the subtask with the fewest possible alternatives. Since this subtask is the most constrained, it needs to be added to the constraint graph early. Figure 4 shows the recursive algorithm for selecting the subtask.

1. If no children for node, at leaf. Return 1.
2. If the node is the root, return the child with the minimum value as the selected subtask.
3. If node is an AND, return the minimum value of all its children.
4. If node is an OR, return the sum of the values for all its children.

Figure 4: Selecting a Subtask.

3.3.1 Adding the Capability to the Constraint Graph

When a capability is added to the constraint graph, it must be connected to constraints which affect it. For each VIP in its domain, it finds the appropriate constraint and places itself on that constraint's adjacency list. If by adding this capability, the limit of variable's resources is exceeded, it removes the variable from the domain of the capability with the largest domain. This variable follows the *value goodness* texture [1]. If domains are of the same size, the value is removed from the domain for the capability which requires the fewest resources. If a variable is removed from any domain, constraints are propagated.

Any capability that is in the task decomposition could be part of the final method for achieving the mission and, consequently, is a *potential* node in the constraint graph. When the MLO selects a capability to add to the constraint graph, it becomes a *committed* node because the system is committed to including it in the final solution. We do not want potential nodes to restrict the values of committed nodes because the potential nodes may not be included in the solution. On the other hand, if we allow the committed nodes to restrict the values of potential nodes, we get better information for selecting capabilities to add to the constraint graph. In our current implementation, constraints are only propagated between committed nodes. This is the simplest method we can adopt for our preliminary system. In the near future, we will implement and evaluate propagating constraints from committed to potential nodes.

3.3.2 Selecting Additional Capabilities

All capabilities required to perform the mission, and only those required to perform the mission, should be added to the constraint graph. To keep track of the capabilities that remain, we copy the task decomposition tree and remove nodes as they are added to the constraint graph. When no nodes remain in the copy, we have selected all of the capabilities.

Siblings of the node are removed from the task decomposition copy if they are ORed to that node. They remain in the copy if they are ANDed to the node. A parent is removed when all of its children are removed. In this way, nodes that will not affect the solution are removed from consideration, and nodes that must be selected in conjunction with previously-selected nodes remain in the graph.

4 An Example

We illustrate our method of task assignment with an example. Suppose the task decomposition of Figure 2 is provided by the MLO. Next, VIPs are identified which have the required capabilities. The annotated task decomposition graph is shown in Figure 5. We assume all VIPs have 5 resource units. For simplicity, we omit constraints on the volume of work.

First, a capability is selected. Since all the values for the textures remain the same until the graph is changed by removing a leaf, we can calculate both values once and use them throughout

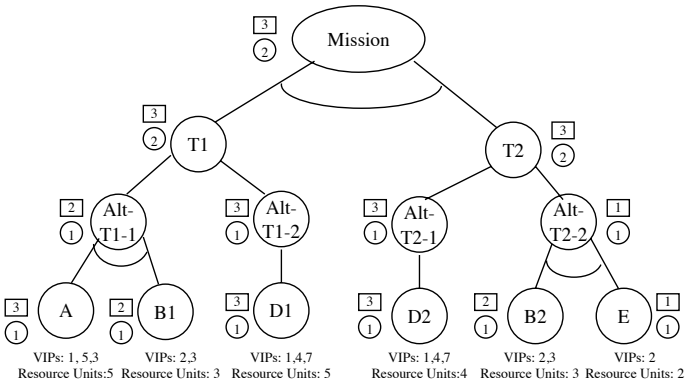

Figure 5: Annotated task decomposition for the example mission.

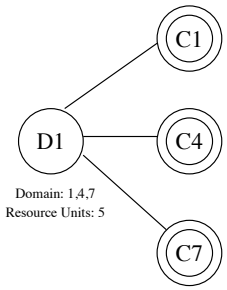

Figure 6: Constraint graph after D1 has been added.

the entire selection process for one capability. The values are shown at each node. Values for **select-alternative** appear in squares; values for **select-subtask** appear in circles. Since both T1 and T2 have the same value as subtasks, T1 is arbitrarily selected at Mission. It's children are ORed together, so the alternative with the maximum value, Alt-T1-2, is selected. There is only one alternative here, so D1 is added to the constraint graph.

The new constraint graph is shown in Figure 6. For bookkeeping purposes constraints for each VIP are added to the graph, even though the values have already been checked to ensure that the resources needed do not exceed the VIPs' limits. These constraints will make it easier to add new capabilities. In the figure, constraints are labeled "Cx" where x is the number of the VIP associated with the constraint. No constraints are propagated.

The selected node is removed from the task decomposition tree. Since it has no siblings, when D1 is removed, Alt-T1-2 becomes childless and is removed. Alt-T1-2's siblings are ORed together, so they are removed. This leaves T1 childless, so it is removed. T1 is ANDed to T2, so T2 remains as the only child of the mission. Figure 7 shows the new graph. The values for **select-alternative** and **select-subtask** must be recalculated. In this case, they have not changed because no values were removed from any capabilities[2] and leaves were only removed from a node, Mission, which received the same value from its remaining child.

[2]This could happen if we were propagating constraints from committed nodes to potential nodes.

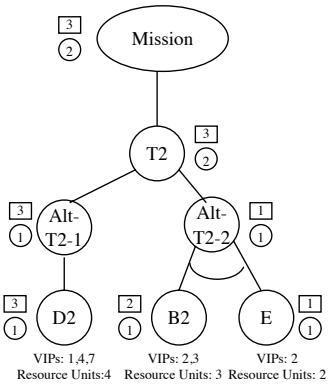

Figure 7: Task decomposition after removal of D1.

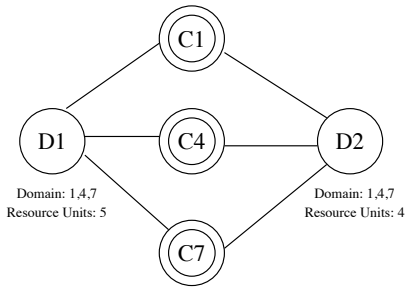

Figure 8: Constraint graph after D2 has been added.

Since Mission has only one child it is selected using **select-subtask**. T2's children are ORed together, so Alt-T2-1 is selected. Its single alternative is selected and added to the constraint graph, as shown in Figure 8. When D2 is added to the constraint graph, constraints are checked. All of the constraints are violated because resource units required for D1 and D2 are greater than any VIP's allotment of 5. When D2 is made adjacent to C1, the system notices the constraint violation. Both domains have the same number of values, so we remove VIP1 from the domain which requires the least resources. VIP1 is removed from the domain of D2. D2 is no longer adjacent to that constraint. When D2 is made adjacent to C4, the resource limits of VIP4 are also violated. Because D1 has the most values in its domain, VIP4 is removed from that domain. Circumstances for C7 are similar to those for C1, so VIP7 is removed from the domain of D2. The resulting constraint graph is shown in Figure 9.

When this capability is removed from the task decomposition graph, all of Mission's children are removed, so Mission is removed. Since no nodes remain in the task decomposition, the formation of the constraint graph is complete. We now proceed with the original CHS algorithm. Using the same textures as for propagating constraints above, we choose a value for the only variable node whose domain requires further restriction, D1. Since both of the remaining values appear equally useful according to the heuristics, we arbitrarily select VIP1.

Our method has found an acceptable task assignment. However, it may appear odd that the

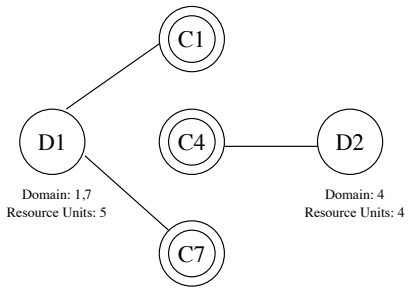

Figure 9: Constraint graph after constraints propagated for D2.

same capability was selected for both tasks, requiring the same VIPs, when other capabilities could have been chosen. There are two ways to ensure that this does not create problems. We will examine them in future work. First, we will look at propagating constraints to potential values. This will remove values form the domains of capabilities still in the task decomposition tree when a VIP's resources are exceeded. This, in turn, will decrease the desirability of adding the capability to the constraint graph. The second possibility is to change our heuristics for selecting capabilities. This will give us better information, but will increase the complexity for our heuristics.

5 Conclusions

We are currently implementing a version of our approach in CommonLISP. This implementation will be added to our simulator [2] to perform task assignment. The current implementation allows us to get initial results and provides a framework for future experimentation.

As work continues on this project, we will refine our extension to CHS. We will also implement and test heuristics for different textures to be applied during both the original CHS and our extension. In addition, we will integrate our **select-capability** operator with CHS. Finally, we plan to implement a version of task assignment which can be distributed between all members of the MLO.

References

[1] M. S. Fox, N. Sadeh, and C. Baykan. Constrained heuristic search. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1989.

[2] R. M. Turner, E. H. Turner, and D. R. Blidberg. Organization and reorganizing of autonomous oceanographic sampling networks. In *Proceedings of the 1996 Symposium on Autonomous Underwater Vehicle Technology (AUV'96)*, Monterey, CA, 1996.

[3] R. M. Turner and E. H. Turner. Adaptive organization and reorganization of autonomous oceanographic sampling networks. Submitted to the *Journal of Applied Intelligence.*, submitted.

[4] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, 1991.

[5] R. M. Turner, D. R. Blidberg, S. G. Chappell, and J. C. Jalbert. Generic behaviors: An approach to modularity in intelligent systems control. In *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology (AUV'93)*, Durham, New Hampshire, 1993.