A Constraint-Based Approach to Assigning System Components to Tasks*

Elise H. Turner and Roy M. Turner

Department of Computer Science, University of Maine, Orono, ME 04469, USA
{eht, rmt}@umcs.maine.edu

Abstract. In multi-component systems, individual components must be assigned to the tasks that they are to perform. In many applications, there are several possible task decompositions that could be used to achieve the task, and there are limited resources available throughout the system. We present a technique for making task assignments under these conditions. Constraint satisfaction is used to assign components to particular tasks. The task decomposition is selected using heuristics to suggest a decomposition for which an assignment can be found efficiently. We have applied our technique to the problem of task assignment in systems of underwater robots and instrument platforms working together to collect data in the ocean.

1 Introduction

In this paper, we address the problem of assigning tasks to components of systems comprised of multiple robots and instruments. Our method views each component as a collection of capabilities. Each component is also expected to have limited resources which can be expended during the mission. The overall mission of the system is decomposed into subtasks which are further decomposed into the capabilities which are required to achieve them. Task assignment is viewed as a constraint satisfaction problem in which the assignment of a component to a particular task is constrained by the capabilities and resource limitations of the component.

Our method has been developed specifically for *Autonomous Oceanographic Sampling Networks (AOSNs)* [1]. AOSNs are multi-component systems that will be deployed for long durations to collect data in the ocean. We will call specific high-level tasks *missions* of the system to distinguish them from the subtasks which will be assigned to individual components. Components of the AOSN include both underwater Vehicles and non-mobile Instrument Platforms (VIPs). Because of the limited number of underwater vehicles in existence, and because of the expense of developing new VIPs, the AOSN should be able to take advantage of slack resources in the ocean engineering community. Consequently, the configuration of the system will be driven by the availability of components.

* 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, and the UMaine CDPS group for many insightful comments on this work.

In *Lecture Notes in Artificial Intelligence 1415: Methodology and Tools in Knowledge-Based Systems*, J. Mira, A.P. del Pobil, and M. Ali (eds.), Springer, New York, 1998. (Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA–98–AIE), Benicàssim, Spain, June, 1998.) Vol. I, pp. 538–547.

AOSNs share several important characteristics with other multi-component systems. These include:

1. Each component will have a collection of capabilities, usually many more than one. However, not all components have the capability to reason or to act autonomously in the world.
2. Because of resource limitations, not all components will be able to contribute all of their capabilities to the mission.
3. There may be some special-purpose components which have rare capabilities that are needed for the mission.
4. The total number of resources and capabilities in the system may be close to the total needed to perform the mission.
5. The task will have several known decompositions, each of which will require a different combination of capabilities.
6. The system must be able to organize itself autonomously. This may be necessary for the initial organization or for the system to respond to changes in the user's goals or in the environment.

In the next section, we describe our technique in more detail. We present an example in Sect. 3. In Sect. 4, we compare our work to related systems. We conclude with a discussion of our results and plans for future work.

2 Assigning Components to Tasks

We assume that each mission can be achieved using one of several *task decompositions*. A task decomposition breaks down a task into subtasks, then those into their subtasks, etc. Each task decomposition requires different capabilities and demands different levels of resources. The problem of task assignment can be seen as having two parts: selecting the appropriate task decomposition and assigning VIPs to the capabilities required for that alternative. Since the discussion of selecting the decomposition requires an understanding of how assignment is done, we present that first.

2.1 Assigning VIPs to Capabilities

The assignment of VIPs to capabilities can be represented as a constraint satisfaction problem in a straightforward way. The variables in the constraint satisfaction problem are the capabilities required. The value of each variable is the VIP which is assigned to contribute that capability to the task. There are three constraints on this assignment:

1. The VIP assigned to the variable must have the required capability.
2. The VIP must have the required resources available.
3. The total resources required by all of the capabilities to which the VIP is assigned cannot exceed the VIP's available resources.

The first two are unary constraints. They are checked when VIPs are assigned to the initial domains of the variables, which are created when the task decomposition tree is formed. The third is an n-ary constraint between all variables which have the given VIP in their domains. There will be one such constraint for each VIP.

We use the *constrained heuristic search* (CHS) [2] formalism for solving the constraint satisfaction problem. CHS combines constraint satisfaction and heuristic search. Search states are *constraint graphs*. As in standard CSPs, variables are represented as nodes and their domains are their remaining potential values. In CHS, constraints are also represented by nodes. Constraint nodes are adjacent to the variable nodes whose values they restrict. This allows constraints to be added to the constraint graph after some constraint propagation has taken place. 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 select the operators used to generate the next search state. Constraints are propagated within each newly created state. Any search mechanism can be applied to the search space.

The heuristics that we apply during constraint propagation select the unassigned variable with the fewest values left in its domain and assign to it the value that is a member of the fewest domains. The state space is searched using a version of hill-climbing which allows backtracking to previous choice points.

2.2 Selecting the Task Decomposition

We assume that the VIPs, their capabilities and resource limitations, and the task decomposition tree for the mission are known before task assignment begins [3]. The task decomposition tree has the form shown in Fig. 1. The subtasks that must be performed to accomplish a task are the ANDed children of that task. In the figure, T1 and T2 must be performed in order to perform the mission. Alternative methods for performing a task are the ORed children of the parent task. For example, Alt-T1-1 and Alt-T1-2 are two alternatives for performing T1. All of the children of the node are either ANDed or ORed together; there are no mixed nodes. The children of the root can be either ANDed or ORed together. The leaves of the task decomposition tree are variable nodes representing capabilities required to achieve the parent task. These nodes can be moved to the constraint graph. The capabilities are shown in our figure by letters. We have numbered the capabilities which appear more than once to indicate that they are distinct usages of the same capability and to identify the node in our discussion of the example. In addition to the capability, the number of "resource units" required[1] and the domain of the variable are associated with these nodes. The task decomposition tree represents the ways in which the mission can be performed. For our example, the mission can be performed using the following collections of capabilities: {A, B1, D2}; {A, B1, B2, E}; {C, D1, D2}; and {C, D1, B2, E}.

[1] These correspond to time, energy, etc.

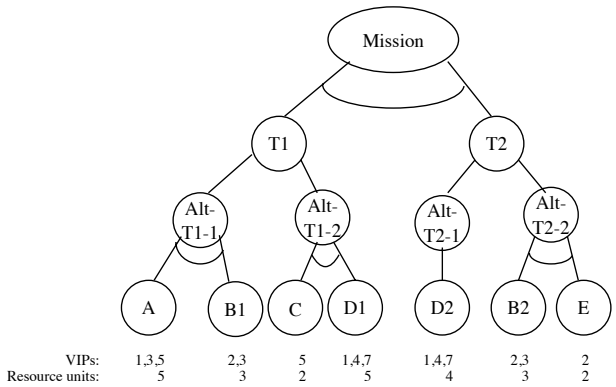

Fig. 1. A task decomposition tree for a mission

To build the constraint graph as described above, we must add all of the capabilities which constitute a decomposition to the constraint graph. Capabilities are selected using textures from CHS so that the resulting constraint graph can be solved efficiently. In particular, we focus on textures that increase the likelihood that a solution can be found without backtracking.

Our algorithm starts with an empty constraint graph and the initial task decomposition tree; this is the initial state. A new state is generated by selecting a leaf node or ANDed leaf nodes to add to the graph. Each selection represents a partial commitment to one particular task decomposition. Selections are made by applying two recursive algorithms (Fig. 2), *select-alternative* at OR nodes and *select-subtask* at AND nodes, starting at the root of the tree and ending when a selection is made at the leaf level.

```
select-alternative(node):
  if node is leaf then
    return(size(domain),node);
  else
    call select-alternative on each child;
    if node is OR then
      return(max value, child with max value);
    else  ;; it's an AND node
      return(min value, child with min value);
    fi;
  fi;
end;
```

```
select-subtask(node):
  if node is leaf then
    return(size(domain),node);
  else
    call select-subtask on each child;
    if node is AND then
      return(min value, child with min value);
    else  ;; it's an OR node
      return(sum of values of children);
    fi;
  fi;
end;
```

Fig. 2. Algorithms used to select capabilities

Following the *value goodness* texture [2], in *select-alternative* we choose the

alternative that can be satisfied by the greatest number of values. This gives the constraint algorithm the most flexibility. Since the VIPs that can perform the task are the potential values for the variables, they provide the basis for the selection. The values are propagated up the tree from the leaves so that the interdependence of the ANDed siblings can be taken into account.

Following the *constraint reliance* texture [2] in *select-subtask* 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. This ensures that these important constraints will be in the constraint graph, and will be used to construct partial solutions, early in problem solving.

When a node is selected, its associated capabilities are added to the constraint graph. The task decomposition tree is then pruned to remove the selected nodes and other nodes which cannot be part of the same task decomposition. This includes pruning alternatives to ancestors of the node.

3 Example

Returning to the task decomposition in Fig. 1, we will now consider how the technique described above would select a task assignment. This tree is reproduced in Fig. 3 showing the domain of each variable and resource units required. Suppose that there is the following situation with respect to VIPs: VIP1 has capabilities A and D, with 6 resource units total; VIP2 has B and E, with 3 units; VIP3 has A and B, with 5 units; VIP4 has D, with 5 units; VIP5 has A and C, with 5 units; VIP6 has no capabilities needed for this task; and VIP7 has D, with 5 units.

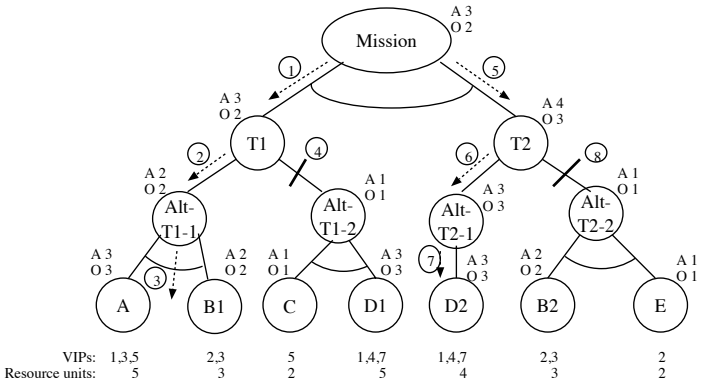

Fig. 3. Annotated task decomposition tree

The numbers beside the nodes labeled with "A" and "O" show the and- and or-ratings of the node, respectively, as computed by *select-alternative* and *select-*

subtask.[2] The progress of the capability selection algorithm is shown on the tree. Arrows indicate selections of children and slashes indicate branches that have been pruned. These marks are numbered and the corresponding number appears in parentheses after the text describing that step.

Starting with the root of the marked tree, *select-subtask* chooses the subtask to consider first. It selects the subtask with the lowest and-rating, T1 (1). From there, *select-alternative* will select the child of T1 with the highest or-rating, Alt-T1-1 (2). This alternative has only ANDed children, so they become the first variables added to the constraint graph (3). The first part of Fig. 4 shows the constraint graph after A and B1 are added. Double circles show the n-ary resource constraints, labeled with the VIP concerned and resource limit. Constraints are checked at this point, but since no commitment has yet been made to a value for either variable, no values are eliminated from either domain. The tree is now pruned (4) to eliminate non-chosen OR branches in the ancestors. At this point, a new search state has been created.

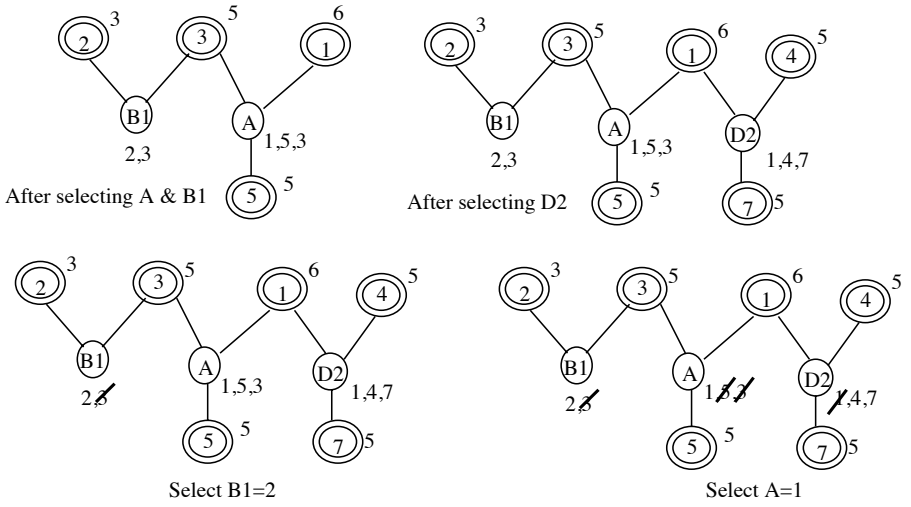

Fig. 4. Evolution of the constraint graph in succeeding states

From this state, the algorithm returns to the top of the tree and selects the only remaining subtask, T2 (5). *Select-alternative* now selects Alt-T2-1, and D2 is added to the constraint graph, as shown in the upper right of Fig. 4. The constraint graph is consistent, the tree is pruned (7), and another search state is created.

[2] An optimization is done in the program to record the markings at the nodes so that *select-subtask/-alternative* do no unnecessary work.

At this point, there are no more subtasks at the top level, so a complete possible configuration is represented in the constraint graph. The normal CHS algorithm now takes over. As shown in the bottom of Fig. 4, CHS selects the variable (B2) with the fewest values in its domain, then assigns to it the value (VIP2) that appears in the fewest domains. Constraint propagation does not prune any other domains. From this state, CHS assigns VIP1 to A. Constraint propagation prunes D2's domain in this case. Our heuristics evaluate all remaining values for D2 as equal, so VIP4 is arbitrarily chosen for the value of D2. This yields a goal state, A=VIP1, B1=VIP2, D2=VIP4.

Although space does not permit us to include backtracking in our example, backtracking could occur under two circumstances. The system can backtrack during constraint satisfaction to try different choices for assigning values to VIPs. If the constraint graph cannot be satisfied, the system can backtrack to states created during capability selection to try a different task decomposition.

4 Related Work

Several techniques from distributed artificial intelligence can be used to do task assignment (e.g., [4], [5]). However, most of them rely on local decisions, often made by local negotiation between agents, to produce global solutions. This can lead to sub-optimal solutions or even to no solution at all, since there is no global perspective and no possibility to coerce an agent into doing a locally-unacceptable, but globally-necessary, task. It is difficult to see how a technique such as the Contract Net Protocol [4] or Partial Global Planning [5] can handle selecting between alternative configurations; this requires a global notion of what the possible configurations are. The local contracts or agreements in those systems would also over-commit to alternatives that are tentative. Some approaches also require sophisticated agents [5], which we cannot guarantee in our application.

Our technique is also related to applications of CHS in other domains. At first glance, job-shop scheduling [6] appears to be the most closely-related application because it is concerned with scheduling resources to perform tasks. However, it differs from our domain because there are no alternative decompositions for performing the task.

Other applications of CHS have considered alternative constraint graphs. Of these, Wright [7] and CORAL [8] are most closely related to our work. Wright uses texture-based heuristics to select alternative layouts for spatial planning problems. However, Wright relies on representing possible alternatives in the CHS formalism for constraint graphs instead of in a standard task decomposition tree. The CHS formalism is difficult to use in our domain and is less than ideal for any application in which the task decomposition tree will be provided by some other planner.

CORAL works in the domain of configuring and allocating available components to assist inventory managers. It represents possible configurations in a standard task decomposition tree, selects a feasible configuration, and assigns

resources from its inventory to the configuration. CORAL is very similar to our work. First, selections are made from the task decomposition tree, propagating constraints when relevant. After a complete configuration is selected, assignments are made to actual components.

CORAL differs from our work in two significant ways. First, CORAL does not take advantage of texture-based heuristics when selecting its configuration. We have found that using such heuristics increases the overall efficiency for task assignment [9]. Second, CORAL selects a particular type of part that will be used in a configuration from the task decomposition tree. In our domain, it is analogous to selecting the VIP that will be used to perform a task as part of choosing the alternative. In other words, this would be analogous to binding a variable to a value before adding the variable to the constraint graph. The technique works in CORAL's domain because there are usually enough parts available to make any configuration. However, in our application, texture heuristics must be used to help select VIPs for task assignment because not all assignments will be possible. Consequently, the resource limitations of the VIPs must be taken into account when a VIP is assigned to a task.

We have implemented a version of CORAL, adapting it to our application by assigning a VIP when a capability is selected from the task decomposition tree. This version of CORAL was used to collect the data described below. Our early experiements with CORAL suggest that our technique reduces backtracking for our application.

5 Conclusion

Our system was used to produce task assignments for 100 randomly-generated, solvable task decomposition trees. Examples in this set had a mean of 2574 configurations (possible task decompositions) per mission. The median time of solution was 50 ms.[3] The median number of backtracks per runs in which the system backtracked was 1. Our version of CORAL was run on the same set of task decomposition trees as above. It had a median run time of 70 ms and backtracked a median of 1.5 times per run with backtracks.

Our technique will be used in our AOSN to make task assignments. In the future, we plan to run additional experiments to test our heuristics for selecting task decompositions. We also plan to integrate our operator for selecting capabilities with the other CHS operators.

References

1. Curtin, T.B., Bellingham, J.G., Catipovic, J., Webb, D.: Autonomous oceanographic sampling networks. Oceanography. **6** (1993)

[3] Since there was an outlier that skewed the mean, we are using median values. The constraint graph produced for the outlier had a high number of backtracks. A mechanism for identifying constraint graphs which will have high numbers of backtracks and selecting values early on to avoid backtracking is discussed in [9].

2. Fox, M.S., Sadeh, N., Bayken, C.: Constrained heuristic search. In: Eleventh International Joint Conference on Artificial Intelligence. (1989) 309 – 315
3. Turner, R.M., Turner, E.H.: Organization and reorganization of autonomous oceanographic sampling networks. In: International Conference on Robotics and Autonmation. (to appear)
4. Smith, R.G.: The contract net protocol: High-level communication and control in a distributed problem solver. IEEE Transactions on Computers. **12** (1980) 1104 – 1113
5. Durfee, E.H., Lesser V.R.: Negotiating task decomposition and allocation using partial global planning. In: Gasser, L., Huhns, M. N. (eds.): Distributed Artificial Intelligence. Morgan Kaufmann Publishers, Inc., San Mateo, CA (1989) 229 – 243
6. Sadeh, N., Fox, M.S.: Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. Artificial Intelligence. **86** (1996) 1 – 41
7. Baykan, C.A., Fox, M.S.: Constraint techniques for spatial planning. In: ten Hagen, P.J. W., Veerkamp, P. J. (eds.): Intelligent CAD Systems III: Practical Experience and Evaluation. Springer-Verlag, New York (1991) 187 – 204
8. Sathi, N., Fox, M.S., Goyal, R., Kott, A.S.: Resource configuration and allocation: A case study of constrained heuristic search. IEEE Expert. **7** (1992) 26 – 35
9. Turner, E.H., Turner, R.M.: Selecting task decompositions for constrained heuristic search (in preparation)

This article was processed using the LaTeX macro package with LLNCS style