Information Sharing among Heterogeneous Reusable Agents in Cooperative Distributed Search

Susan E. Lander, Victor R. Lesser, and M. V. Nagendra Prasad
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{lander,lesser,nagendra}@cs.umass.edu

Abstract

Information sharing among heterogeneous reusable agents in cooperative distributed search systems can greatly affect the quality of solutions and the runtime efficiency of the system. In this paper, we first give a formal description of shareable information in systems where agents have private knowledge and databases and where agents are specifically intended to be reusable. We then present experiments, run within a mechanical design system for steam condensers, that substantiate expected performance improvements related to information sharing and assimilation. Finally, we discuss the practical benefits and limitations of information sharing in application systems comprising heterogeneous reusable agents. Issues of pragmatic interest include determining what types of information can realistically be shared and when the costs of sharing outweigh the benefits.

1 Introduction

Search has always been a central issue in Artificial Intelligence. Search systems are historically described as comprising three components: a *database* or *state space* describing the current state of the search, a set of *operators* used to manipulate the database, and a *control strategy* used for deciding what to do next, specifically, deciding what operator to apply and where to apply it [1]. When all operators reside in a single program or logical entity and have access to a central store of knowledge and databases, the search is centralized. In this paper, we are concerned with the problem of distributed search as described in [10]:

> A distributed search involves partitioning the state space and its associated operators and control regime so that multiple processing elements can simultaneously perform local searches on different parts of the state space; the (intermediate) results of the local searches are shared in some form so that the desired answer is produced in a timely manner.

When information is represented in a modular and logically separable way, the information is potentially *reusable* [11]. The computational equivalent to teams of human specialists is the reusable-agent system, a multiagent system for which the agents can be dynamically selected from an existing

This work was supported by ARPA contract N00014-92-J-1698, Office of Naval Research contract N00014-92-J-1450, and NSF contract CDA 8922572. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

library and integrated with minimal customized implementation. With reusable agents, diverse types of information can be applied in situations that were not explicitly anticipated at agent-development time. The partitioning of the state space in this type of system is induced by the *a priori* division of expertise of agents in the agent set.

Throughout the paper, we will augment the presentation of concepts with examples from a seven-agent system, **STEAM**, that performs parametric design of steam condensers. The agents in **STEAM** each take responsibility for either: 1) designing some component of a steam condenser; or 2) critiquing some aspect of the condenser. The agent set in **STEAM**, $\mathcal{A}$, is

$$\mathcal{A} = \{\texttt{pump-designer}, \texttt{heat-exchanger-designer}, \texttt{motor-designer}, \texttt{platform-designer}, \texttt{vbelt-designer}, \texttt{shaft-designer}, \texttt{system-frequency-critic}\}$$

Figure 1 shows the general form of a steam condenser.

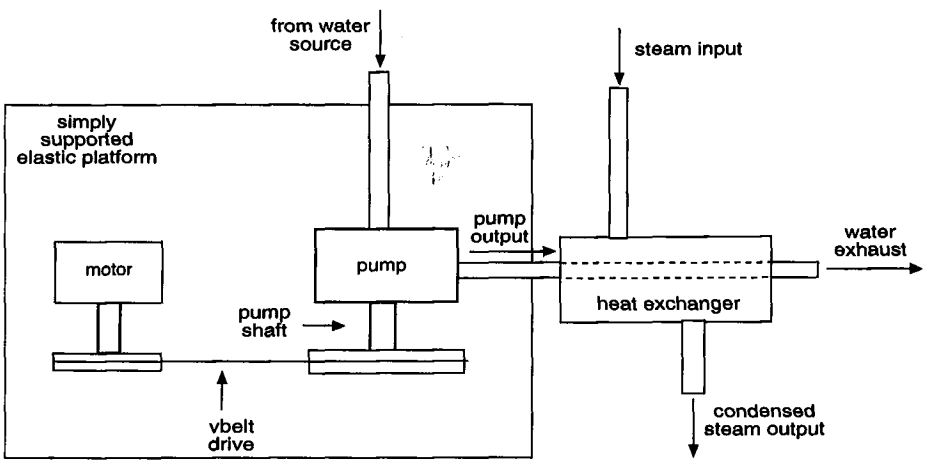

Figure 1: *A Steam Condenser*

The agents in this system are *globally cooperative*, meaning that there is some global measure of system performance that overrides any local measures. This is generally true for design problems: there is some measure of the quality of a design that is distinct from the quality of any subcomponents of that design. An important aspect of globally cooperative systems is that it is not useful for agents to attempt to maximize their local payoffs for solutions by withholding information from other agents. The overriding goal of the system is to maximize the global, rather than local, payoff for solutions. In this situation, sharing information is not restricted by selfish or adversarial motives of agents as in some multiagent domains [15, 16].

Another example of a globally cooperative multiagent system is found in [3], which describes the Asynchronous Team (A-Team) approach to solving a class of problems where multiple partially satisfactory algorithms exist but no completely satisfactory algorithm is known (such as the Traveling Salesman Problem). In the A-Team approach, each agent represents one of the known algorithms and the goal is to cooperate in such a way that the agent set produces better results as an organization than any one agent would produce alone. In this work, the only information shared among agents is in the form of partial solutions and the emphasis of the research is on how intermingling of the control flow of the agent organization improves performance.

Our goal in this paper is to show that globally cooperative agents involved in distributed search can improve their joint performance by assimilating information from the other agents and using this information to refine their local views of the state space. The development of operators and control

strategies for distributed search is an important issue and is addressed in [9]. Current research on languages, ontologies, and protocols for agent interaction, such as KQML [4], complements this work but focuses on the mechanics of communication rather than on the impact of the communicated information on problem solving.

In Section 2 we give a formal description of how information sharing among heterogeneous agents in globally cooperative domains affects the local view of each agent. The next section, Section 3, presents experimental results from **STEAM** that substantiate our hypothesis that information sharing and assimilation can improve system performance, both in terms of solution quality and in terms of problem-solving efficiency. Section 4 discusses what costs are involved in information sharing and what the practical limitations of the technology are from an application-system perspective. We conclude with a summary of the observed results and some speculation as to the significance of these results within the **STEAM** system and within the more general realm of multiagent systems.

2 Shared Spaces

When talking about the shared spaces of agents, we distinguish between the *local* space of an agent and the *composite* space of the system. A local space is one that is private to an agent, the composite space is one that is shared by all agents.[1] The local *solution space* of an agent is defined by the parameters that are assigned values by an agent in its local solutions. The local *search space* is defined by the parameters the agent uses to constrain its local search. Each agent starts out with a completely local view of search and solution spaces. However, this local view is unlikely to be effective in finding solutions that are mutually agreeable to all agents (solutions in the composite space). A primary goal of communication among agents, therefore, is for each agent to end up perceiving the closest approximation possible to the actual composite search and solution spaces. In nontrivial cases, it is unlikely that a complete and correct global view can be achieved at every agent. However, to the extent that its local view approaches the global view, an agent is likely to be more effective at proposing solutions that will be mutually acceptable.

We will use examples from the **STEAM** system to illustrate the concepts being discussed. Figure 2 shows a simplified version of the solution space of **pump-agent**. This figure is simplified both in the number of parameters and the specification of the parameters' domains. The set of parameters in

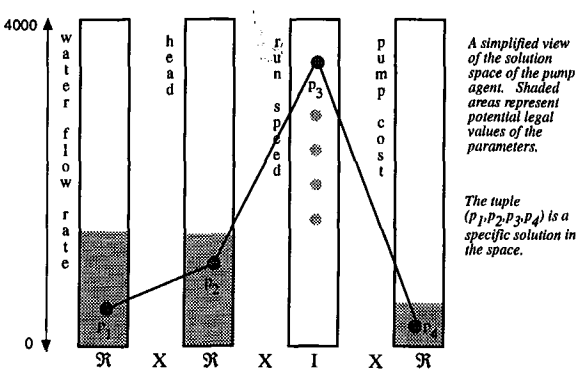

Figure 2: *The Local Solution Space Of Pump-Agent from the* **STEAM** *System*

the solution space of an agent α will be represented as $\mathcal{P}^{\alpha}$, the *parameter set* of α. The parameter set of **pump-agent**, as shown in Figure 2, is {water-flow-rate,head,run-speed,pump-cost}.

[1]Spaces may also be shared by some subset of agents, but not the entire agent set [8]. These *common* spaces are outside the scope of this paper, however, and we will not discuss them further here.

The set of legal values for a parameter θ at agent α is its *parameter space*, $\mathcal{V}_\theta^\alpha$. To illustrate, the parameter space of **run-speed** from Figure 2 is the set of integers {1200,1800,2400, 3000,3600}. The legal values of parameters can be of different types such as integers, reals, numeric intervals of the form {*(min,max), min,max* $\in \mathcal{R}$}, or discrete labels such as {*model-1, model-2,...,model-n*}. A solution in the solution space of α is a tuple $s_j^\alpha = (p_1, p_2, \ldots, p_n)$ such that $p_x \in \mathcal{V}_x^\alpha$ and such that all constraining relationships on and between the $p_x \in s_j^\alpha$ are satisfied. A parameter space may be constrained by explicit constraints on solutions such as (**run-speed** $\geq$ 1200) or through implicit requirements that are embedded in the functions an agent uses to search for solutions.[2] As a trivial example of an implicit constraint, consider the following loop in pseudo-code:

```
head :== 0;
DO  water-flow-rate = 0 to 500
  new-head :== calculate-head water-flow-rate;
  head :== select-best new-head head;
END LOOP;
```

An agent using this code implicitly constrains the parameter space of **water-flow-rate** to be the set of integers from 0 to 500, although it may not declaratively represent this anywhere. In reality, functions tend to be more complex and the implicit constraints more difficult to discern. In the above example, the value of **head** is tacitly constrained by the implicit constraint on **water-flow-rate**. However, the effect of the implicit head constraint on the values that can be assigned to that parameter may not be determinable without in-depth expertise.

The existence of *implicit* constraints must be expected in the general case of expert agents because much expert knowledge is captured in application systems in a procedural format. If this were not true, all agents could essentially be reduced to a single inference engine being applied over different sets of constraints. Implicit constraints cannot normally be shared since they are an integral part of the agent's expertise and cannot be easily extricated.[3] Unshareable constraints strongly affect properties of the agent sets in which they are embedded. For example, in [6], Khedro and Genesereth present a search model in which agents provably converge on a globally satisfactory design if one exists. However, the property of convergence can only be guaranteed if all constraining information can be explicitly exchanged. When implicit constraints are added, this desirable property no longer holds.

Explicit (declaratively represented) constraints are those that can be shared and, as will be discussed in Section 3, this sharing can greatly enhance the effectiveness and coherence of the agent set. In **STEAM**, explicit constraints are limited to simple boundary constraints of the form (**water-flow-rate** $<$ 800) and (**water-flow-rate** $\geq$ 0) that specify minimum or maximum values for a parameter. In Section 3, we will describe experiments in which we measure the effect of exchanging constraints of this form. To explicitly include these constraints in the definition of a solution space, we use the following notation: let c_j^α be a declaratively represented local solution requirement of agent α in the set of all explicit solution requirements of α, $\mathcal{C}^\alpha$. Then, let the notation $\{c_j^\alpha : s_k^\alpha\}$ mean that c_j^α is satisfied with respect to a particular solution, s_k^α. For example if c_1^α is $(p_1 \leq 10)$ and $s_1^\alpha = (8, 5, 3, 7)$, then $\{c_1^\alpha : s_1^\alpha\}$. If c_j^α is neutral with respect to s_k^α (it does not constrain any parameters in s_k^α), it is considered to be satisfied.

[2] We use the terms *solution requirement* and *constraint* interchangeably. The terms are not intended to imply any specific representation.

[3] It is possible that some agents may be able to share either code or some form of abstracted explanation of implicit constraints. However, this requires specialized capabilities on the part of both the sending and receiving agents. Although it is possible to support these capabilities in specific situations, generalized code exchange and assimilation among heterogeneous reusable agents is not a realistic option.

Using this notation, the shareable solution space of agent α can be defined by specifying the parameter set of α, $\mathcal{P}^\alpha$ and the set of explicit solution requirements over those parameters, $\mathcal{C}^\alpha$. This shareable solution space is an approximation of the actual solution space since it does not represent any implicit solution requirements that are embedded in the agent. We formally describe the shareable local solution space of agent α as follows: $\delta^\alpha = \{(p_1, p_2, \ldots, p_n) \mid \forall c_j \in \mathcal{C}^\alpha, \{c_j : (p_1, p_2, \ldots, p_n)\}$. In nontrivial cases, δ^α will be a superset of the valid solutions of agent α since it does not take implicit constraints into account.

The Composite Solution Space: Given a set of agents, $\mathcal{A}$, and a problem that they are cooperating to solve, the desired solution must derive its parameter values from the local solution spaces of the agents. However, as seen in Figure 3, the parameters of the global solution space are not necessarily the union of all parameters in the local solution spaces . In this figure, the solution space of agent

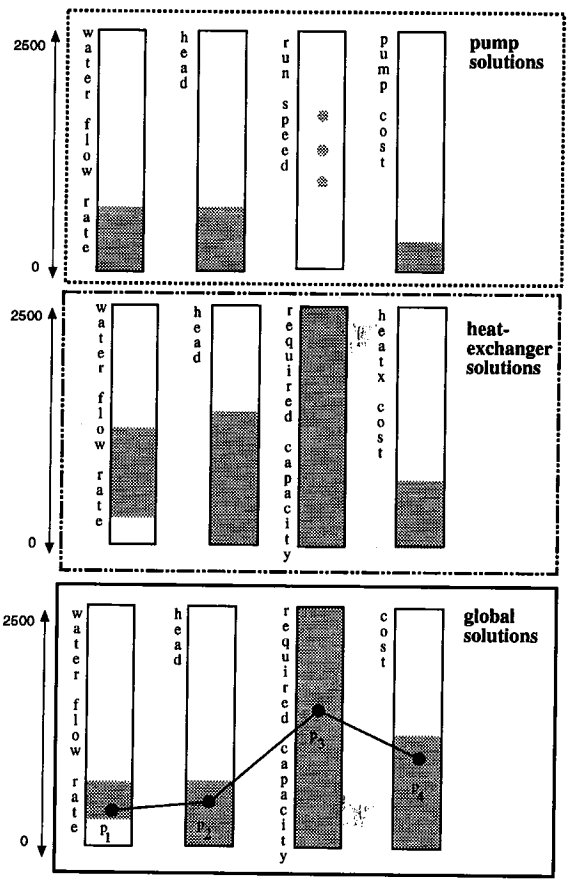

Figure 3: *Constructing a Global Solution from the Local Solutions of Agents*

p (the pump agent) contains the parameters *water-flow-rate*, *head*, *run-speed*, and *pump-cost*. The solution space of agent h (the heat-exchanger agent) contains the parameters *water-flow-rate*, *head*, *required-capacity*, and *heatx-cost*. Ignoring for the moment the problems associated with determining the semantic equivalence of local parameter spaces, we find in Figure 3 that the parameters *water-flow-rate* and *head* are common to both agents while *run-speed*, *pump-cost*, *required-capacity*, and *heatx-cost* represent parameters unique to individual agents. The global solution space shown in Figure 3 contains the shared parameters, *water-flow-rate* and *head*, the parameter *required-capacity* from agent h, and also a unique parameter, *cost*. *Cost* is not local to either agent p or agent h, but represents a transformation on local parameters of those agents, i.e., the sum of *pump-cost* and *heatx-cost*. To summarize, each parameter in the global solution space is local to either agent p or

agent h, local to both agent p and agent h, or is a unique parameter whose value can be derived from parameters local to agent p and/or agent h.

In Figure 3, notice that the constrained set of values (the shaded areas) of the shared parameters, *water-flow-rate* and *head*, are not identical for the two agents. If we are looking only at constraint-satisfaction problems, problems in which all constraints must be satisfied or no solution can be found, the constrained global parameter space is the intersection of the constrained local parameter spaces. When the intersection is empty, no solution exists. Consider a global parameter such as *water-flow-rate* from Figure 3 which is the intersection of the local *water-flow-rate* parameters of the two agents p and h. To differentiate the spaces, we will denote the global parameter as w^G. If w^G is empty, no solution exists that will be mutually acceptable to agents p and h. If w^G is not empty, there are two possibilities: 1) a mutually acceptable solution, $s_x^G = (p_1, p_2, p_3, p_4)$, exists in the global space with $p_1 \in w^G$; or 2) no solution exists in the global space because there are implicit constraints at one or both agents that exclude any solutions in the explicitly constrained space. Because of the possibility that implicit constraints exist, it is impossible to tell by looking at w^G whether or not a mutually acceptable solution exists. If one exists, however, it will be in that space.

In constraint-optimization problems, not all constraints must be satisfied in a solution. Instead an attempt is made to satisfy constraints to the fullest extent possible. Constraints may have differing amounts of flexibility: some may be *hard*, meaning that they must be satisfied in any legal solution, while others may be *soft*, meaning that they can be relaxed if necessary. Soft constraints again can have different degrees of flexibility: some can be "softer" than others. In these types of problems, composite solutions must lie within the intersection of the local parameter spaces under the set of hard constraints, but not necessarily under all soft constraints. Issues of which constraints should be relaxed and when are beyond the scope of this paper. The order in which constraints are relaxed can strongly affect system performance and solution quality. Several researchers have looked at developing heuristics for constraint ordering [5, 12]. In distributed search, not only is constraint ordering at a single agent important, but power and ordering relationships among agents must be considered.

In this section, we have looked at how information sharing can be used to refine local perspectives to more closely approximate the composite perspective required for effective solution generation. In the next section, these techniques are empirically investigated through information-sharing experiments conducted in the **STEAM** system.

3 Empirical Analysis of Information Assimilation

In this section we empirically demonstrate the cost effectiveness of sharing potentially useful information with other agents during distributed search. Notice that with reusable agents, the usefulness of shareable information cannot be determined at agent-development time since it is dependent on capabilities and interests of other agents that may eventually be integrated into a joint agent set. The experiments described here were run in the **STEAM** system with seven active agents. In order for an agent to use information received from an external source to guide its local processing (i.e., *learn* about other agents' requirements for solutions), the agent must be able to receive constraining information sent from other agents, translate that information into a locally usable form, and store the translated information into a local knowledge base, indexed by its source. We call this process *information assimilation.* The assimilated information can then be retrieved by the agent to guide its search for local solutions. We have developed mechanisms that extend or replace the traditional retrieval capability of an agent to extract relevant constraining information from its knowledge base. These extensions were developed specifically to enable reusable agents to handle potentially conflicting information that has been received from external sources since there is no guarantee that

shared external information will be consistent with internal information. The goal of the retrieval process is to find the most restrictive, but non-conflicting, set of known solution requirements for the current problem using both local and assimilated information. Intelligent conflict-resolution capabilities are required to select which information should be applied when inconsistencies are detected. Conflict resolution is an important and encompassing problem that has been extensively studied [7, 8, 13, 14, 16].

The information shared in these experiments was limited to simple boundary constraints: single-clause constraints of the form $(x < n)$, $(x <= n)$, $(x > n)$, or $(x >= n)$ where x is a shared numeric parameter and n is some numeric value. For example, pump-agent specifies a hard constraint (*water-flow-rate* <= 404.34).

In the experiments reported below, there are two categories of experimental trials: *non-assimilation trials* and *assimilation trials*. For the assimilation trials, three agents instantiate the capabilities required for information assimilation: the pump, motor, and heat-exchanger agents. The other agents do not attempt to assimilate information. The reasons for this are explained in Section 4.1. In the non-assimilation trials, the assimilation capabilities are not active at any agent. The system was run on each of 100 different feasible problem specifications, once with active assimilation capabilities and once without.

3.1 Measuring System Performance

We expected to see that the extra costs associated with assimilating information would be balanced, in the majority of cases, by improvements in performance. To test this hypothesis, we first had to determine how to measure performance.

There are two measures of system performance in the **STEAM** system: run time and solution quality. In the **STEAM** domain, solution quality is more important—we don't mind spending extra time in the design process (within reason) to get a high-quality design. In other domains, however, this tradeoff is not always the best one. Also in deference to the design domain, in order to give the user final control over the design process, the termination policy used was to process until three acceptable solutions were completed. Solutions were deemed acceptable if they were locally acceptable to all agents and if the solution cost was below a user-defined threshold. After the completion of three acceptable solutions, no new solutions were initiated, but existing acceptable partial solutions were completed before halting. When the system halted, all acceptable solutions were ranked and presented to the user.

3.2 Solution Quality

We compared the results of running the system when agents assimilated constraining information and when they did not. In **STEAM**, solution quality is determined by the monetary cost of each solution: the minimum-cost acceptable design is considered the most highly rated. The results of these experiments are graphically summarized in Figure 4. In this figure, the results are sorted into ascending order based on cost in the assimilation trial.

For the 100 problem specifications tested, the mean cost in the assimilation trials was \$8504.77, in the non-assimilation trials, it was \$9020.43. The mean cost improvement with assimilation operators enabled was 5.72%, meaning that the monetary cost of the most highly rated solution in an assimilation trial was 5.72% lower on average than that in the associated non-assimilation trial under the identical problem specification. We had hypothesized that enabling assimilation would lower the cost of a design (thereby improving solution quality) and the experimental results appeared to support this hypothesis. To statistically confirm this result, we applied a paired difference t-test. In this type of test, the results from two matched trials are compared—in our case non-assimilation trials

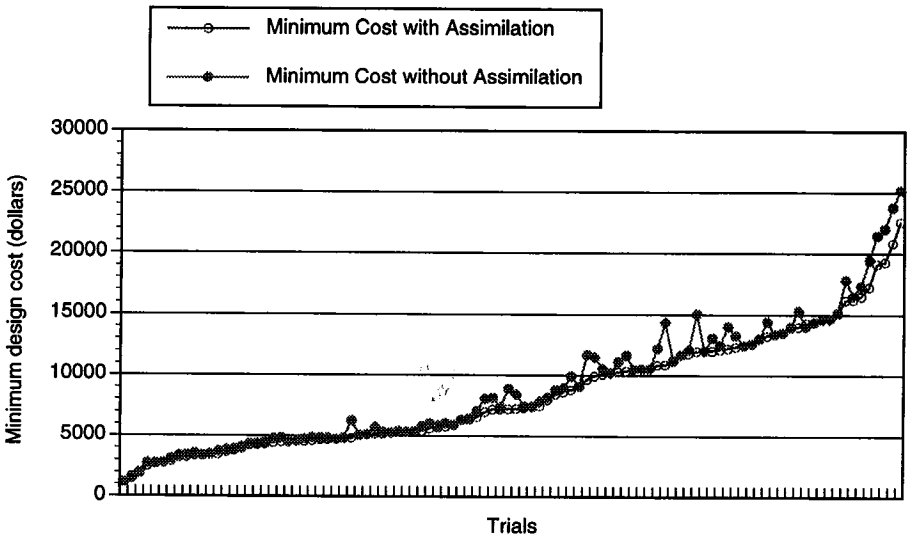

Figure 4: *Solution Quality Results in Assimilation Experiments*

are compared to assimilation trials performed under the same problem specification. For each paired trial the difference between the design costs is taken, and then the mean of the differences across all trials is computed. The null hypothesis in this case is $H_o : \mu_D = 0$ (the population mean of the differences is 0), meaning that the results of the two trials are not significantly different. The alternative hypothesis is $H_a : \mu_D > 0$ (the population mean of the non-assimilation trial results minus the assimilation trial results is greater than 0), meaning that the cost of designs in the non-assimilation trials are higher than those in the assimilation trials. Applying the paired t-test results in a t-score of 6.455, which allows us to reject the null hypothesis with a confidence of more than 99%. We can thus say with a high level of confidence that when **STEAM** agents apply assimilation capabilities, the average quality of solutions improves.

An inherent characteristic of the **STEAM** domain is that good solutions are easy to find under many problem specifications (the solution space is dense). We believe that there is a significant *floor effect* in the domain, meaning that minimum-cost designs are easy enough to find even in the non-assimilation trials that it is difficult to dramatically improve solution quality. However, the ability to consistently lower design costs approximately 5.72% by sharing simple boundary constraints is compelling evidence that information sharing and assimilation is an important technique for improving solution quality in multi-agent systems. Furthermore, if it is the case that a floor effect is influencing our results, larger improvements could be expected in some domains.

3.3 Run Time

Run time was directly measured in these experiments as the elapsed real time from the invocation of the system until termination of the system.[4] The average runtime with assimilation is 121.98 seconds, without assimilation the average runtime is 132.67 seconds. The assimilation runtimes are, on average, 8.06% lower than the non-assimilation runtimes. However, direct comparison of the run times of assimilation and non-assimilation trials is somewhat misleading. As discussed earlier, the termination policy used for the **STEAM** system is that when 3 acceptable solutions are found, the

[4]These experiments were run on a TI Explorer. Incremental garbage collection was turned off during the runs. However, the recorded time includes time spent on process and memory management tasks such as paging, etc. Therefore, recorded times varied slightly across identical runs.

system enters a termination phase during which all acceptable partial solutions are completed. The intent is to finish potentially good solutions that are initiated later in problem solving. This policy is appropriate for the **STEAM** domain since solution quality and user participation are higher priorities than run time. If it is assumed that agents are assimilating information, agents are more likely to initiate good solutions as information is incrementally accumulated, i.e, solutions get better as problem solving progresses. One result of this termination policy is that there are likely to be more acceptable solutions produced per run in the assimilation trials than in the non-assimilation trials and this is indeed the case. However, this leads to a bias in which direct runtime measures favor non-assimilation trials. In those trials, the system doesn't complete as many solutions as it does in the matching assimilation trial because it isn't able to focus on mutually acceptable regions of the composite search space.

To make runtime comparisons more meaningful, we divided the run time of each trial by the number of solutions completed during that trial, resulting in a *runtime-per-solution* measure. The results obtained using this method are graphed in Figure 5.

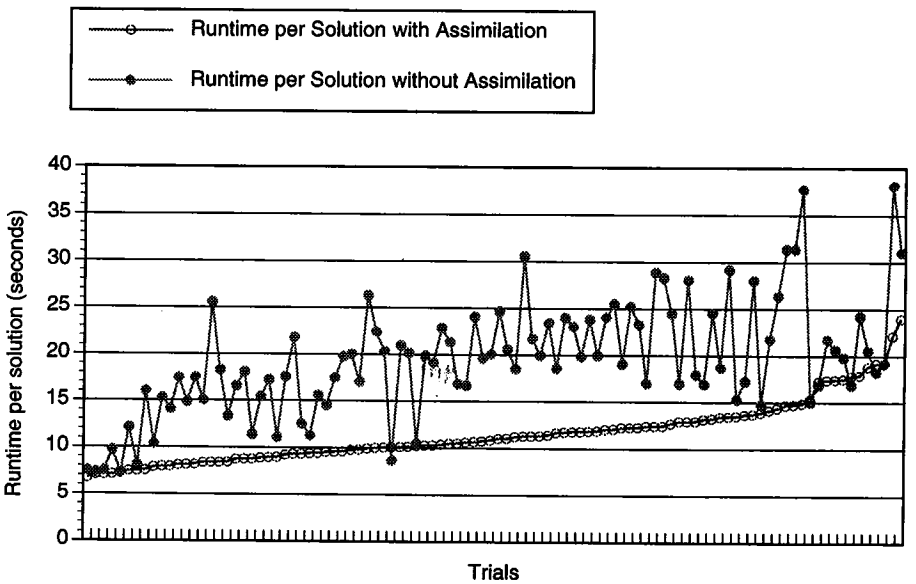

Figure 5: *Runtimes-per-Solution Results in Assimilation Experiments*

The *runtime-per-solution* observations in the assimilation and non-assimilation trials were averaged over the 100 experiment sets for comparison. The average runtime per solution in the assimilation trials was 11.58 seconds and in the non-assimilation trials it was 19.50 seconds. The average percent improvement achieved by the assimilation trials over the non-assimilation trials in runtime-per-solution was 40.62%.

The results presented in this section demonstrate that information sharing can positively affect both solution quality and system performance in a heterogeneous reusable-agent system. However, there are costs associated with information sharing and, in fact, the more sophisticated information sharing becomes, the higher the costs are likely to become. In the following section, we discuss what costs are involved in information sharing, particularly in the case of sharing among heterogeneous reusable agents.

4 Information-Sharing Costs

Sharing information has five primary costs:

1. constraint generation costs: generating restricting information that can be transmitted to other agents to guide their local searches;
2. information determination costs: determining which constraints to transmit to other agents at a given point in problem solving;
3. transmission costs: the actual costs associated with the physical transfer of messages among agents;
4. translation costs: translating constraining information from the local language to a sharable format at the sending agent and then translating the information from a sharable format into a local language at the receiving agent;
5. local management costs: determining the applicability of the information at the receiving agent (sorting, filtering, detecting conflicts and locally resolving those conflicts) and managing the greater volume of information that results from accumulating received information (storage and retrieval costs).

In the timing analysis described later, we assume that transmission costs (those associated with the physical transfer of information from one agent to another) are minimal. This is consistent with the current implementation of **STEAM** in which all agents reside on the same machine and run in the same process, but this assumption cannot be extended to the general case. However, in these experiments, we do not track transmission costs separately, but do break down the other information-sharing costs.

Constraint Generation: The ability of an agent to generate restricting information to send to other agents is highly domain- and representation-dependent. There are three primary types of information that can be sent: 1) constraints that are completely independent of the specific problem being addressed (*independent* constraints); 2) constraints that are dependent only on the problem specification without regard to any particular solution (*problem-dependent* constraints); and 3) constraints that are dependent on existing instantiated parameters for a particular solution (*solution-dependent* constraints). These different categories are explained in more detail in [8].

In the timing studies reported here, we investigated primarily the use of problem-dependent constraints. Costs associated with independent constraints are not considered to be part of the normal cost of developing a solution because these constraints can be generated in a one-shot preprocessing procedure. In some domains, it may be possible to exploit solution-dependent constraints. For example, in their work on multistage negotiation, Conry et. al. have developed a formalism that explicitly represents the interactions of solution-dependent constraints among subplans and uses these constraints to either derive a solution or determine that no solution exists [2]. However, solution-dependent constraint generation and manipulation techniques are difficult in this domain because of the combinatorics of potential value interactions.

Information Determination: An agent must decide what information to transmit. In **STEAM**, agents transmit information directly in response to conflict situations rather than transmitting information that is anticipated to be potentially useful. Therefore, only constraints that are in direct conflict with an existing solution are transmitted. The costs of retrieving potentially conflicting constraints and checking each constraint to see if it conflicts with the existing solution are reported in the information-determination measure.

Constraint Translation: In the general case of heterogeneous reusable-agent systems, local knowledge can be represented at an agent in any form that is appropriate for that agent but some mechanism must be provided to ensure that agents are able to understand each other. When translation is necessary, the cost can vary greatly depending on exactly what is entailed. Some agents may share a language and have no translation costs, others may translate using simple syntactic procedures, and others may require complex semantic translation. In STEAM, the local representation of an agent's knowledge is unrestricted, but in order for information to be shared, it must be translated into a globally specified language. All agents use the same simple syntactic procedures for translation between local and global formats. Translation costs, therefore, do exist but are relatively small.

Local Management: Conflict between local and assimilated information is one factor that potentially mitigates the benefits of information sharing: what happens when an agent receives information that contradicts something it already knows? With logically heterogeneous agents, it must be assumed that conflict will occur. In the STEAM system, the costs of storing information and managing conflicts between inconsistent local and external information are categorized as local management. Other local management costs include costs that accrue from the greater volume of information that must be stored and retrieved due to assimilated information.

4.1 Observed Information-Sharing Costs in STEAM

In these experiments, the costs attributed to sharing information are broken down into the categories: 1) constraint generation (for problem-dependent constraints); 2) transmission determination; 3) constraint translation; and 4) local management, as described earlier. The observed costs for each of these categories over the 100 problem specifications are summarized in Figure 6.

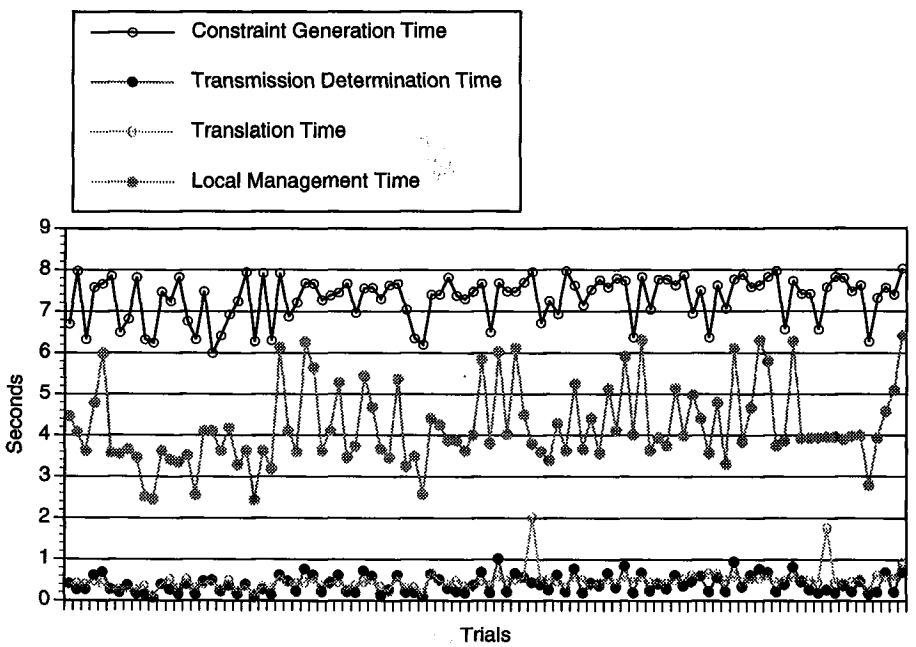

Figure 6: *Information-Sharing Costs in* STEAM

The approximate average time spent in constraint generation per problem is 7.3 seconds, constraint transmission is .4 seconds, translation is .5 seconds and local management is 4.2 seconds, for an average total time for information assimilation of approximately 12.4 seconds per run. The average total percentage of time spent in information sharing in these trials is 10.17%. These fig-

ures are highly domain-dependent and each of the different areas could be more or less expensive in other situations. For example, if more elaborate constraints were being generated, the constraint generation time would be higher and consume a larger proportion of the processing time. Likewise, if translation were more difficult and involved some semantic interpretation as well as strict syntactic replacement, it would take more time. The question that must be asked is not whether information sharing and assimilation takes time—it does. Rather, the questions to ask are:

1. What information should be shared by each agent?
2. For each type of information that can potentially be shared, will sharing it decrease or increase the processing time of the system?
3. For each type of information that can potentially be shared, will sharing it improve solution quality in the system?

Although it is possible to empirically answer the second and third questions for a particular system and, therefore, to tune the system to appropriate tradeoffs in quality and processing time based on information sharing, the first question cannot be answered in any general way when agent reusability is an issue. In the assimilation experiments described above, only three of the seven agents instantiated information-assimilation capabilities. The primary reason for this is that implementing these capabilities is very difficult. For each agent, the implementation is unique and requires a thorough understanding of the information requirements and search mechanisms of that agent. This suggests that it must be done by the agent implementor at the time the agent is built. The agent implementor cannot be responsible for determining what information will be relevant in a particular application system since the agent may be embedded in different systems. However, the agent implementor must determine what internal knowledge will be sharable. Furthermore, the agent developer must anticipate the types of information that may become available to the agent during problem solving and build into the agent the capabilities required to effectively apply that information.

We demonstrate the difficulty inherent in implementing effective information assimilation through an example. Say that the **pump agent** receives a constraint from the **heat-exchanger agent** that restricts the *run-head* parameter of pumps proposed by the pump agent. This constraint is not directly applicable during the search for candidate pumps because the value of *run-head* is computed after the specific pump is chosen: it is an output parameter rather than an input parameter. However, once a candidate pump has been generated, the *run-head* for that pump can be computed and the constraint can be applied as a filtering mechanism to eliminate non-viable candidates. If pump-agent does apply the filtering constraint, it will still have to iteratively generate and test candidate pumps locally, but will eliminate infeasible ones before other agents are asked to respond to them. Therefore, by appropriately applying assimilated information, it can reduce the workload of other agents.

The point here is that it is not only necessary to understand the language of received information, it is also necessary that the agent know how to apply it. Applying the information appropriately can be very subtle because it may have to be applied differently than the agent's own local knowledge, for example, as a post-search filter as described above. This implies that an agent must anticipate the kinds of information it may receive and have internal procedures available to effectively use that information.

5 Conclusions

In this paper, our objective was to show that the ability of agents to assimilate external information about the composite search space and use this information to guide local search affects both solution

quality and system performance. The experiments in Section 3 demonstrated conclusively that this is so. When external information is assimilated by an agent, that agent is able to focus its search efforts in areas of its local solution space that are more likely to be contained in the composite solution space as well. By focusing its search in areas that are likely to be mutually acceptable, the agent's work is more productive and will tend to improve both solution quality and system performance. However, there are implementation and performance costs associated with information sharing and, in some situations, these costs may outweigh the benefits.

After empirically demonstrating the benefits of information assimilation in multiagent problem solving, we took a detailed look at the costs of assimilation. We classified the costs of information sharing as involving: the generation of information to share; the translation of information into and out of a shared language; the determination of what information to communicate at any given time; the transmission of information[5]; and local management (storage, retrieval, and use of potentially conflicting assimilated information). We observed these costs within the **STEAM** system and found them to total approximately 10.17% of the overall runtime. Although this is not a trivial figure, in this domain, the time spent in sharing information is more than balanced by the productivity enhancement that comes from focusing on mutually acceptable areas of the composite solution space.

Our experience with information assimilation suggests some conflicting perspectives on achieving information sharing in systems of heterogeneous and reusable agents. On the one hand, the experiments showed that information sharing and assimilation can be highly effective in improving system performance, both in terms of solution quality and runtime. On the other hand, we found sharing and assimilation difficult to actualize because they require in-depth understanding of the domain characteristics of individual agents. The application-system developer that is responsible for integrating a set of reusable agents into a system cannot be expected to have a deep enough understanding of individual agent domains to install the necessary mechanisms into the agents.

Information sharing requires that each agent know: 1) what information it can share; 2) what information it can assimilate; and 3) how information that is assimilated from external sources is to be applied. When the mechanisms required for information sharing are installed at agent-implementation time, the implementor will not know whether shared information will ever be used, whether anticipated information will ever arrive, or whether functional capabilities for applying certain types of information will ever be applied. If the agent is implemented with highly sophisticated information-sharing capabilities, it must be expected that in any given application system, these capabilities may be beyond what is required or even usable for the domain. The price of generality goes beyond implementation costs for the agent, since there may be system-wide runtime repercussions based on the transmission of unusable information, or on applying assimilated information that degrades system performance rather than enhancing it. Future research in reusable-agent systems should examine questions of balancing the information-sharing capabilities of agents with the benefits of sharing various types of information. It may be that some general guidelines will emerge that can be applied by agent implementors to decide what capabilities are likely to be beneficial in an agent.

In conclusion, we have shown that information sharing and assimilation can enhance system performance in the **STEAM** system. Although there is no basis on which to generalize any specific figures outside of **STEAM**, the **STEAM** domain is typical of a class of small-scale globally cooperative design domains. This leads us to believe that information sharing and assimilation can improve performance in this class of systems. Furthermore, the categories of information-sharing costs described in this chapter hold across all domains. Both the empirical evidence demonstrated here and intuitive arguments for the benefits of focused search suggest that information sharing and assimilation will

[5]Although we recognize that transmission of information will add to the cost of information sharing, it was not included as one of the categories in our experiments. Because of the physical environment in which our experiments were run, these costs were trivial.

be effective in more complex domains. However, although information sharing is potentially beneficial, it is not particularly easy to achieve. Most of the work must be done at agent-implementation time when nothing is known about the application system(s) into which the agent will be embedded. The costs of making agents that are highly proficient in sharing and using assimilated information may outweigh the benefits that accrue from those capabilities. Future work may clarify the boundaries of benefit versus hindrance based on types of information and the capabilities required by agents to use those various types. Until that time, however, it is clear that information sharing and assimilation should be considered a potential source of performance enhancement when designing distributed-search systems comprising heterogeneous reusable agents.

References

[1] Avron Barr and Edward A. Feigenbaum. *The Handbook of Artificial Intelligence, Volume 1.* William Kaufmann, Inc., 1981.

[2] S.E. Conry, K. Kuwabara, V.R. Lesser, and R.A. Meyer. Multistage negotiation for distributed satisfaction. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1462–1477, November/December 1991.

[3] Pedro S. de Souza and Sarosh Talukdar. Asynchronous organizations for multi-algorithm problems. In *Proceedings of the ACM Symposium on Applied Computing*, Indianapolis, Indiana, February 1993.

[4] Tim Finin, Rich Fritzson, Don McKay, and Robin McEntire. KQML: A language and protocol for knowledge and information exchange. Technical report, Universitiy of Maryland, Baltimore, MD, 1994.

[5] M.S. Fox, B. Allen, and G. Strohm. Job-shop scheduling: an investigation in constraint-directed reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 155–158, Pittsburgh, Pennsylvania, August 1982.

[6] Taha Khedro and Michael R. Genesereth. Progressive negotiation: A strategy for resolving conflicts in cooperative distributed multidisciplinary design. In *Proceedings of the Conflict Resolution Workshop, IJCAI-93*, Chambery, France, September 1993.

[7] Mark Klein. Supporting conflict resolution in cooperative design systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1379–1390, November/December 1991.

[8] Susan E. Lander. *Distributed Search and Conflict Management Among Reusable Heterogeneous Agents.* PhD thesis, University of Massachusetts, Amherst, Massachusetts, May 1994.

[9] Susan E. Lander and Victor R. Lesser. Understanding the role of negotiation in distributed search among heterogeneous agents. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 438–444, Chambery, France, August/September 1993.

[10] Victor R. Lesser. An overview of DAI: Viewing distributed AI as distributed search. *Journal of the Japanese Society for Artificial Intelligence*, 5(4):392–400, July 1990.

[11] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, Fall 1991.

[12] Yoshiyasu Nishibe, Kazuhiro Kuwabara, and Toru Ishida. Effects of heuristics in distributed constraint satisfaction: Towards satisficing algorithms. In *Workshop on Distributed Artificial Intelligence*, pages 285–302, Michigan, February 1992.

[13] Arvind Sathi and Mark S. Fox. Constraint-directed negotiation of resource reallocations. In Les Gasser and Michael Huhns, editors, *Distributed Artificial Intelligence, Volume 2*, chapter 8, pages 163–193. Pitman Publishing, London, 1989.

[14] K. Sycara. *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytic Methods.* Ph.D. thesis, Georgia Institute of Technology, Atlanta, Georgia, June 1987. Also published as Technical Report GIT-ICS-87/26.

[15] Katia Sycara. Arguments of persuasion in labour mediation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 294–296, Los Angeles, California, 1985.

[16] Gilad Zlotkin and Jeffrey S. Rosenschein. Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1317–1324, November/December 1991.