Distributed Heuristic Synthesis Search

Daren Krebsbach,*[,†,§] Herbert Gelernter,[†] and Scott McN. Sieburth[‡]

Departments of Chemistry and Computer Science, State University of New York at Stony Brook, Stony Brook, New York 11794

Received December 17, 1997

The Stony Brook SYNCHEM program, a well-established intelligent problem-solving system for synthesis route discovery, recently has been upgraded to execute in a network of multiprocessor workstations under both Linda tuple space and PVM message passing protocols. We describe the implementation and report experimental results. In addition, we discuss the effect of parallelizing SYNCHEM's search algorithm on the global best-first heuristic exploration of the synthesis problem space. Though reasonably wide-ranging and inclusive, sequential exploration exhibits a pronounced depth-first flavor. Parallelization introduces a countervailing breadth-first tendency that alters the region of the problem space explored by the search engine with mixed consequences, the positive clearly outweighing the negative.

1. INTRODUCTION

SYNCHEM is a large knowledge-based problem-solving system for synthesis route discovery. It has been designed to offer the practicing organic chemist a tool for discovering valid synthesis routes for molecules of interest without the need for online guidance.[1,2] Some of the synthesis routes proposed by the SYNCHEM system have been original and have been considered suggestive, useful, and even creative by chemists who have experimented with the system. Such a synthesis design program could be optimized for a variety of tasks, from total syntheses of scarce natural products (for instance, taxol[3]) to cost-effective large-scale solution phase syntheses of pharmaceuticals discovered using combinatorial chemistry.[4]

Attempts to design computer-based synthesis planning tools go back a quarter of a century; some have exploited search systems, some have relied upon algebraic mechanisms, and a number have combined both approaches.[5−8] Of the heuristic search systems that are still under development, SYNCHEM is the only one that executes a self-guided global best-first exploration of the problem space. While other search-based synthesis-discovery programs can be run non-interactively, all of these (as described in the literature) have been restricted to bounded ordered depth-first search. Bersohn introduces a number of sophisticated chemistry-derived heuristics to limit the growth of the problem space, and these have enabled his system (called SYNSUP-MB) to find some interesting solutions to some interesting problems,[9] but bounded ordered depth-first search can become extremely inefficient if the search guidance function is not perfect.

SYNCHEM approaches its task by executing a retrosynthetic (backward-chaining) search from target compound to starting materials, successively applying known organic reactions represented as transform operators. In addition to generating the precursors of the given molecule as specified by the graph rewriting rule of the transform operator, the system performs extensive tests, first to ensure that the application of the given reaction is a valid one, and second to calculate the heuristic merit of the reaction application. These tests take into consideration such factors as functional group compatibility, electronic effects, steric hindrance, leaving group ability, ring strain, etc. Even with its currently modest knowledge base of about 1000 reactions, a branching factor of dozens of descendent nodes for each developed node of the search space *after* heuristic pruning is not unusual.

The application of a transform operator, both the graph rewriting phase and test evaluation, can be computationally intensive. Whereas a straightforward problem can take as little as a few minutes to solve, a more complex problem—those that a chemist might routinely submit to the system—can take several hours. Since the knowledge base for a production-level version of the system will entail a 10-fold increase in size,[10] it is clear that every possible approach to improving the efficiency of the search must be pursued. Foremost among these is the parallelization of the search for distribution over the increasingly ubiquitous network of workstations in which the system is likely to be installed in a chemistry laboratory.

In this paper we describe our efforts to date in implementing an efficient distributed version of the SYNCHEM system. The compounds we have chosen for our experiments are by no means leading-edge targets, their syntheses having been carried out by bench chemists long since. The criteria in choosing the compounds was their varying synthetic complexity within the limitations of the current knowledge base of SYNCHEM in order to measure the performance of our distributed heuristic search program. In section 2 we explain how SYNCHEM performs heuristic search, and in section 3 how we intend to distribute the search over a network of workstations. We define the metrics we will use to measure the performance of distributed SYNCHEM in section 4 and

[†] Department of Computer Science. E-mail: daren@cs.sunysb.edu and gelern@cs.sunysb.edu.

[‡] Department of Chemistry. E-mail: ssieburth@notes.cc.sunysb.edu.

[§] Current address: SmithKline Beecham Pharmaceuticals, 709 Swedeland Rd UW2810, P.O. Box 1539, King of Prussia, PA 19046. E-mail: daren_krebsbach@sbphrd.com.

S0095-2338(97)00115-7 CCC: $15.00
Published on Web 06/06/1998

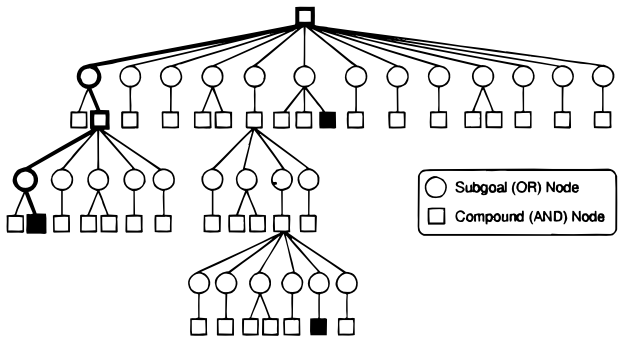

Figure 1. A problem solving graph (PSG) with the current best pathway indicated with bold lines.

present our experimental results. In section 5 we propose modifications, based on our analysis of the experimental results, which should lead to major improvements in the current implementation of distributed SYNCHEM.

2. HEURISTIC SEARCH IN SYNCHEM

There are many forms of heuristic search reported in the AI literature.[11] To avoid confusion, we will describe briefly the SYNCHEM best-first search algorithm. We will restrict our description here to those features that are pertinent to the parallelization of the search algorithm.[12]

SYNCHEM takes a problem reduction approach to synthesis discovery, whereby an organic compound is transformed, via the retrosynthetic application of known reactions, to simpler subproblems, each of which may be composed of more than one precursor molecule. The resulting problem reduction space is represented by a problem-solving graph (PSG), or AND/OR graph, as shown in Figure 1.

A *compound node* (AND node) contains either the target compound, an intermediate compound, or an available compound (starting material). A compound node is solved if either it is an available compound, or if *any one* of its descendent subgoal nodes has been solved. The compound node merit is 100 (the highest possible value) if the compound is available; it is the maximum of the merits of its descendent subgoal nodes if the compound node is an intermediate, or it is calculated using a heuristic predictor function, which estimates the merit of the remaining pathway from the current node to starting materials, if it is an undeveloped leaf node.

A *subgoal node* (OR node) contains information about the reaction that generated it (including the results of the reaction evaluation tests), which is used to calculate the reaction merit. A subgoal node is solved if *all* of its descendent compound nodes have been solved. The subgoal node merit is the product of the reaction merit and the downwardly adjusted minimum merit of its descendent compound nodes; solving a subgoal is at least as hard as solving its most difficult conjunct.

The search itself cycles through three phases: *selection*, *expansion*, and *update*. During the selection phase, a global best pathway is chosen for continuation. Beginning with the target compound node, its descendent subgoal with the highest merit is added to the best pathway. Among the conjuncts of the subgoal, the compound node with the lowest merit is chosen. The rationale for this tactic is that there is little sense in spending the time to solve the simpler subproblems until one knows that one can solve the more difficult subproblem, or, to put it another way, if the chosen pathway is going to fail to find a solution, one should fail as soon as possible. The selection process recursively descends the PSG, alternately choosing the subgoal node with the highest merit and the compound node with the lowest merit, until an undeveloped compound node has been selected (a leaf node). This compound node is called the *selected compound* and its parent the *selected subgoal*. Figure 1 shows the best pathway, where it is assumed that both the compound and subgoal nodes are sorted in descending order of merit from left to right.

In the next phase, expansion, the subgoals for the selected compound are generated by exhaustive application of the relevant transform operators in the reaction knowledge base. Before incurring the cost of embedding the goal pattern of the graph rewriting rule in the selected compound, reactions are screened for the presence of a reaction center in the goal compound and the absence of functional groups sensitive to the conditions of the given reaction. Many of the reactions are eliminated from further consideration by this filtering process. After the precursor compounds have been generated for the subgoal, more sophisticated tests can be performed: is the reaction site adequately activated, is a favored size of ring formed, etc. These tests can either reject this application of the reaction outright or adjust the reaction merit. Finally, the precursor molecules themselves are checked for validity—for instance, is Bredt's rule violated? The resulting set of subgoals (with their associated compound nodes) is referred to as the *expansion* of the selected compound.

In the final phase, the expansion of the selected compound is added to the PSG, and a global update of the node merits in the PSG is performed. First, the newly generated compounds are looked up in the database of starting materials and, if found, are given a merit of 100. Otherwise, the merits are calculated using the heuristic predictor function. Next, the merits of the subgoals in the new expansion are calculated. The maximum subgoal merit in the new expansion replaces the estimated value of the predictor function in the selected compound. This new merit is backed up along the current pathway. Furthermore, if the selected compound exists anywhere else in the PSG, the new expansion and compound merit are assigned to those compound nodes with the duplicate instances, and the new merit is also backed up along the pathways leading from those compound nodes to the root node. In Figure 1, the shaded leaf node on the best pathway was selected for expansion. The other shaded compound nodes to the right contains other instances of the compound; their paths to the root are also backed up with the new compound merit. This process of backing up the merits for all duplicate instances of a given compound is done not only for the selected compound but also for any compound along a pathway whose merit is recalculated during the global update of the PSG.

There are a few important characteristics of the SYNCHEM best-first search algorithm worth noting. Unlike many other heuristic search algorithms, the execution time required to generate nodes during compound expansion places a much higher demand on computational resources than the size of the search space. On average, roughly 35% of the compound nodes in a PSG are duplicate instances of

```
Master()
1  Initialization ()
2  while (not end_search)
3     Select (SelComp)
4     out (SelComp)
5     in (Expansion)
6     Insert (SelComp, Expansion, PSG)
7     Update (PSG)
8  Save (PSG)
```

```
Worker()
1  Initialization ()
2  while (true)
3     in (SelComp)
4     foreach Rxn ∈ KB do
5        Apply (Rxn, SelComp, Expansion)
6     foreach NewComp ∈ Expansion do
7        CalculateMerit (NewComp)
8     out (Expansion)
```

Figure 2. The distributed search algorithm.

another compound, so there are significant interdependencies throughout the search space. In particular, solving a subproblem (a conjunct of a subgoal) along one pathway can improve the merits of other pathways far removed from the current pathway, often promoting those other pathways in the global ranking. Indeed, many of the solutions proposed by the system contain subproblems that were solved while exploring a different region of the search space. Finally, the compound and subgoal merits in the PSG are not static but vary in response to new information on the progress of the search after each expansion cycle.

3. DISTRIBUTED HEURISTIC SEARCH

There are several possible methods of decomposing the heuristic search problem for parallelization, each with its own advantages and disadvantages. The choice of a computing environment does impose limitations on the granularity and type of tasks assigned to each processor and where the search space itself resides. Because we intend to distribute the parallelized version of SYNCHEM over a local area network of workstations, interprocessor communication should be minimized as much as possible. It is also desirable to retain the best-first character of the heuristic search.

Although the literature describes a number of research programs concerned with parallel heuristic state-space search, only Kale and Saletore have seriously discussed problem-reduction systems which lead to AND/OR search spaces, and the problem domains considered are orders of magnitude less complex than that of synthetic organic chemistry.[13] In particular, they deal with a limited number of well-defined transform operators (unlike the large collection of relatively incompletely specified and constrained reaction transforms in our system), branching factors are usually low, and node expansion task granularity is minimal. Most other reported research deals with problems of parallelizing various formulations of heuristic A* state space search.[14−17] The logic programming community has explored methods of distributed AND/OR search, but here either pure depth-first or iterative depth-first search strategies are employed.[18,19]

We have chosen to exploit *task-parallelism* using the Master-Worker Model of distributed computing. The reasons are many, but the most significant consideration is the management of the search space. Both the selection and update procedures are global operations; to be effective, they require access to the entire search space, which typically contains tens of thousands of subgoal and compound nodes. If the search space were to reside in distributed shared memory, as would be necessary under a Peer Model where each processor performs the same tasks on shared data, the communication overhead for each processor in accessing the search space for global selections and updates would be unacceptably high. Dividing the search space up among the peer processors—exploiting a form of *OR-parallelism*—would lead either to significant duplication of work or considerable communication overhead as each peer would have to inform every other peer of the selections it has made. This is because the search space is a graph with numerous interdependencies rather than a tree with independent branches. Also, this form of OR-parallelism would preclude the use of a global best-first search strategy, and there would be the additional difficulty of merging the subspaces into a coherent whole at the end of the search.

Under the Master-Slave Model, the master process maintains in local memory the search space and is given the task of managing the workers and the search space: global best-first selection of compound nodes for expansion (allocating tasks) and PSG update (processing results). The workers are responsible for compound expansion. The algorithm for distributed SYNCHEM is shown in Figure 2. Earlier prototypes of the algorithm were implemented using both Linda (distributed shared memory) and PVM (message passing). While the PVM version was slightly more efficient, the Linda version proved to be easier to modify and maintain. We chose to continue the development of distributed SYNCHEM using Linda, the results of which are reported here.

Linda supports a distributed shared memory using a *tuple space*. Distributed SYNCHEM treats the tuple space as a message board and uses two Linda operations for interprocessor communication: **in** removes a tuple from tuple space, and **out** places a tuple in tuple space. The **in** operation is a blocking read: if there is no tuple in the tuple space that matches the arguments of the **in** operator, then the process waits until one becomes available. A consequence of the above distributed algorithm is that at least one processor is always waiting. After a worker places its results in tuple space, it waits until the master places a new selected compound in tuple space, usually after processing that worker's results.[20] If all of the workers are busy, then the master waits until one of the workers completes its task and places an expansion tuple into tuple space.

4. EXPERIMENTAL RESULTS

There are several metrics for evaluating the performance of distributed heuristic search; the most commonly reported

is the speedup in finding a first solution. This is not a very useful metric for synthesis discovery, nor for many other complex problem domains, because more often than not, the first discovered solution is not the best solution.[21] In fact, the first solution may not even be a valid synthesis of the target compound. As with many knowledge-based systems, SYNCHEM does not have perfect knowledge of its domain. Many of its reaction descriptions are fairly complete and accurate, but some are missing the constraints that would preclude their inappropriate application under unfavorable conditions,[22] so SYNCHEM does occasionally generate a solution with an invalid reaction step. The objective of the SYNCHEM system is to generate as many potential solutions as is possible in a reasonable amount of time. This is in accordance with the demands of the chemist, who would like to consider many diverse syntheses of the target compound before deciding on any one approach.

Another limitation of the first solution metric is that it does not distinguish measuring the effectiveness of parallelizing the search engine code—where the predominate issues are limiting communication overhead and maximizing processor load balancing—from measuring the effectiveness of the heuristic search guidance function when multiple nodes are being expanded concurrently. As a case in point, while conducting experiments with the distributed version of the system, we observed that the time to first solution can vary considerably from one run to the next when the same parameters are used. In fact, for some problems, the first solutions themselves can differ. This is due to the slightly different order in which node expansions are completed on the network of processors, altering the selection of the current best pathway, and thus resulting in different regions of the search space being explored. This reflects the quasi-chaotic behavior of distributed heuristic search as realized by the SYNCHEM system.

We thus use two sets of metrics. In the first, we compare the computation times of sequential and distributed SYNCHEM in generating *identical search spaces*. These experiments allow us to analyze the effectiveness of the distributed implementation independent of the behavior of the best-first selection. In the second, we compare the computation times and number of expansion cycles needed to discover a given *set of solutions*, those chosen by a chemist as the most promising syntheses.

The experiments for distributed SYNCHEM were run on a local network of nine Sparc20 workstations, each with four 50 MHz processors and 64MB RAM, and connected with 10Mb Ethernet. The workers were uniformly distributed by Linda over the available processors.

4.1. Experiments in Generating Identical Search Spaces. To determine the appropriateness of the Master-Worker model for distributed heuristic search and the effectiveness of our implementation, we compared the sequential and distributed versions of SYNCHEM in generating *identical search spaces*. This was accomplished by first executing distributed SYNCHEM for 1000 expansion cycles, saving the compound nodes which were chosen for expansion in the order in which they were returned to the master, and then generating the same search space using sequential SYNCHEM by forcing it to make the same selection of compound nodes for expansion. In this execution mode of sequential SYNCHEM, the best-first selection routine is not called, so the response times of sequential SYNCHEM were adjusted using the selection time for distributed SYNCHEM. Two target compounds were chosen for the experiments: sulforhodamine, which on average has a relatively low branching factor (9.4), and cortisone, which has a relatively high branching factor (18.2). The experiments were conducted using from 1 to 30 workers in increments of 5. The response times and speedups are shown in Figures 3 and 4, respectively.

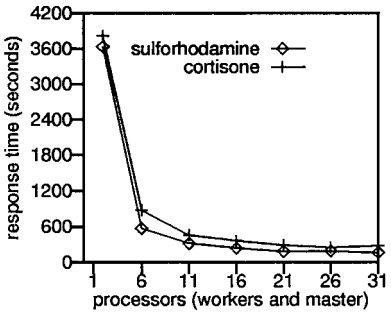

Figure 3. The response times of distributed SYNCHEM in generating identical search spaces.

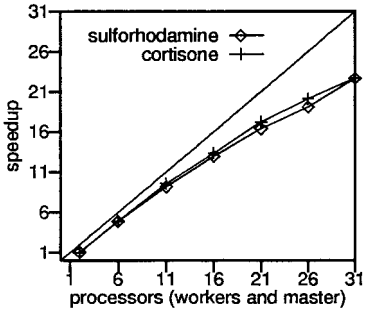

Figure 4. The speedups of distributed SYNCHEM over sequential SYNCHEM in generating identical search spaces.

Because there are differences between any two search spaces generated by distributed SYNCHEM for the same problem but using different numbers of workers, it is difficult to make direct comparisons of the response times. Nonetheless, from the graph in Figure 3, one can see a leveling off of the response times following a sharp reduction. In comparison, the forced sequential response times corresponding to the distributed searches for sulforhodamine ranged from 2781.70 s (5 workers) to 3682.57 s (25 workers), and for cortisone they ranged from 3935.46 s (1 worker) to 6356.10 s (30 workers). The increase in execution times for the forced sequential searches can be attributed to the breadth-first character of distributed search in which a greater number of compound nodes near the root of the PSG are expanded. These compounds tend to require more time to expand than compounds nodes near the leaves of the PSG, which represent simpler subproblems.

The response times reported for the one worker plus master runs are slightly above those of sequential SYNCHEM even though the workload is divided among two processors. This is because one of the processors is always blocked: either the worker is waiting for the master to select a compound, or the master is waiting for the worker to finish an expansion. This also explains why the speedups are always less than linear by a factor of at least $1/p$, where p is the total number of processors dedicated to the search. Of course, there are more significant reasons why the speedups are sublinear.

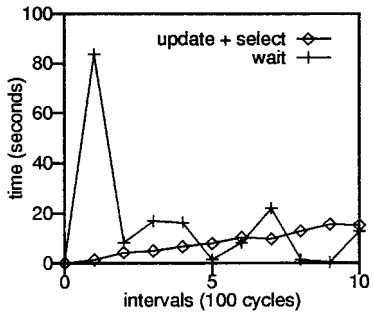

Figure 5. The execution and wait times for the master in generating the search space for cortisone using 30 workers. The reported times were accumulated over 100 expansion cycle intervals.

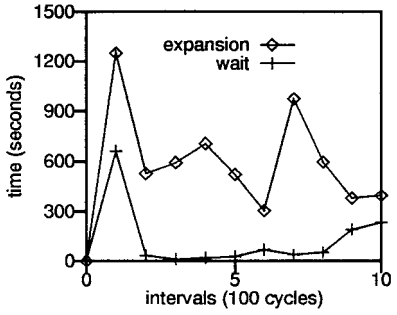

Figure 6. The expansion and wait times for the workers in generating the search space for cortisone using 30 workers. The reported times were accumulated over 100 expansion cycle intervals.

Figures 5 and 6 show the breakdown of the execution times for the master and the workers, respectively, over the course of the 30-worker cortisone run. The times plotted are summed over 100 cycle intervals. The master spent 16.73% of its time in selection, 14.85% in update, 3.24% in communication, and 61.05% waiting; the workers spent 82.21% in expansion, 0.04% in communication, and 17.47% waiting. Communication overhead does not appear to have much of an impact on the response time and speedup, but the amount of time the master spends waiting for workers to finish their tasks and the amount of time the workers spend waiting for the master to select a task are major factors. Unfortunately, large portions of the master and worker wait times are unavoidable since they occur while the search is being *primed* (the initial spikes in the graph of Figures 5 and 6). At the start of the search, there is only one compound to expand, the target compound. During the time that it takes one worker to expand the target compound, the master and the other workers must wait. After the workers have been given their tasks from the first expansion, the master again waits for those results. By the end of the first 100 cycles of the cortisone run, the master had spent a total of 83.79 s waiting and the workers 660.07 s waiting. Similarly, at the end of the search when the master is no longer making selections but only collecting expansions and updating the PSG, there was another smaller peak in the master wait time. The rest of the master and worker wait time can be attributed to an imperfect load balance and the interleaving of the master and worker tasks. For instance, during the second 100-cycle interval, the master spent 4.22 s managing the search space while on average a worker spent 17.59 s expanding nodes; the master spent the difference (13.37 s) waiting. In contrast, during the ninth 100-cycle interval, the master spent 15.52 s managing the search space while on average a worker spent 12.65 s expanding nodes; thus, each worker spent roughly 1.23 s waiting. Or, to look at it from a different perspective, toward the end of the 1000-cycle search, the master was only able to keep pace with roughly 25 workers; the other five workers remained idle waiting for the master to select new tasks.

This load imbalance becomes more pronounced when the search is extended beyond 1000 cycles. As the search space grows larger, the master requires more time to execute its tasks, as can be seen in Figure 5. Conversely, although there can be pronounced variation, the execution time of the workers tends to diminish as a greater number of simpler subproblems are chosen for expansion. Thus, more workers fall idle as the search progresses, which reduces the amount of speedup that can be achieved by distributed SYNCHEM. Even with this limitation of task-parallelism, our implementation is able to gain reasonable speedups with quite tolerable response times. In section 5.3 we discuss a modification that should improve the performance of the system.

It is worth noting that the branching factor does not appear to have a significant affect on the performance of distributed SYNCHEM. While the response times for the lower branching sulforhodamine problem are less than the higher branching cortisone problem, as expected, the speedups achieved are unexpectedly similar. An examination of the execution times for the 30 worker experiments reveals the reason: for sulforhodamine, the distributed response time was 159.26 s (the master spent 11.69 s in update, 19.88 s in selection, and 109.71 s waiting), whereas the sequential response time using forced selection was 3604.28 s; for cortisone, the distributed response time was 280.03 s (the master spent 41.58 s in update, 46.85 s in selection, and 170.95 waiting), whereas the sequential response time using forced selection was 6356.10 s. Problems with a higher branching factor require more time for expansion, but the resulting search spaces also require more time for selection and update. Thus, the speedups were 22.63 for sulforhodamine and 22.70 for cortisone. Further experiments with a diverse collection of target compounds are needed to confirm that speedups are not significantly affected by the branching factor, but, if this bears out, distributed SYNCHEM should still perform well when the knowledge base undergoes a 10-fold increase in size.

4.2. Experiments in Covering Sets of Solutions. Because SYNCHEM does not have a perfect heuristic guidance function,[23] a sizable region of the search space is often explored before a *set of solutions* is found. Speedups in finding the set of solutions can be achieved when this region is searched in parallel. If this sequentially developed region is small with relatively few explored nodes that do not actually lie on one of the solution pathways, then only a modest speedup can be achieved by distributing the search. Expanding nodes outside this sequentially developed region, however, can have a detrimental effect on the speedup of the search. Not only is time wasted if these nodes do not ultimately lead to a solution, they also can lead the search astray. We conducted several experiments to measure the performance of distributed SYNCHEM in covering sets of solutions.

For these experiments, four compounds were chosen based on their branching factor and on the number of high-quality solutions (Figure 7). A very large search space was

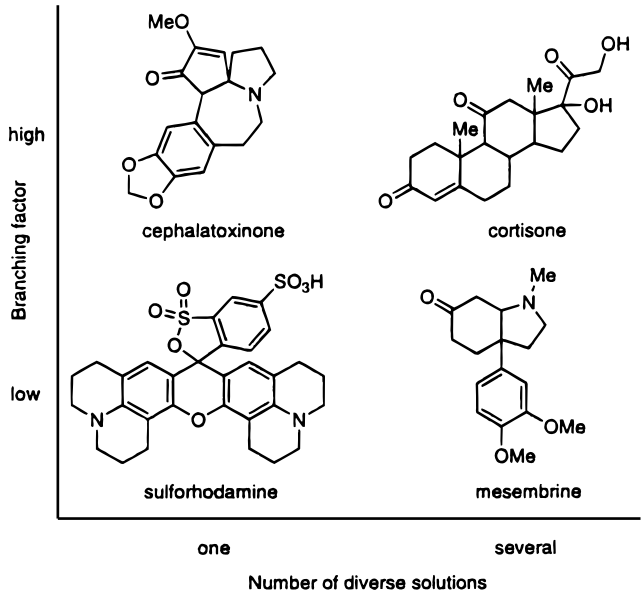

Figure 7. Compounds chosen for the experiments in covering sets of solutions.

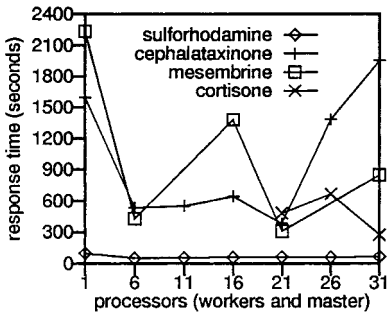

Figure 8. The response times of distributed SYNCHEM using pure best-first search in covering the set of solutions.

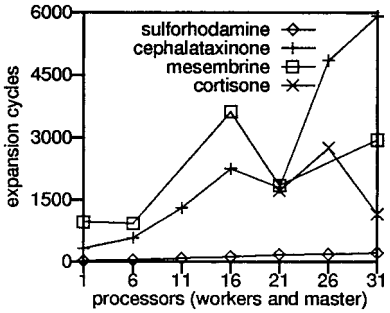

Figure 9. The number of expansion cycles needed by distributed SYNCHEM using pure best-first search to cover the sets of solutions.

generated for each compound, and from the search spaces we selected those syntheses which showed the most promise and exhibited the most diversity. In the cases of cephalataxinone and sulforhodamine, there was no significant diversity in the solutions, so only the best solution for each was chosen. Each compound was submitted to distributed SYNCHEM using from 5 to 30 workers in increments of 5. The response times are shown in Figure 8, and the number of expansion cycles needed to cover the set of solutions are shown in Figure 9. The coverage results reported for one worker are for sequential SYNCHEM. No results are reported for cortisone with 5, 10, and 15 workers (nor with sequential SYNCHEM), and no results for mesembrine with 10 and 25 workers, because the solution sets were not covered in the given amount of time (60 min for distributed SYNCHEM, 240 min for sequential SYNCHEM).

The speedups in finding the set of solutions, while not insignificant, are neither as sizable nor as uniform as those for identical search space generation. One cause is that having multiple workers concurrently exploring the search spaces introduces a breadth-first character to the search, and thus many compounds are selected for expansion that were not selected in the sequential search. If there is a single superior solution and the heuristic search guidance function accurately differentiates this solution from the other possible pathways, then the search should not be lead too far astray (nor would one expect a significant speedup). This was the case for the sulforhodamine problem, which has a solution set that contains eight developed nodes with a maximum depth of 5. Sequential SYNCHEM expanded only 24 nodes before discovering this solution. From this perspective, a maximum speedup of 1.94 (five workers, 42 nodes expanded) for distributed SYNCHEM is reasonable. Even with 30 workers, where many nodes that are not in the solution set are necessarily selected to keep the workers busy, only 215 nodes were expanded (few pathways were pursued beyond the initial selection), resulting in a speedup of 1.45. Such a rosy scenario was not evident in the case of the cephalataxinone problem. Sequential SYNCHEM expanded 324 nodes in 1596 s to cover the solution set, which contains 13 developed nodes with a maximum depth of 10. Thus, there was a much greater potential for speedup, which, however, remained unrealized. The best speedup was 4.18 with 20 workers. The explanation can be seen in the much larger region of the search space explored by distributed SYNCHEM. Whereas sequential SYNCHEM expanded roughly 24 nodes for every developed node in the solution set, distributed SYNCHEM expanded as many as 454 nodes per developed node in the solution set. Not only are nodes outside the sequentially developed region being selected for expansion, but entirely new regions of the search space are being explored as well.

Another cause for less than linear speedups is that pure best-first search was used for these experiments, whereas the search spaces from which the sets of solutions were selected had been generated with a best-first search strategy augmented with effort distribution (described in the next subsection). Thus, nodes that had been expanded (and had led to solutions) during the sequential search were expanded much later in distributed search or were not expanded at all. For cortisone the latter situation predominated when too few workers were used but was averted when 20 or more workers broadened the exploration of the search space. In particular, the 30 worker cortisone run had a substantial speedup of 25.67 over the sequential run using effort distribution. The mesembrine problem has four syntheses in the solution set. In all the experiments with mesembrine, three of the syntheses were covered within the first 315 cycles (60 cycles in the sequential run). Once the pathway leading to the fourth synthesis was developed to a depth of two, the rest of the solution was covered in short order, but exploration of that pathway was delayed. Both low predictive heuristic merits of the fourth synthesis pathway and an enlargement of the region of the search spaced being explored contributed the delay and to the relatively low speedups (the highest was

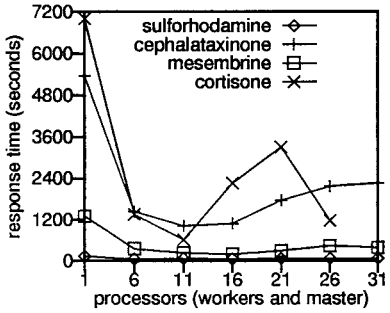

Figure 10. The response times of distributed SYNCHEM using best-first search augmented with effort distribution in covering the set of solutions.

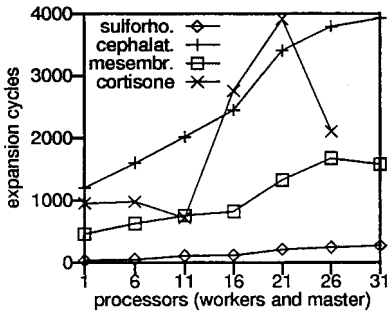

Figure 11. The number of expansion cycles needed by distributed SYNCHEM using best-first search augmented with effort distribution to cover the sets of solutions.

7.23 with 20 workers). In short, our implementation of distributed search using a pure best-first strategy can perform quite well or quite poorly in covering a set of solutions depending upon the distribution and the number of desirable solution pathways in the search space. We next examine whether the performance can be improved with added control of selection.

4.2.1. Effort Distribution. Early in the development of SYNCHEM, it was recognized that the reaction merits and predictor function alone could not be relied upon to guide the search effectively. For a continually evolving domain as complex as organic chemistry, the knowledge base would be in a perpetual state of incompleteness and prone to contain errors. SYNCHEM's best-first search algorithm tends to continue down the best pathway until the merit of that pathway degrades appreciably. This was seen as putting all of one's eggs into a single basket. To add more breadth-first character to the sequential search, a meta-strategy to control the search was introduced into the search engine.

Two tactics to control the search were introduced. The first temporarily closes a newly solved pathway for a specified number of cycles to force the search into other regions of the search space. This prevents SYNCHEM from spending an inordinate amount of time making minor improvements to a given solution, when there may exist reasonable alternatives that might not reveal themselves until they have been further developed.

The second tactic takes into consideration the amount of time that has been spent developing a pathway. Subgoals are allocated search time that is proportional to their current merit. The selection process will skip over those subgoals with higher merit if their time limit has been reached and continue down a presently less favored pathway which might possibly lead to a useful solution. This also helps spread the effort across several subgoals that have nearly the same merit, and it limits the time SYNCHEM might spend developing a pathway which includes a misapplied reaction or an overvalued node. This tactic allows SYNCHEM to perform reasonably well in the presence of defective knowledge.

Effort distribution can prevent distributed search from spending excessive time down a pathway, just as in sequential search. Figures 10 and 11 show the results when the experiments were performed again, this time using effort distribution with the number of temporary closed cycles set to 5. (Note that the response time and expansion cycle scales have changed.) There are a few differences worth pointing out: in all experiments except for the 30 worker cortisone run, the sets of solutions are now found for cortisone and mesembrine within a reasonable amount of time—in fact, the response times and number of cycles for mesembrine, a multiple-solution problem, improved substantially over the search without effort distribution; the single-solution problems, cephalataxinone and sulforhodamine, take significantly longer, which is not unexpected, since this also occurs in sequential SYNCHEM. The results for the cortisone problem present a bit of a conundrum: without effort distribution and 20 or more workers or with effort distribution and 10 or fewer workers, distributed SYNCHEM performs exceptionally well; otherwise, it performs somewhat poorly. We discuss some of the reasons for this behavior when we introduce possible improvements to the implementation in section 5.

4.2.2. New Solutions. In most cases many more compound nodes are expanded in distributed SYNCHEM than were developed in the sequential version. Does this speculative search amount to wasted effort? In the context of many heuristic search systems, where a single best solution is sought with an admissible guidance function, the answer would be yes. But in SYNCHEM, such speculative search can prove to be quite fruitful. Figure 12 shows the first few retrosynthetic steps of two diverse solutions of cortisone. The bottom synthesis was discovered by distributed SYNCHEM with 10 or 15 workers and without effort distribution but was not found by sequential SYNCHEM with effort distribution. In fact, when the subgoal node leading to this new solution was selected, it had a merit of only 16, ranked 13th highest out of the 51 subgoals for the target compound; the best subgoal had a merit of 38. Sequential SYNCHEM had developed only the best five of these 51 subgoals nodes after 1000 expansion cycles.

The reason the merit for this subgoal was considerably lower than the best subgoal is that a deprotecting reaction was used to generate this subgoal, which SYNCHEM tends to disfavor. The rationale is reasonable: delay developing the precursors of a deprotecting reaction until those pathways where the target compound was decomposed into simpler subproblems have been pursued. However, when such large branching factors are involved, that delay may be indefinite. The breadth-first character introduced by distributed SYNCHEM overcomes some of the biases in the search heuristics.

5. IMPROVEMENTS

The numerous experiments we have performed with distributed SYNCHEM have given us a better understanding

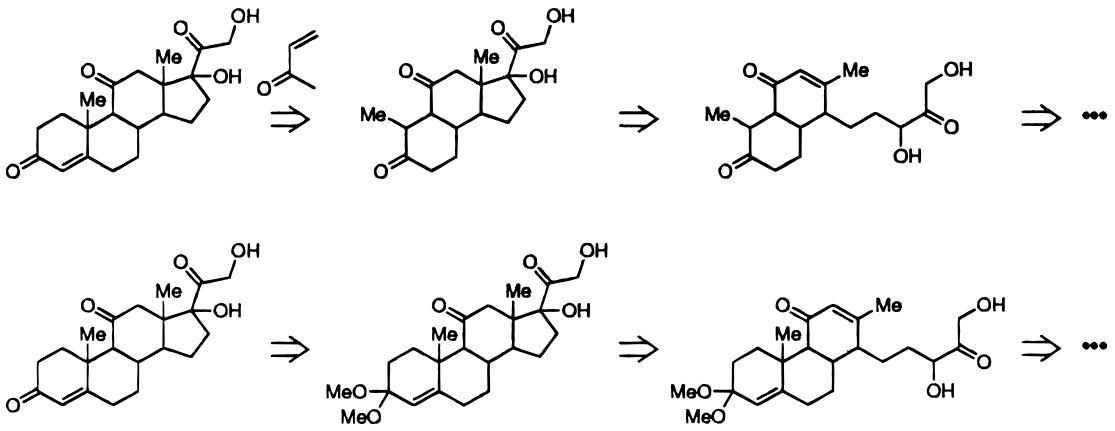

Figure 12. The first few retrosynthetic steps of two different syntheses of cortisone. The top synthesis is one of the best solutions discovered by sequential SYNCHEM with effort distribution. The bottom synthesis was found by the distributed SYNCHEM with 10 workers.

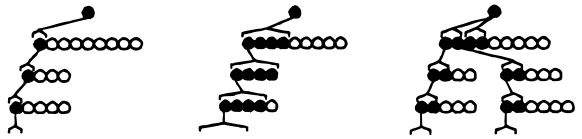

Figure 13. Flavors of search in terms of the width of probe. From left to right: sequential best-first search, distributed best-first search with four workers, and improved distributed best-first search with four workers.

of the issues involved in parallelizing heuristic search. The experimental results have also given us ample clues to the directions one should pursue to improve the current system. We describe a few potential areas of improvement in this section.

5.1. Selection of Next Best Subgoals. Sequential SYNCHEM performs global best-first search. After a compound is expanded, the information about the status of the search—whether the development of the current selected pathway has improved or degraded—is used to update the merits in the PSG before the next subgoal is selected. The current pathway will continue to be developed as long as acceptable progress is being made; this gives the sequential best-first search a strong depth-first flavor, as shown in the leftmost search space in Figure 13. Of course, if the pathway degrades, the lower merit is backed up along that pathway until at some level a subgoal merit is demoted in the ranking, in which case a different pathway will be selected next.

The situation is quite different when several workers are exploring the search space concurrently. While one worker, which we will call the *scout*, is expanding the best compound, the pathway through that compound node cannot be further extended until the scout completes its task. Yet *next best subgoals* need to be selected for the remaining workers before the scout finishes—that is, a best-first selection needs to be made without complete information of the current status of the search because of a delay in acquiring that information. Until the scout completes its task, that pathway tends to remain the current best pathway for subsequent selections, which results in the sibling subgoals of the scout's selected subgoal being chosen. The consequence is a widening of the search probe in the best-first search, as shown in the center search space in Figure 13. This is only a general picture of how the current implementation of distributed search tends to proceed. Recall that the master selects the next subgoal after retrieving the results of a completed expansion and updating the search space. If the pathway for the just completed expansion has improved enough to be now considered the current best pathway, then the next selection will extend that pathway rather than the previous best pathway.

It is not clear, however, that a sibling subgoal of the current best subgoal is always the *global* next best subgoal. While this selection strategy can find new and interesting solutions, as was shown in section 4.2.2, it can also lead to many inferior nodes being expanded. This is because the lower ranking subgoals at the tail of the best pathway may have significantly lower merit than the best subgoal. To ensure that the *global next best subgoal* is selected, the current best subgoal needs to be temporarily removed from the PSG after it is selected and the PSG updated using the merit of the second ranking sibling subgoal (or a merit of 0 if there are no siblings). Thus, the PSG would have to be updated twice every expansion cycle with a postselection update and a postexpansion update. This has the disadvantage of increasing the burden on the master, which would reduce the number of workers it can effectively manage. On the other hand, it should guide the workers to more fruitful regions of the search space and thus reduce the number of expansion cycles required to find the set of best solutions.

5.2. Effort Distribution. Even with an improvement in the distributed best-first selection process to more closely reflect sequential best-first search, there may still be the need for the effort distribution meta-strategy and for similar reasons as were given for sequential search. One would like a more balanced search (as seen in the rightmost search space in Figure 13), where only a proportion of the workers are allowed to pursue any one pathway, and the remaining are forced into less favored but potentially promising regions of the search space. Indeed, experiments reported in section 4.2.1 showed that effort distribution can improve the search for multiple diverse solutions. Its effectiveness, however, is hampered by the delay in acquiring information about how much accumulated time has been spent by the workers down a given pathway—the time spent expanding a compound is added to the nodes along the path after the worker has completed its task. Meanwhile several other workers could have been assigned compounds on that pathway. We have designed a method to maintain an accurate record of the time spent down a pathway using only three parameters for each node: an accumulated time, a last-modified time stamp, and

a worker count, which will only increase the memory requirements for the search space by roughly 5%. It would require that the master update the time spent on a pathway each time a worker is assigned a task, each time a worker completes a task, and each time a pathway is considered for further development. Whether the benefits outweigh the costs will have to be established with further experiments.

It is not clear, however, that a meta-strategy based on effort (time spent developing a pathway) is the best approach to supplement the best-first search strategy when multiple workers are employed in the search. In those experiments of the cortisone problem that performed somewhat poorly, a disproportional amount of time was spent refining solutions discovered early in the search, even when effort distribution and temporary closure of solved pathways were used. There are often multiple pathways to the same intermediate compound; and in the cortisone search space, there were several such intermediates with pathways that had relatively high merits; thus, the workers spent a significant amount of time attempting to improve the solutions to these solved subproblems rather than exploring new regions of the search space. With parallel search, there is already a tendency to broaden the search—what is needed is a more effective method of blocking workers from refining existing solutions until more of the search space has been explored (often the errors and biases of the heuristic evaluation function, particularly an underestimation of the merit of a pathway, can be rectified after that pathway has been extended a level or two). Unfortunately, it is not clear how this can be achieved—further analysis of the performance of distributed SYNCHEM is required.

5.3. Localization of Search. So far we have not addressed the limitations of the Master-Worker implementation of heuristic search: as the search space grows, the master becomes overwhelmed with the tasks of managing the workers and the increasingly complex search space. In fact, we have described improvements in subgoal selection that will further increase the burden on the master. The most reasonable approach to this problem—without abandoning the Master-Worker Model altogether—is to transfer some of the master's tasks to the workers. This can be accomplished by allowing localization of the search.

If, after a worker has expanded a compound and sent its results to the master, there is no *global* next best compound selected by the master available, the worker could be allowed to select a *local* next best compound from its own sub-PSG rather than wait for the master. The worker would then inform the master on which compound it is currently working so that the master itself does not select that compound for expansion by another worker. The master would be required to perform fewer global selections, often its most time-consuming task, in exchange for performing the additional bookkeeping necessary to integrate the local results of the workers with its own PSG.

Each time a worker completes an expansion and before it makes a local selection from the local copy of its sub-PSG, it should check to see if any global next best compounds have already been selected by the master; that is, workers should prefer global selections over local selections. If a global selection is available, the worker would begin a new sub-PSG search with that *global* next best compound as its local target. Furthermore, conditions will be needed on the development of local sub-PSGs so that the global search does not lose its best-first character. For instance, a worker should discontinue to develop its local sub-PSG and wait for a globally selected compound once the local target compound has been solved or if the backed up compound merit for the local target falls below a certain threshold.

There are several advantages to this approach, not the least of which is that by reducing the execution time of the master, we could proportionally increase the number of workers that could be employed in the search, thus improving the scalability of the system. And it may be possible to do so without losing the best-first character of the search. There are disadvantages as well. There will be more communication between the master and the workers. Perhaps more importantly, there may be duplication of effort since a worker could locally select a compound that has already been developed or is currently being expanded by another worker. Idle worker or redundant worker, who is to say which would be less desirable.

Another approach would be to lay off the workers as they fall idle. This approach will not improve the response time of the system, but it will improve the utilization of processor resources by releasing some of them for other purposes. Rather than burdening the master with this task, one of the workers could monitor the tuple space in addition to its normal duties. When the count of completed tasks in tuple space remains above a certain threshold for a length of time, this manager worker can start sending out pink slips. With the improvements in the performance of distributed SYNCHEM outlined above, we hope to avoid such draconian measures.

6. CONCLUSIONS

Real-world heuristic search problems often differ substantively from those that are typically selected by the A.I. community as exemplars for the study of parallel and distributed heuristic search algorithms. It is rarely possible to formulate a real-world search guidance function that is both effective and admissible, and even if one could find such a heuristic, the uncertainties introduced into the evaluation of solution pathways by the intrinsically defective knowledge base entailed by imperfect knowledge of the problem domain could still result in an optimum solution which is invalid because it reflects a misapprehension of the problem. The significance of our work lies in our effort to understand the effects of parallelization and distribution on best-first heuristic search under conditions of uncertain knowledge of what is best. There are clear tradeoffs between the cost of wasted effort expanding disfavored and possibly unproductive nodes and the very real possibility that a region of the search space that our imperfect search guidance function regards as unpromising might contain a superior (or, indeed, the only) solution to the problem.

The experimental results presented here show that we have made significant strides in effectively distributing such a real-world heuristic search algorithm. While we have not been able to achieve unbounded scalability with linear speedup, we have been able to dramatically reduce the response times of the system to within a more acceptable range. Perhaps more importantly, the results of the experiments have confirmed that our general approach to the problem is

feasible, and they suggest where we should direct our efforts to further improve the performance of distributed SYNCHEM.

ACKNOWLEDGMENT

This research was supported in part by NSF Grant No. CDA-9303181. We would like to thank Tito Autrey and Shu Cheung for their efforts in porting the SYNCHEM program to a C/Unix platform. Shu Cheung also assisted in coding earlier prototypes of distributed SYNCHEM.

REFERENCES AND NOTES

(1) Gelernter, H. L.; Sanders, A. F.; Larsen, D. L.; Agarwal, K. K.; Boivie, R. H.; Spritzer, G. A.; Searleman, J. E. Empirical Explorations of SYNCHEM. *Science* **1977**, *197*, 1041−1049.

(2) Agarwal, K. K.; Larsen, D. L.; Gelernter, H. L. Application of Chemical Transforms in SYNCHEM2, A Computer Program for Organic Synthesis Route Discovery. *Comput Chem.* **1978**, *2*, 75−84.

(3) Swindel, C. S. *Org. Prep. Proc. Int.* **23**, 1991.

(4) For an overview of combinatorial chemistry applied to drug synthesis, see the special issue edited by Anthony W. Czarnik and Jonathan A. Ellman. *Acc. Chem. Res.* **1996**, *29(3)*.

(5) For a review, see Barone, R.; Chanon, M. Computer-aided Organic Synthesis. In *Computer Aids to Chemistry*; Vernin, G., Chanon, M., Eds.; Ellis Horwood: Chichester, 1986; 19−102.

(6) Hendrickson, J. B. Organic Synthesis in the Age of Computers. *Angew. Chem., Int. Ed. Engl.* **1990**, *29*, 1286.

(7) Hippe, Z. S. *Artificial Intelligence in Chemistry: Structure Elucidation and Simulation of Organic Chemistry*; Elsevier: Amsterdam, 1991.

(8) Ihlenfeldt, W.-D.; Gasteiger, J. Computer-Assisted Planning of Organic Synthesis: The Second Generation of Programs. *Angew. Chem., Int. Ed. Engl.* **1995**, *34*, 2613−2633.

(9) Takahashi, M.; Dogane, I.; Yoshida, M.; Yamachika, H.; Takabatake, T.; Bersohn, M. The Performance of a Noninteractive Synthesis Program. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 436−441.

(10) The SYNCHEM Group has been conducting research in automated knowledge acquisition of organic reactions. Gelernter, H. L.; Rose, J. R.; Chen, C. Building and Refining a Knowledge Base for Synthetic Organic Chemistry via the Methodology of Inductive and Deductive Machine Learning. *J. Chem. Inf. Comput. Sci.* **1990**, *30*, 492−504. Krebsbach, D.; Gelernter, H. L. Machine Learning of Semantic Constraints for Concept Descriptions Extracted from a Very Large Database. *Found. Comput. Dec. Sci.* **1994**, *19*, 53−69.

(11) For a general discussion of heuristic search, see Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence*; McGraw-Hill: New York, 1971. Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*; Addison-Wesley: Reading, MA, 1984.

(12) For a more detailed explanation of best-first search in SYNCHEM, see: Gelernter, H. L.; Miller, G. A.; Larsen, D. L.; Berndt, D. J. Application of Chemical Transforms in SYNCHEM2, A Computer Program for Organic Synthesis Route Discovery. In *IEEE Proc. Conf. A. I. Applications* **1984**, 92−106.

(13) Kalé, L. V.; Saletore, V. A. Parallel State-Space Search for a First Solution with Consistent Linear Speedups. *Intl. J. Parallel Prog.* **1990**, *19*, 251−293.

(14) Cook, D. J. A Hybrid Approach to Improving the Performance of Parallel Search. In *Parallel Processing for Artificial Intelligence 3*; Geller, J., Kitano, H., Suttner, C. B., Eds.; Elsevier: Amsterdam, 1996.

(15) Evett, M.; Hendler, J.; Mahanti, A.; Nau, D. PRA*: Massively Parallel Heuristic Search. *J. Parallel Dist. Comput.* **1995**, *25*, 133−143.

(16) Harvey, W.; Kalp, D.; Tambe, M.; McKeown, D.; Newell, A. The Effectiveness of Task-Level Parallelism for Production Systems. *J. Parallel Dist. Comput.* **1991**, *13*, 395−411.

(17) Mahanti, A.; Daniels, C. J. A SIMD Approach to Parallel Heuristic Search. *Artif. Intell.* **1993**, *60*, 243−282.

(18) Segre, A. M.; Sturgill, D. B. Using Hundreds of Workstations to Solve First-Order Logic Programs. In *Proc. 12th Nat. Conf. A. I.* **1994**, 187−192.

(19) Saletore, V. A.; Kalé, L. V. Obtaining First Solutions Faster in AND-OR Parallel Execution of Logic Programs. In *N. A. Conf. Logic Prog.* **1989**, 390−406.

(20) Linda's tuple space is not a FIFO data structure. A counter tuple is needed to ensure that the master retrieves the expansion tuples in the same order as they were placed into tuple space. Such an ordering is not placed on the selected compound tuples, so retrieval of the next available task by the waiting workers is nondeterministic.

(21) SYNCHEM's heuristic predictor function, like most predictors for complex, real-world problems, is not *admissible*. It has proven to be effective in guiding the search, but it is not guaranteed to never underestimate the actual merit of a pathway and hence may choose to develop a path other than the actual best path.

(22) The reasons are varied. Most of the reactions in SYNCHEM's knowledge base were entered by chemists, and it is notoriously difficult to elicit negative information from experts (that is, the conditions under which a reaction will *not* work as opposed to when it will work). Some constraints require complex expressions (for instance, those involving regioselectivity), which may not be comprehensive nor conclusive. Also, there may be constraints that cannot be expressed in SYNCHEM's reaction description language.

(23) If the heuristic guidance function always led directly to the best solution, there would be no need to enhance the performance of the system by parallelizing the search engine.

CI970115V