Comparison of Different Approaches for Solving Distributed Constraint Satisfaction Problems

Q. Y. Luo *P. G. Hendry* *J. T. Buchanan*
Department of Computer Science
University of Strathclyde
Glasgow G1 1XH, UK

Abstract

A large number of problems in AI and other areas of computer science can be viewed as special cases of the constraint satisfaction problem (CSP). Various distributed or parallel computing approaches have been used to solve these problems. Mainly, these approaches can be classified as variable-based, domain-based, and function-based distributed problem solving strategies. Only some of the strategies used are scalable to large parallel machines while others are suitable only for small distributed platforms. The different approaches are presented and discussed.

Key words: Distributed Constraint Satisfaction Problems, Parallel Search Algorithms

1 INTRODUCTION

A large number of problems in AI and other areas of computer science can be viewed as special cases of the constraint satisfaction problem. Some examples [Kumar92] are machine vision, belief maintenance, scheduling, temporal reasoning, graph problems, floor plan design, the planning of genetic experiments, and satisfiability.

CSPs have three basic components: *variables, values* and *constraints.* The goal is to find an assignment of values to variables, from their separate domains, such that all the constraints are satisfied. A CSP is defined by a set of variables $\{ V_1, V_2, ..., V_n \}$, where each variable V_i has a domain D_i (usually finite), and a set of constraint relations R_i.

Many sequential algorithms for solving CSPs have been developed including backtracking (BT) ([Golomb65], [Bitner75]), backmarking (BM) [Gaschnig77], forward checking (FC) [Haralick80], backjumping (BJ) [Gaschnig78], graph-based backjumping (GBJ) [Dechter88], hybrid algorithms such as checking forwards while backjumping (FC-CBJ) [Prosser91] and extensions to these hybrid algorithms [Luo92b].

However, in the real world, we may have to deal with distributed CSPs. There are two main reasons to address distributed CSPs:

- CSPs themselves are logically or geographically distributed in a number of cases. These problems may best be solved by a multi-processor platform.
- Parallel or distributed computers may provide more computing power if used effectively which is important when considering the amount of computation required to solve CSPs.

A distributed constraint satisfaction problem (DCSP) is defined as a CSP in which multiple agents (software processes) are involved. Normally each (problem solving) agent is in charge of one or more variables, one part of the search tasks (functions), or one part of the whole search space (domains), although other arrangements are possible.

DCSPs are important sub-problems in distributed artificial intelligence (DAI). As described by [Yokoo90], various DAI problems can be formalised as DCSPs, and in turn DCSPs can provide a formal framework for studying various DAI methods. Various distributed approaches have been used to solve these problems. Mainly, these approaches can be classified as variable-based, domain-based, and function-based distributed problem solving (DPS) strategies.

A parameter which is rarely mentioned when presenting algorithms for DCSPs is *scalability.* The following sections describe algorithms developed for DCSPs and addresses the problem of scaling them to use large, highly parallel machines.

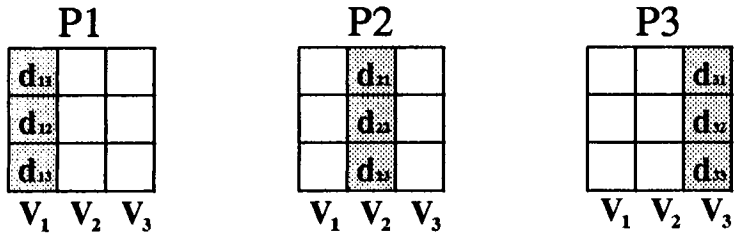

Figure 1: Variable-based DPS example for the 3-queens problem

2 VARIOUS DPS STRATEGIES

2.1 Elements of Distributed Problem Solving

There are four basic elements in DPS for DCSPs:

- Control structure: centralised or decentralised

  When using centralised control with multiple processes a much higher level of control is possible because of the global knowledge available to the problem solving processes (agents). This global knowledge may be something as simple as an agent ordering which gives agents a relative position within a global order. With decentralised control no such global order is available so the level of control is lower. If a conflict arises during search then a control mechanism must be used which does not require global knowledge. This may be a form of negotiation which attempts to resolve the conflict with only local knowledge.

- Search space types: shared or separated

  If the search space is *shared* by different processors then each processor may be in charge of one or more variables (as in [Luo92b]) and will try to assign values which do not conflict with the variables of other processors. Decisions on one processor are affected by the current state of other processors and so communication is essential in this case.

  In a *separated* search space each processor is in charge of a separate area of the search space (each has a unique branch of the search tree to traverse). In this case the agents may proceed autonomously and need only communicate when a solution is found or the search space is exhausted. There is therefore very little need for communication.

  The type of search space varies dramatically from problem to problem. Some problems which are naturally or logically distributed can be easily mapped to a multi-processor platform using a shared search space. Because the problem is naturally distributed, there will be little communication needed to solve it. Using this method on a closely coupled problem (with a high interconnection between agents) will result in a large amount of communication.

- Communication: message-passing or shared memory

  The method of communication may be shared global memory, message-passing or a combination of the two. The cost of communication is dependent on the type of problem, the control method used (centralised or decentralised) and search space type chosen. For centralised control the cost of communication is lower because of the use of shared global data to pass information between agents. Also, when the search space is divided with the sub search spaces being disconnected there may be no need for communication at all.

- Detecting termination

  A problem in many distributed algorithms is determining when the algorithm has reached its goal state. In the case of DCSP algorithms this is when the algorithm has either found a solution or has exhausted the search space entirely. When the search tree is divided amongst processors it is straight-forward to detect when a solution is found as, within one processor, a sequential CSP algorithm is being used. The processor will detect a solution and can inform the other processors of this.

  However, when using a distributed algorithm where each processor is in charge of several variables, it becomes a problem to detect a solution. When all variables satisfy all constraints there will be no search activity within or between processors. This leads into another area of research for detecting given states in a distributed environment. Termination detection algorithms must be used [Raynal88]. This increases the overheads of using a shared search space. Detecting that the search space is exhausted with both types of search space is trivial.

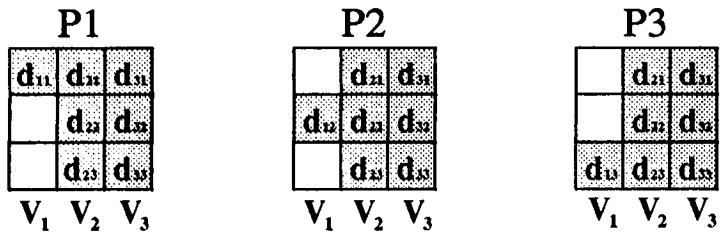

Figure 2: Domain-based DPS example for the 3-queens problem

2.2 Variable-based DPS

In the variable-based DPS strategy (Figure 1), the problem is distributed, based on the variables. Each (problem solving) agent is in charge of one or several variables and their domains. A variable is controlled by only one agent which attempts to assign it a legal value from its domain. In this case the search space is shared between agents and the action of one agent will affect other agents directly or indirectly because of the constraints acting between variables. An agent first solves its local CSP (between the variables which it controls) and then communicates its partial results with other agents sharing constraints. Using this method, illegal values may be removed from variable domains and a solution arrived at if one exists.

Difficulties arise because of the asynchronous behaviour of agents. It is necessary to introduce mechanisms to prevent agents from being distracted by out-of-date or inaccurate partial results from other agents. Other problems inherent in variable based DPS include: distributed memory (no shared global data), infinite processing loops and communication overheads (which can be very high in many cases).

Variable-based algorithms may have:

- **centralised control** via a total (linear) order.

  An order is imposed between agents where each agent knows its own place and also the order of all other agents. When a conflict arises during search an agent uses this order to decide what is causing the conflict and to decide where it should direct its efforts to resolve the conflict. The order can also be utilised during search to decide how an agent should behave towards other agents. For example, in [Yokoo90] and [Luo92a] an agent only sends the new value of a variable to agents with a higher order which then act as constraint evaluators.

- **decentralised control.**

  In this case there is no global information shared by agents. They do not have any information about other agents that gives them a position of reference. When a conflict arises during search some form of conflict resolution must be applied. This normally attempts to solve the conflict locally and so does not require gathering much knowledge or global synchronisation. The local solution to the conflict may in turn cause other conflicts among different agents which will again be resolved locally. In this way, conflict resolution propagates its local solutions on a global scale without requiring global information. Local conflicts may use a form of negotiation or they may, for example, impose a temporary local order for each conflict.

2.3 Domain-based DPS

In the domain-based DPS strategy (Figure 2), the problem is distributed based on the domains of some variables (one variable in figure 2 but each resultant search space could be further broken down). From figure 2, processor 1 (P1) traverses the search space rooted on $V_1 = 1$, P2 on $V_1 = 2$ and P3 on $V_1 = 3$. Each sub-search space involves all variables and each is independent of the other search spaces. Each processor is therefore solving a unique CSP problem and no communication is necessary.

Within each processor a sequential CSP algorithm is used directly without modification for the DCSP. This means that domain-based methods can draw directly on the research applied to sequential CSP algorithms. Also, any future advances can be applied directly to improve algorithm performance.

An advantage of domain-based DPS is its improvement in time to solution over sequential algorithms. In variable-based DPS it is quite common, because of the amount of (time consuming and distracting) communication, that the distributed algorithm will require more time than a sequential algorithm applied to the same problem. This is not the case with domain-based DPS. If both the distributed and sequential methods use the same order of instantiation (the order in which variables are assigned values) then it can be seen that the worst performance of the domain-based DPS algorithm will be the same as the performance of the sequential

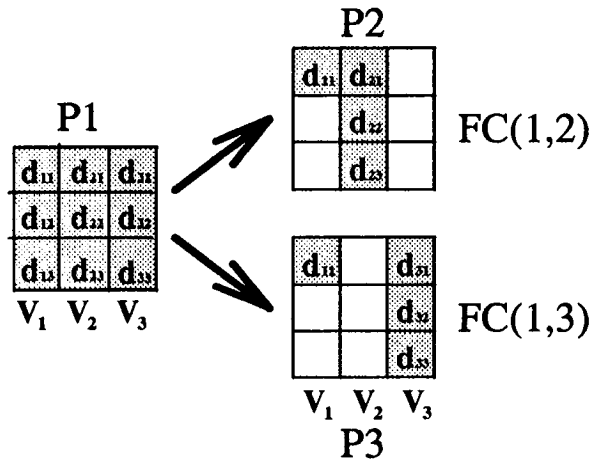

Figure 3: Function-based DPS example for the 3-queens problem

algorithm (i.e. the domain-based algorithm will perform better than or the same as the sequential algorithm) plus a small overhead to divide the search space. This is because one of the processors will follow the path in the search space taken by the sequential algorithm.

Domain-based DPS also provides a good basis for research into issues such as search space type, conflict sharing, effects of instantiation order, heuristic methods, etc. For example, once the search space is divided each processor could then use a different sequential algorithm or a different instantiation order. Different processors could also be given the same sub-search space (i.e. the same initial state) but could then change the instantiation order so that in effect each would change the path of search and hopefully find a solution sooner.

2.4 Function-based DPS

Function-based DPS attempts to utilise parallelism by using spare processors to perform repeated tasks. For example, if search is taking place on one processor using the forward checking (FC) sequential algorithm then spare processors could be used to perform the actual forward checking (the domain filtering) in parallel (figure 3). This will, of course, incur some overheads starting new processes but these will be small if the problem space is large. This method is only suitable for shared-memory machines where the information in a parent process can be seen and manipulated by its children.

Function-based DPS could also be used in conjunction with domain-based DPS where several processors are allocated for domain-based DPS and the remaining (spare) processors could then be used for function-based DPS. If each process were to run FC on its part of the search space it could then use the spare processors to perform domain filtering in parallel. This would appear to be most useful near the root of the search tree where a lot of domain filtering may be required, especially in a large problem space.

3 THE ALGORITHMS

3.1 Variable-based Algorithms

For the variable-based algorithms, the most trivial strategy may be a centralised backtracking method where, when a conflict occurs, one leader agent gathers all the information from the agents and attempts to solve the conflict. This approach is wasteful, both in terms of communication overheads and in loss of parallelism. Another strategy is synchronous backtracking where the agents share a global instantiation order and act synchronously. At one time, only one agent is active. Obviously, this approach does not exploit parallelism.

3.1.1 Asynchronous Backtracking

An asynchronous backtracking method to address DCSPs was introduced in [Yokoo90]. It performs the distributed search concurrently. It manages asynchronous change by a method called *context attachment* (referred to as *agent view*) where an agent must include its current beliefs about other agents when sending a conflict related message (referred to as a *nogood* message). To avoid infinite processing loops [Yokoo90] used a *total* (linear) *order* amongst agents. The use of a total order unifies the search space even though the algorithm

is on a distributed platform. The total order however causes problems with load balancing and with message passing volume. [Yokoo90] failed to address either of these problems.

[Luo92a] presents an algorithm (PBM-GBJ) which uses the *context attachment* and *total order* ideas of [Yokoo90]. The algorithm addresses methods of reducing the amount of communication required and ways in which redundant computation can be avoided. A simple load balancing strategy is also suggested but is not sufficient to solve the imbalance caused by using the total order.

To avoid unnecessary search paths and consistency checks [Luo92a] uses distributed extended forward checking which records nogood values (values which cause conflict) and uses them during search. It also employs a conflict-directed backjump method [Prosser91] when resolving conflicts which attempts to find the root cause of any conflict.

An important part of the algorithm of [Luo92a] is its ability to cope with dynamic changes within a DCSP. Many real-world problems are dynamic and it is desirable not to lose valuable information or, even worse, to have to start search again when the problem changes slightly.

The previous two papers made the simplification of having one agent in charge of only one variable. This leads to a greater amount of communication as there are no local CSPs to be solved and the ratio of compute:communication time decreases. A more realistic scenario is used in [Luo92c] where each agent is in charge of several variables. For real world problems this represents each agent first attempting to solve its own problem and then exchanging partial results until a solution is found. The more independent the sub-problems are of each other the less communication is required to maintain an up-to-date view within each agent (as there will be less inter-agent constraints to be satisfied). [Luo92c] shows that a mapping mechanism of several variables per agent means that both distributed search algorithms (between agents) and sequential search algorithms (within each agent) must be used. In turn this allows heuristic methods (methods which attempt to guide search towards a solution) to be used locally which have already been developed for sequential algorithms, such as the search rearrangement method of [Bitner75]. If each agent has its own local constraints and global constraints, it may be wiser to search the variables with more global constraints first, in order to make global backtracking occur earlier.

In [Nishibe92], the heuristic method of dynamic variable ordering was applied to the hill climbing method to address DCSPs. However, the hill climbing methods may fail to find a solution because no backtracking is performed. Both [Luo92a] and [Nishibe92] noticed that a (static) variable ordering does have a marked impact on the performance of distributed asynchronous backtracking.

Global heuristics for distributed search algorithms are not yet established and developing these will not be straight-forward because of the lack of shared global information between agents.

A final point about variable-based DPS algorithms is the problems presented by using a total order. The total order is a very attractive method to control both search and conflict resolution because of the ease of implementing algorithms around a centralising control structure and because of its similarities to the instantiation order of the sequential algorithms. However, it also introduces several problems of its own:

- **Communication Overheads**

  To use the total order it is necessary to have rules about which agents send messages to which other agents. In the previous algorithms mentioned, agents only send partial results to other agents sharing constraints and with a higher order than the sending agent (to prevent infinite processing loops). This, of course, means that the higher order agents are receiving a lot more partial results than lower order agents. Some high order agents may become communication bottlenecks in this case.

- **Load Balancing**

  Because the higher order agents receive more partial results, they must do more computation in evaluating new information and in updating their own state. This leads to a load imbalance with the higher agents tending to do most of the work while the lower agents spend much of the time idle.

- **Problems applying global heuristic methods**

  The fixed global order and distribution of variables makes it very difficult to apply heuristic methods. In some cases where heuristics can be applied, synchronisation may be needed between agents (for example to change the agent or variable order) which is very undesirable.

3.1.2 Negotiation

A variable-based DPS which does not have a centralising control strategy (such as the total order) requires some other control method which does not require global knowledge. Some research has put forward theoretical

models such as in [Zlotkin91] which address the problems of conflict resolution. This involves an agent using many heuristics to decide on its roll when involved with other agents.

Another method which makes use of the aforementioned research using a total order among agents is to use a partial order when conflicts arise. If a conflict occurs, the agent discovering it evaluates which other agents it believes are involved in the conflict. It can then impose a partial (or local) order amongst only those agents involved. This order is only valid for one conflict and an agent may be involved in several conflicts at the same time and so have several different orders of its own. Once a conflict is resolved the order is forgotten.

This local resolution strategy has a global effect because of propagation. One conflict resolution may lead to another and so on until the effect of one change affects many or all of the agents not involved directly in the conflict.

Control mechanisms within such a negotiation strategy are complex. The total order removed problems of *infinite processing loop* and context attachment eased problems of *asynchronous changes*. However, using negotiation it is necessary to guard against these problems with other methods. These methods are complex because of the lack of global knowledge and introduce a large overhead in problem solving.

Disadvantages of using a negotiation strategy include:

- much communication required to disseminate partial results.
- no global reference for applying search heuristic methods.
- may require synchronisation of agents during conflict resolution.
- large overheads to prevent loops and manage asynchronous changes.
- require methods to undo work based on out-of-date information.
- very difficult to find all solutions to a problem.

Negotiation also suffers from not being scalable to large platforms. When a large number of processors are used, message volume will quickly swamp the system, and it would take too long for effects of conflict resolution to propagate to other agents.

3.2 Domain-based Algorithms

[Burg90] divides the search space between a set of processors each executing a sequential CSP algorithm. In this case forward checking (FC) is used although any other sequential algorithm could replace it. This method attempts to use the power of parallelism directly, having each processor kept busy but at the same time not allowing the possibility of two processors performing the same work since each is in a separate part of the search space.

Redundant work may however still be done because one processor does not make use of the search information obtained by another during search. For example, if one processor finds that a value d_{ij} cannot appear in any solution then this information may be useful for other processors to avoid redundant work by pruning any sub-tree rooted on d_{ij}.

[Burg90] takes the extreme stance of having no communication of search information between processes. It seems that [Burg90] and [Yokoo90] are two extremes. [Burg90] does not use communication to aid search whereas [Yokoo90] relies heavily on communication of partial results at a heavy cost to performance. [Luo92c] tries to find a middle ground where the parallelism of [Burg90] is used along with a limited sharing of partial results to prune the search space.

[Luo92c] uses the nogood mechanism reported in [Dechter88] and [Luo92a] to share information between search processes. A *nogood* contains a (*variable*, *value*) pair which is disallowed due to conflict. The reason for the conflict is also stored so that, if this reason no longer holds (i.e. the search state which caused the conflict changes) the nogood value can be released. If there is no change in the conflicting variables then the nogood is not released. This allows the search space to be pruned. When there is no conflict variable left in a nogood, its value will be not released. This special kind of nogood is called dead-end.

In [Luo92c] agents share these dead-ends and make decisions based on the current dead-end information. When an agent receives a dead-end from another agent, it will prune the disallowed value from its domain to reduce search.

[Burg90] also addresses load balancing for domain-based DPS. When a processor exhausts its search space it sends a request to another processor for some of its search space. The other processor splits its remaining

search space between the two and search continues. This method uses several simple heuristics to aid in splitting the search space and again requires very little communication.

The main advantages of the domain-based DPS are:

- If the problem is not naturally distributed, it can still be addressed by domain-based DPS. Solving the same problem using variable-based DPS may cause serious communication bottlenecks and load balance problems.
- It can directly use any sequential CSP algorithms.
- It needs little communication.
- It can use established global heuristic search strategies.

However, domain-based DPS may have difficulty when a problem is over-constrained as it is difficult to establish a cause due to a lack of logical links among different search spaces. After all agents fail to find any solution, the algorithm may need to check all conflicts again to find the real cause of the failure. This approach is not suitable for problems which are geographically distributed.

A major plus for this method is its scalability to very large multiprocessor networks. There is little or no communication to worry about and each processor can execute the same algorithm. There is also no global control or synchronisation and the setup cost of the initial propagation of information is small. This is important for DCSPs because of the vast search spaces in larger problems. At the current technology level a single processor could take years to find a solution to a large CSP problem.

3.3 Function-based Algorithms

In function-based DPS the problem solving functions are distributed among processors by a single search process. The strategy is easily realised on a shared memory machine where each function can manipulate the global data directly.

[Luo92c] presents a function-based DPS algorithm. The domain filtering of FC is performed in parallel. If a domain becomes empty then the variable will change its value or backtrack without waiting for other forward checks to finish.

The main problem with this approach is the overheads involved in starting processes to perform the domain filtering. If the problem size is large requiring a lot of filtering then this overhead is small.

4 COMPUTATIONAL RESULTS

This section presents results obtained by implementing these DCSP algorithms on a SEQUENT SYMMETRY using the C programming language. The SEQUENT SYMMETRY is a shared memory multi-processor platform.

Two sets of results are presented. The first for the Zebra problem. This is described in [Dechter88] and [Prosser91] and is composed of 25 variables which correspond to five groups (each containing five variables) in which the variables have constraint relations between each other and additionally there are a number of constraint relations between the variables in different groups. The order of the variables is important to the efficiency of search. For this reason, some of the results which follow are the average values taken over all ordering permutations of the groups ($5! = 120$ permutations in total). The order within each group was fixed. In this paper, we use [Prosser91]'s definition of the Zebra which has 11 solutions. The density ρ[1] of the Zebra problem is 0.20.

The second problem is the well-known n-queen problem. This is the problem of finding a way to place n queens on an $n * n$ chess board so that no two queens attack each other. Although this is quite an "artificial" problem, it represents a 'worst case' scenario where every variable has a constraint with all other variables. The density ρ of the n-queen problems is 1.0.

The computational results are obtained mainly from the various distributed or sequential algorithms [Luo92a] [Luo92b] [Luo92c] based on the algorithm forward-checking while conflict-directed backjump (FC-CBJ) supported by nogoods (NG) and the postponed revision (PR) mechanism (except that PBM-GBJ [Luo92a] does not support postponed revision). The algorithm extensions used are: HY for a hybrid using both a distributed search algorithm and a sequential search algorithm in variable-based algorithms, SN for

[1] Density $\rho = \frac{c}{n(n-1)}$, where c is the number of constraints within the problem and n is the number of variables.

sharing nogood among different processors in a domain-based search strategy and DO for dynamic variable ordering (in variable-based algorithms, dynamic variable ordering is only performed locally). All domain-based algorithms are supported by a dynamic load balancing mechanism. In Tables 1 and 2 all the values (except time) are *averages* over 120 runs of the algorithms, and the time includes all overheads such as the initial propagation of constraints. These times were obtained using the *time* system call under UNIX in the unit of seconds.

In the tables the abbreviation ALG is used to denote the base algorithm FC-CBJ-NG-PR.

Table 1 Comparison of Variable-based DPS, Domain-based DPS and Sequential Algorithms for the Zebra to find the first solution against 120 orders

	Processor	Check	Backtrack	Message	Time[s]
Sequential ALGs					
ALG	1	1636	128	n/a	108
ALG-DO	1	268	0	n/a	17
Variable-based ALGs					
PBM-GBJ	3	3356	555	2859	379
HY-ALG	3	2036	155	363	154
HY-ALG-DO	3	1022	49	154	74
Domain-based ALGs					
ALG	3	1925	118	n/a	66
ALG-SN	3	1848	117	n/a	58
ALG-DO	3	763	15	n/a	30

Table 2 Comparison of Variable-based DPS, Domain-based DPS and Sequential Algorithms for the Zebra to find all solutions against 120 orders

	Processor	Check	Backtrack	Message	Time[s]
Sequential ALGs					
ALG	1	7780	830	n/a	579
ALG-DO	1	2322	270	n/a	207
Variable-based ALGs					
HY-ALG	3	10994	1113	2090	692
HY-ALG-DO	3	8942	801	1542	549
Domain-based ALGs					
ALG	3	2867	294	n/a	298
ALG-SN	3	2617	284	n/a	251
ALG-DO	3	2903	310	n/a	160

From the Table 1 and Table 2, we may make the following observations:

- In the variable-based approach, HY-ALG may out perform PBM-GBJ. The main reason may be that the hybrid performs a more efficient search within each processor and therefore requires less checks, backtracks and messages.
- In the variable-based algorithms, HY-ALG-DO may run faster than HY-ALG alone. This shows that, even when we only apply local dynamic variable ordering (limited within each processor), it does help the performance.
- In a domain-based approach, ALG-SN may be better than ALG alone. This shows that it may be useful to share nogoods among processors.
- Domain-based algorithms are generally faster than variable-based algorithms for solving the Zebra problem in either finding the first solution or all solutions. A reason for this may be that the Zebra problem is not naturally distributed. When the variable-based algorithms are used to solve it, the problem may cause serious communication bottlenecks and load imbalance. Furthermore, due to a lack of global

knowledge and no global heuristic methods, it may cause more checks, backtracks and delayed backtracks to reach a solution. All of these take time. Detecting termination of variable-based algorithms may also add some overhead.

- To find the first solution to the Zebra problem, neither variable-based algorithms nor domain-based algorithms perform better than their sequential copartners. The domain-based algorithms require time to propagate constraints among processors (this is an overhead which cannot be avoided but which would become insignificant for larger search spaces). Variable-based algorithms simply perform more checks, backtracks and delayed backtracks and spend a lot of time communicating.

- To find all solutions, domain-based algorithms perform better than their variable-based and sequential equivalents. The reason for this is that a domain based algorithm exhausts the search space using several processors, each in a different part of the search tree, whereas the sequential algorithm must exhaust the search space alone. Domain-based algorithms also use little communication and use the parallel computing power in a balanced way.

- Although the variable-based algorithms perform poorly compared to other algorithms for the Zebra problem, that is not to say that there is no use for variable-based DPS. For some geographically distributed problems, variable-based algorithms may be the only way to address them. It is clear that simply applying more processors to a given problem will not necessarily improve the solution to that problem. There are several reasons for this such as their inability to use global heuristics, lack of knowledge, communication overheads and load imbalance.

Table 3 Comparison of Function-based DPS and Sequential Algorithms for the 25 queens problem to find the first solution

	Processor	Check	Backtrack	Message	Time[s]
Sequential ALG ALG-DO	1	9246	184	n/a	3.96
Function-based ALG ALG-DO	3	9408	185	n/a	2.89

From Table 3, there is little difference in checks and backtracks performed by the sequential and function-based algorithms. However, the latter may perform better due to parallel domain filtering.

Initial tests using a negotiation strategy to control conflict resolution showed that the extra message passing incurred, and overheads from mechanisms to prevent problems such as processing loops, caused the algorithm to take considerably longer to find a solution than other variable-based DPS methods. In fact, for the 120 orders of the zebra problem the negotiation algorithm was between 8 and 12 times slower than PBM-GBJ alone. It is also very difficult when not using some global knowledge to have an algorithm find all solutions to a problem.

5 CONCLUSION

Table 4 sums up the methods described in this paper.

From the table it can be seen that Domain based DPS is very versatile. However it suffers from not coping easily with dynamic changes to the problem. This is quite important as many real world problems naturally require dynamic change.

Variable based DPS algorithms are generally slower than other methods but, if given the correct problem type, may perform better. For some problems they are the only algorithms which could be used.

Further work is required in many areas of the above algorithms to allow them to fully exploit the power of parallel and distributed computing.

Acknowledgement: This work is partially supported by the U.K. Science and Engineering Research Council (GR/F82733).

Table 4 Characteristics of Basic DPS Strategies

	Variable-based		Domain-based	Function-based
algorithms	specially designed		any sequential	any sequential
Control Type	central	negotiation	n/a	n/a
Control Level	high	lower	n/a	n/a
Communication Cost	medium	high	lower	n/a
Memory type	both	both	both	shared
Heuristic scope	local	local	global	global
Search Space Type	shared	shared	separated	single
Dynamic Change	easy	easy	difficult	easy
Termination Detection	difficult	difficult	easy	easy
Scalability	poor	poor	good	reasonable

References

[Bitner75] J. R. Bitner and E. Reingold *Backtrack programming techniques* CACM, vol 18, 1975, pp651-656

[Burg90] Bernard Burg *Parallel Forward Checking Parts 1 and 2* Technical Reports TR-594, TR-595. Institute for New Generation Computer Technology, Japan, 1990

[Dechter86] Rina Dechter *Learning While Search In Constraint Satisfaction Problems* AAAI-86, vol. 1, August 1986, pp178-183

[Dechter88] Rina Dechter *Constraint Processing Incorporating, Backjumping, Learning, and Custer-Decomposition* CAIA-88, March 1988, pp312-319

[Gaschnig77] J. Gaschnig *A General Backtracking Algorithm that Eliminates Most Redundant Tests* Proceeding of the International Joint Conference on Artificial Intelligence, 1977

[Gaschnig78] J. Gaschnig *Experimental case studies of Backtrack vs Waltz-type vs. new algorithms for satisficing assignment problems* Proceeding 2-ed Biennial Conference Canadian Society for Computational Studies of Intelligence. Toronto, July 1978

[Golomb65] S. W. Golomb and L. D. Baumert *Backtrack Programming* JACM vol 12, 1965, pp516-524

[Haralick80] R. M. Haralick and G. L. Elliot *Increasing Tree Search Efficiency for Constraint Satisfaction Problems* Artificial Intelligence, vol 14, 1980, pp263-314

[Kumar92] V Kumar *Algorithms for Constraint-Satisfaction Problems: A Survey* AI Magazine, Spring 1992, pp32-44

[Luo92a] Q. Y. Luo, P. G. Hendry and J. T. Buchanan *A Hybrid Algorithm for Distributed Constraint Satisfaction Problems* Proc. EWPC-92, Barcelona, Spain, 1992

[Luo92b] Q. Y. Luo, P. G. Hendry and J. T. Buchanan *Extending Algorithms for Constraint Satisfaction Problems* Dept of Computer Science, University of Strathclyde, KEG-5-92, June 1992

[Luo92c] Q. Y. Luo, P. G. Hendry and J. T. Buchanan *Heuristic Search For Distributed Constraint Satisfaction Problems* Dept of Computer Science, University of Strathclyde, KEG-6-92, October 1992

[Nishibe92] Y Nishibe, K Kuwabara and T Ishida *Effects of Heuristics in Distributed Constraint Satisfaction: Towards Satisficing Algorithms* The 11th International Workshop for DAI, 1992, p18

[Prosser91] Patrick Prosser *Hybrid Algorithms for the Constraint Satisfaction Problem* Technical Report AISL-46-91, Computer Science Dept, University of Strathclyde, Scotland, 1991

[Raynal88] M. Raynal *Distributed Algorithms and Protocols* John Wiley and Sons, 1988, p163

[Yokoo90] Makoto Yokoo, Toru Ishida and K. Kuwabara. *Distributed Constraint Satisfaction for DAI Problems* The 10th International Workshop for DAI, Oct 1990, p20

[Zlotkin91] 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, Vol. 21, No. 6 November/December 1991