Boosting Distributed Constraint Satisfaction

Georg Ringwelski[1,⋆] and Youssef Hamadi[2]

[1] 4C, University College Cork, Ireland
g.ringwelski@4c.ucc.ie
[2] Microsoft Research, 7 J J Thomson Avenue,
Cambridge CB3 0FB, United Kingdom
youssefh@microsoft.com

Abstract. Competition and cooperation can boost the performance of search. Both can be implemented with a portfolio of algorithms which run in parallel, give hints to each other and compete for being the first to finish and deliver the solution. In this paper we present a new generic framework for the application of algorithms for distributed constraint satisfaction which makes use of both cooperation and competition. This framework improves the performance of two different standard algorithms by one order of magnitude and can reduce the risk of poor performance by up to three orders of magnitude. Moreover it greatly reduces the classical idleness flaw usually observed in distributed hierarchy-based searches. We expect our new methods to be similarly beneficial for any tree-based distributed search and describe ways on how to incorporate them.

1 Introduction

In many application domains constraint-based tree-search methods are the technology of choice to solve NP-complete problems today. However, when actually applying the algorithms without further customization, we have often experienced unacceptable performance. This results from various well-investigated factors including bad modelling and the choice of a wrong labelling strategy. The solution for bad modelling often resides in a good understanding of the constraint processing which results in the application of well known modelling patterns (channeling constraints, redundant modelling, etc). Finding a good labelling strategy is not obvious and usually requires long and expensive preliminary experiments on a set of realistic problem instances. Performing those experiments or defining realistic input samples is far from being simple for today's large scale real life applications. Ideally we would not have to make a choice for a labelling strategy at all and rather be able to use an algorithm "out-of-the-box" which finds the best strategy itself [Pug04].

The previous observations are emphasized in the processing of distributed constraint satisfaction problems (DisCSPs). Indeed, the distributed nature of

⋆ This work has received support from the Embark Initiative of the Irish Research Council of Science Engineering and Technology under Grant PD2002/21. We'd like to thank Rick Wallace and Mark Hennesy of 4C for providing the problems.

P. van Beek (Ed.): CP 2005, LNCS 3709, pp. 549–562, 2005.

those problems makes any preliminary experimental step difficult since constrained problems usually emerge from the interaction of independent and disconnected agents transiently agreeing to look after a set of globally consistent local solutions [FM02].

This work targets on those cases where bad performance in DisCSP can be prevented by choosing a good labelling strategy and executing it in a benefiting order within the agents. In this paper we define a notion for the risk we have to face when choosing an agent-ordering and present the new "M-" framework[1] for the execution of distributed search. An M- portfolio executes several distributed search strategies in parallel and let them compete and cooperate for being the first to finish. We apply the framework in two case studies where we define the algorithms "M-ABT" and "M-IDIBT" which improve their counterparts ABT [YDIK92] and IDIBT [Ham02] significantly. With these case studies we can show the benefit of competition and cooperation for the underlying distributed search algorithms. We expect the "M-" framework to be similarily beneficial for other tree-based DisCSP algorithms. Cooperation of distributed searches is implemented with the aggregation of knowledge within agents and thus yields no extra communication. The knowledge gained from *all* the parallel searches is used by the agents for their local decision making in each single search. We present two principles of aggregation and employ them in methods which are applicable to the limited scope of the agents in DisCSP.

In the next section we define the risks we have to face in search. This can be used as another metric (besides performance) to evaluate algorithms. In Section 3 we present the new "M-" framework. Section 4 describes our case studies and Section 5 their empirical evaluation. Then we discuss related work, summarize the results and outline some ideas for future work.

2 Risks in Search

Here we present two definitions of *risk* is search. The first notion called *randomization risk* is related to the changes in performances when the same algorithm is applied multiple times to a single problem instance. The second notion called *selection risk* represents the risk of selecting the wrong algorithm, i.e., the one which performs poorly on the considered instance.

2.1 Randomization Risk

In [GS01] "risk" is defined as the standard deviation of the performance of one algorithm applied to one problem multiple times. This risk increases when more randomness is used in the algorithms.

Definition 1. *The R-Risk is the standard deviation of the performance of one algorithm applied multiply to one problem.*

[1] M stands for Multi-Directional. "M-" searches in multiple directions, namely agent topologies, at the same time.

Reducing the R-Risk leads in many cases to trade-offs in performance [GSK98], such that the reduction of this risk is in general not desirable. For instance, we would in most cases rather wait between 1–10 seconds for a solution than waiting 7–8 seconds. In the latter case the risk is lower but we do not have the chance to get the best performance.

In asynchronous and distributed systems we are not able to eliminate randomness at all. Besides intended randomness (e.g. in value selection functions) it emerges from external factors including the CPU scheduling to agents or unpredictable times for message passing [ZM03].

To get a standpoint of the R-Risk in DisCSP we made a preliminary experiment, where randomness emerged from distribution only. We solved binary DisCSPs with the IDIBT and ABT algorithms with random message delays and unpredictable agent-activation. It turned out that the R-Risk is in general very high (compared to monolithic systems). Even with completely deterministic value-selection functions the performance of different runs of the algorithm on the same problem differed significantly. For instance, the ABT algorithm with lexicographic labelling applied 100 times to the 10-queens problem could find one solution in 297–5374 ms while IDIBT applied 100 times took 1640–1984 ms. The R-Risk resulting exclusively from distribution was 807 for ABT and 96 for IDIBT.

2.2 Selection Risk

The risk we take when we select a certain algorithm or a heuristic to be applied within an algorithm to solve a problem will always be that this is the wrong choice. For most problems we do not know in advance, which the best algorithm or heuristic will be and may select one which performs much worse than others. We'll refer to this risk as to the Selection-Risk (S-Risk).

Definition 2. *The S-Risk of a set of algorithms A is the standard deviation of the performance of each $a \in A$ applied the same number of times to one problem.*

We investigated the S-Risk emerging from the chosen agent ordering in IDIBT in a preliminary experiment on small, fairly hard random problems (15 variables, 5 values, density 0.3, tightness 0.4). We used one variable per agent and could thus implement variable-orderings in the ordering of agents. We used lexicographic value selection and four different static variable-ordering heuristics: a well-known "intelligent" heuristic, its inverse (which should be bad) and two different blind heuristics. As expected, we could observe that the intelligent heuristic dominates in average but that it is not always the best. It was the fastest in 59% of the tests, but it was also the slowest in 5% of the experiments. The second best heuristic (best in 18%) was also the second worst (also 18%). The "anti-intelligent" heuristic turned out to be the best of the four in 7% after all. The differences between the performances were quite significant with a factor of up to 5. Applied to the same problems, ABT gave very similar results with a larger performance range of up to factor 40.

3 Multi-directional Distributed Search

By a direction in search we refer to a variable ordering. In this paper we consider only static orderings but the "M-" framework can be used with dynamic orderings as well. In DisCSP the variable ordering implies the agent topology. Assume that each agent hosts one variable, for each constraint a *directed* connection between two agents/variables is imposed. The direction defines the priority of the agents and thus in which direction backtracking is performed. In Figure 1 we show two different static agent-topologies emerging from two different variable-ordering heuristics in DisCSP.

The idea of Multi-Directional search is that several variable orderings and thus several agent topologies are used by concurrent searches. We refer to this idea as to the "M-" framework for DisCSP. Applied to an algorithm X it defines a DisCSP algorithm M-X which applies X multiply in parallel. Each search operates in its usual way on one of the previously selected topologies. In each agent the multiple searches use separate contexts to store the various pieces of information they require. These include for example adjacent agents, their current value or their beliefs about the current values of other agents. Given the topologies in Figure 1, agent X3 for example, would contain two contexts. In the one which is related to maxDegree it would store X7 as lower prioritized adjacent agent and in the other it would store X1. In ABT or IDIBT it would thus address messages that notify others of new values (*ok?* in ABT, *infoVal* in IDIBT) to agent X7 in one search effort and to X1 in the other.

In a set of such agents *different* search-efforts can be made in parallel. Each message will refer to a context and will be processed in the scope of this context. The first search to terminate will deliver the solution or report failure. Termination detection has thus to be implemented for each of the contexts separately. This does not yield any extra communication as shown for the multiple contexts of IDIBT in [Ham02].

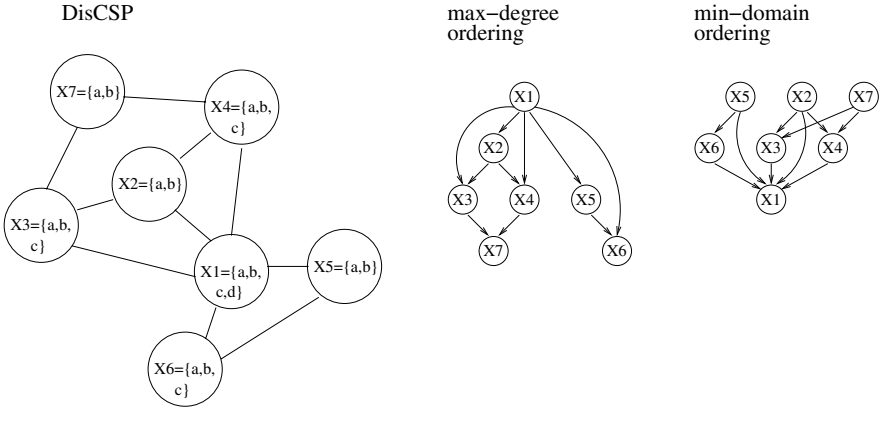

Fig. 1. DisCSP and agent topologies implied by variable orderings

One motivation for this is to reduce the S-Risk by adding more diversity to the used portfolio. Assuming we do not know anything about the quality of orderings, the chance of including a good ordering in a set of M different orderings is $|M|$-times higher than selecting it for execution in one search. When we know intelligent heuristics we should include them but the use of many of them will reduce the risk of bad performance for every single problem instance (cf. experiment in Section on S-Risk). Furthermore the expected performance is improved with the "M-" framework since always the best heuristic in the portfolio will deliver the solution or report failure. If we have a portfolio of orderings M where the expected runtime of each $m \in M$ is $t(m)$ then ideally (if no overhead emerges) the system terminates after $min(\{t(m)|m \in M\})$. The resulting trade-offs and overheads for this are investigated in this paper.

The trade-off in space is linear in the number of applied orderings. Thus, it clearly depends on the size of the data structures that need to be duplicated for the contexts. This will include only internal data structures which are related to the state of search. "M-" does not duplicate the whole agent. The data structures for communication for instance are jointly used by all the concurrent search efforts.

The trade-off in computational costs will be described in detail in the Section on the Empirical Evaluation.

3.1 Aggregation

Besides the idea of letting randomized algorithms compete to become "as good as the best" the "M-" framework can also use cooperation. With this we may be able to be even "better than the best", by accelerating the best search effort even more by providing it with useful knowledge others have found. Cooperation is implemented in the aggregation of knowledge within the agents. The agents use the information gained from one search to make better decisions (value selection) in another search. This enlarges the amount of knowledge on the basis of which local decisions are made.

In distributed search, the only information that agents can use for aggregation is their view to the global system. With multiple contexts, the agents have multiple views and thus more information available for their local reasoning. In this setting, the aggregation yields no extra communication costs. It can be performed locally and does not require any messages or blackboard-access.

In order to implement Aggregation we have to make two design decisions: first, which knowledge is used and second, how it is used. As mentioned before we use knowledge that is available for free from the internally stored data of the agents. In particular this may include:

Usage. Each agent knows the values it currently has selected in each search.
Support. Each agent can store currently known values of other agents (agent-view) and the constraints that need to be satisfied with these values.
Nogoods. Agents may store partial assignments that are found to be inconsistent.

Effort. Each agent knows for each search how much effort in terms of the number of backtracks it has already invested.

The interpretation of this knowledge can follow two orthogonal principles: **diversity** and **emulation**. Diversity implements the idea of traversing the search space in different parts simultaneously in order not to miss the part in which a solution can be found. The concept of emulation implements the idea of cooperative problem solving, where agents try to combine (partial) solutions in order to make use of work which others have already done.

With these concepts of providing and interpreting knowledge we can define the portfolio of aggregation methods shown in Table 1. In each box we provide a name (to be used in the following) and a short description of which value is preferably selected by an agent for a search.

Table 1. Methods of aggregation

	diversity	**emulation**
usage	*minUsed*: the value which is used the least in other searches	*maxUsed*: the value which is used most in other searches
support	–	*maxSupport*: the value that is most supported by constraints wrt. current agent-views
nogood	*differ*: the value which is least included in nogoods	*share:* always use nogoods of all searches
effort	*minBt*: a value which is not the current value of searches with many backtracks	*maxBt*: the current value of the search with most backtracks

4 Algorithms

As a case study to investigate the benefit of competition and cooperation in distributed search we implemented M-IDIBT and M-ABT.

M-IDIBT. This algorithm incorporates IDIBT [Ham02] in the "M-" framework. IDIBT already uses multiple contexts to perform parallel search (i.e., splitting of search tree). We use the contexts for different variable-orderings but apply each of them to the complete search tree. In order to prevent the required pre-processing of the agent topology with DisAO [Ham02] we changed the algorithm to add the required extra links between agents dynamically during search (similar to the processing of "addLink"-messages in ABT). Finally we extended the algorithm to support dynamic value selection which is essential for Aggregation.

M-ABT. This algorithm incorporates ABT [YDIK92] in the "M-" framwork. For this we implemented contexts by duplicating the local storage of current value, agent-view and nogood-store. Storing the nogood-store multiply may have large trade-offs in space, but sharing it means applying Aggregation and is thus

considered separately. In M-ABT every message carries additionally the id of its related search. No other changes were made to the original algorithm.

5 Empirical Evaluation

For the empirical evaluation of the "M-" framework we processed more than 180000 DisCSPs with M-IDIBT and M-ABT. We solved random binary problems (15 variables, 5 values), n-queens-problems with n up to 20 and quasi-group completion problems with up to 81 agents. To compare the performance of the algorithms we counted overall constraint checks (cc), concurrent constraint checks(ccc), the overall number of messages(mc), the longest path of sequential messages(smc) and the run time (t) given in seconds. The runtime represents the "parallel time", i.e., the CPU+System time of the slowest agent.

All tests were run in a Java multi-threaded simulator where each agent implements a thread using random message delays and unpredictable thread-scheduling. All the threads were executed in one process and thus on one processor (2 Ghz Windows PC).

5.1 Basic Performance

In Figure 2 we show the median numbers of messages sent and the runtime to find one solution by different sized portfolios on fairly hard instances (density 0.3, tightness 0.4) of random problems (sample size 300). No aggregation was used in these experiments. The best known[2] variable-ordering (maxDegree) was used in each portfolio including those of size 1 which are equivalent to the basic algorithms. In the larger portfolios blind orderings (lex and random) and more instances of maxDegree were added. It can be seen that with increasing portfolio-size there is more communication (sent messages) between agents. In the same Figure we show the run time, which correlated strongly to smc and ccc. It can be seen that the performance improves up to a certain point when larger portfolios are used. In our experimental setting this point is reached with size 10. With larger portfolios no further speedup can be achieved which would make up the communication cost and computational overhead.

5.2 Risks

To evaluate the risks we used the same experimental setting as before but with random variable orderings and lexicographic value selection. This static value selection would reduce the R-Risk as widely as possible. Using random orderings would eliminate the effects we get from knowledge about heuristics and allow for a non-biased evaluation. Each portfolio was applied 100 times to one hard random problem instance. The standard-deviation of the runtime is shown in Figure 3 on a logarithmic scale. It can be seen that the risk is reduced significantly with the use of portfolios. With portfolio size 20, for instance, the risks

[2] We made preliminary experiments to determine this.

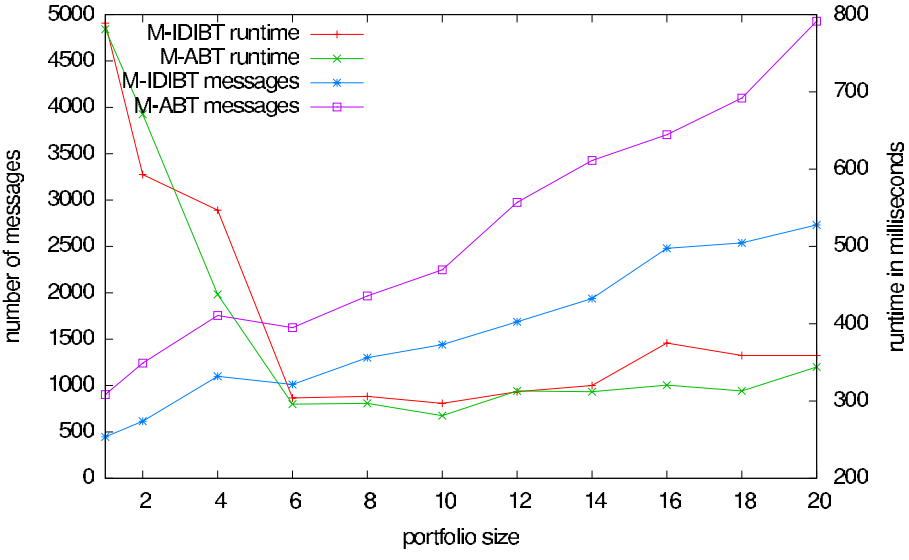

Fig. 2. Communication and runtime in portfolios

of M-IDIBT and M-ABT are 344 and 727 times smaller than the ones of IDIBT and ABT, respectively.

5.3 Performance with Aggregation

The benefit of Aggregation which is implemented with the different value selection heuristics is presented in Table 2. Each column in the table shows the median values of at least 100 samples solved with M-IDIBT with a portfolio of size 10 applied to 30 different hard random and quasigroup completion problems. The latter class of problems (cf. [GS01]) were encoded in a straightforward model: N^2 variables, one variable per agent, no symmetry breaking, binary constraints only. We solved problems with a 42% ratio of pre-assigned values which is the peak value in the phase transition for all orders, i.e., we used the hardest problem instances for our test.

In the table we refer to the aggregation methods introduced in Table 1, the bottom line shows the performance with random value selection (and thus no aggregation). When we consider the running time, it seems that the choice of the best method depends on the problem. For the quasigroup, aggregation based on the emulation principle seems to be better, on random problems not.

Interestingly, message passing operations present a different picture. It can be seen that *maxSupport* uses by far the least messages. These operations are reduced by a factor of 7 (resp. 38) for random (resp. quasigroups) problems. However, it cannot outperform the others significantly since the computation of this aggregation method is relatively costly. To this respect there is, however,

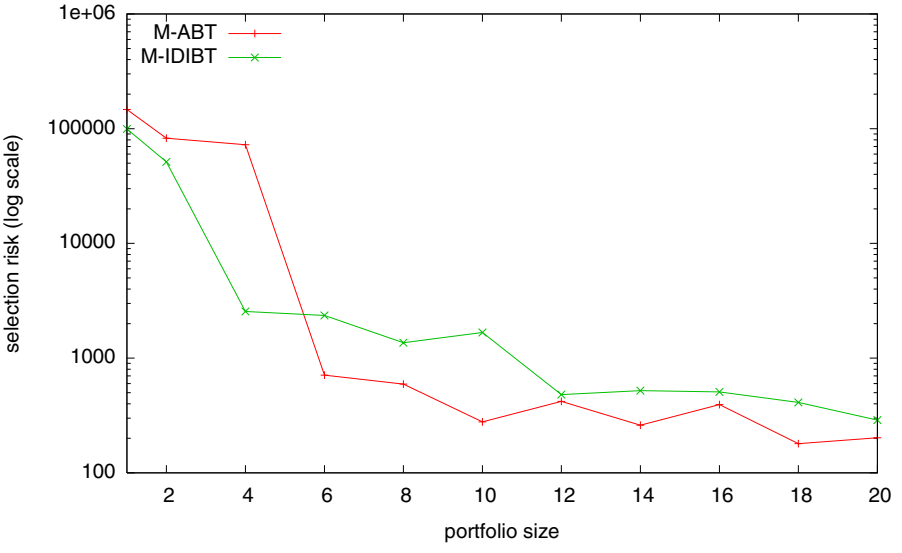

Fig. 3. S-Risk including the R-Risk emerging from distribution

Table 2. Performance of aggregation methods

	random			quasigroups		
	smc	ccc	t	$\frac{smc}{1000}$	$\frac{ccc}{1000}$	t
minUsed	367	2196	1.563	102	1625	448
maxUsed	379	**2118**	**1.437**	40	**635**	182
minBt	392	2281	1.640	104	1330	367
maxBt	433	2541	1.820	43	694	171
maxSupp	**57**	5718	1.922	**1.9**	3727	**143**
random	409	2406	1.664	73	1068	298

potential since we do not use an incremental algorithm for this. Moreover, message passing are the most critical operations in real systems and this for either long latencies or high energy consumption (e.g., ad-hoc networks [FM02]). The previous remark makes the *maxSupport* aggregation method really promising.

5.4 Overall Performance

In order to evaluate the relevance of the "M-" framework we investigated how it scales in larger and more structured problems. For this we applied good configurations found in the previous experiments to the well-known quasigroup completion problem.

Table 3 shows the experimental results of distributed search algorithms on problems of different orders (each column represents an order). ABT and IDIBT used the domain/degree variable ordering, which was tested best in preliminary experiments. In the larger portfolios we used domain/degree and additional other

heuristics including maxDegree, minDomain, lex and random. In all portfolios Aggregation with the method *maxUsed* was applied. For each order (column) we show the median runtime (in seconds) to solve 20 different problems (once each) and the number of solved problems. When less than 10 instances could be solved within a timeout of two hours we naturally cannot provide meaningful median results. In the experiments with M-ABT we have also observed runs which were aborted because of memory problems in our simulator. For order 8 these were about one third of the unsolved problems, for order 9 this problem occurred in all unsuccessful tests. This memory problem arising from the nogood-storage of ABT was addressed in [BBMM05] and is not subject to this research.

Table 3. Median performance and instances solved (out of 20) of quasigroup completion problems with 42% pre-assigned values

	5	6	7	8	9
ABT	**0.3, 20**	-, 8	-, 1	-, 0	-, 0
size 5	0.5, 20	5.9, 19	35.8, 14	-, 2	-, 0
size 10	0.6, 20	6.1, 20	40.6, 17	-, 8	-, 1
IDIBT	1.8, 20	12.4, 20	234, 20	4356, 16	-, 5
size 5	**0.2, 20**	**0.9, 20**	9.3, 20	709, 20	-, 6
size 10	0.3, 20	1.7, 20	**8.2, 20**	**339, 20**	-, 8

From the successful tests it can be seen that portfolios improve the median performance of IDIBT significantly. In the problems of order 7 a portfolio of 10 was 28 times faster than the regular IDIBT. Furthermore, portfolios seem to become more and more beneficial in larger problems as the portfolio of size 10 seems to scale better than smaller one. ABT does not benefit in the median runtime but the reduced risk makes a big difference. With the portfolio we could solve 14 resp. 17 instances of order 7 problems whereas the plain algorithm could only solve one.

5.5 Idle Time

To complete the presentation of our experimental results let us consider time utilization in distributed search. It appears that both considered classical algorithms underuse available resources. This is figured in the first two columns of Table 4 for various problem classes. The numbers represent the average idle times (10-100 samples) of the agents.

We can observe that ABT and IDIBT are most of the time idle. This idleness comes from the inherent disbalance of work in DisCSPs. Indeed, it is well known that the hierarchical ordering of the agents makes low priority agents (at the bottom) more active than high priority ones. Ideally the work should be balanced. Ideally one agent on the top of the hierarchy in context 1 should be in the bottom in context 2, etc (e.g., see agent in charge of variable X_1 in figure 1). Obviously, since we use well known variable ordering heuristics we cannot enforce such a property. However, the previous is an argument for multi-directional

Table 4. Idle times of agents in DisCSP

problem class	idle time of agents			
	ABT	IDIBT	M-ABT	M-IDIBT
easy random	87%	92%	56%	47%
hard random	92%	96%	39%	59%
n-queens	91%	94%	48%	52%
hard quasigroups	87%	93%	28%	59%

search which can use the previous idle time "for free" in order to perform further computations in concurrent search efforts. This is figured in the last two columns of the table where the M- framework with a portfolio of size 10 applied to the same problems makes a better use of computational resources and this can be understood as an important decrease in idle times for either M-ABT or M-IDIBT.

6 Related Work

The benefit of cooperating searches executed in parallel was first investigated for CSP in [HH93]. They used multiple agents, each of which executed one monolithic search algorithm. Agents cooperated by writing/reading hints to/from a common blackboard. The hints were partial solutions or nogoods its sender has found and the receiver could re-use them in its efforts. In contrast to our work, this multi-agent system was an artefact created for the cooperation. Thus the overhead it produced, especially when not every agent could use its own processor, added directly to the overall performance. Another big difference between Hogg's work and ours is that DisCSP agents do not have a global view of the searches and can thus only communicate what's in their agent-view which usually captures partial solutions for comparably few variables only.

Later the expected performance and the expected (Randomization-) risk in portfolios of algorithms was investigated in [GS97, GS01]. No cooperation between the processes was used here. In the newer paper the authors concluded that portfolios, provided there are enough processors, reduce the risk and improve the performance. When algorithms do not run in parallel (i.e., when not each search can use its own processor) the portfolio approach becomes equivalent to random restarts [GSK98]. Using only one processor, the expected performance and risk of both are equivalent. In contrast to Gomes and Selman we cannot allocate search processes to CPUs. In DisCSP we have to allocate each agent, which participates in every search, to one process. Thus the load-balancing is performed by the agents and not by the designer of the portfolio. In this paper we consider agents that do this on a first-come-first-serve basis. Furthermore we use cooperation between the agents and the parallelism is not an overhead-prune artefact.

Recent work on constraint optimization [CB04] has shown that letting multiple search algorithms compete and cooperate can be very beneficial without

having to know much about the algorithms themselves. They successfully use various optimization methods on one processor which compete for finding the next best solutions. Furthermore they cooperate by interchanging the best known feasible solutions. However, this method of cooperation cannot be applied to our distributed constraint satisfaction settings.

A different research trend performs "algorithm selection" [Ric76]. Here, portfolio does not represent competing methods but complementary ones. The problem is then to select from the portfolio the best possible method in order to tackle some incoming instance. [LBNA+03] applies the previous to combinatorial optimization. The authors use portfolios which combine algorithms with uncorrelated easy inputs. Their approach requires an extensive experimental step. It starts with the identification of problem's features which are representative of runtime performances. These features are used to generate a large set of problem instances which allow the collection of runtime data for each individual algorithm. Finally, statistical regression is used to learn a real-valued function of the features which allows runtime prediction. In real situation, the previous function predicts each algorithm's running time and the real instance is solved with the algorithm identified as the fastest one. The key point is to combine uncorrelated methods in order to exploit their relative strengths. The most important drawback here is the extensive offline step. This step must be performed for each new domain space. Moreover a careful analysis of the problem must be performed by the end-user to identify key parameters. The previous makes this approach highly unrealistic in a truly distributed system made by opportunistically connected components [FM02]. Finally knowledge sharing is not possible in this approach.

7 Conclusion and Future Work

In this paper we have presented a new generic framework for the execution of DisCSP algorithms. We have tested it with two standard methods but any tree-based distributed search should easily fit in the M- framework. The framework executes a portfolio of cooperative DisCSP algorithms with different agent-orderings concurrently until the first of them terminates. In real (truly distributed) applications, our framework will have to start with the computation of different orderings. The generic Distributed Agent Ordering heuristic (DisAO) [HBQ98] could easily be generalized at no extra message passing cost to concurrently compute several distributed hierarchies. The main idea is to simultaneously exchange several heuristic evaluation of a sub-problem instead of one.

This use of heterogeneous portfolios is shown to be very beneficial. It improves the performance and reduces the risk of distributed search. With our framework we were able to achieve a speedup of one order of magnitude while reducing the risk by up to three orders of magnitude compared to the traditional execution of the used algorithm.

The portfolios seem to make a better use of computational resources by reducing the idle time of agents. This is the first of two special advantages of the

application of portfolios in DisCSP: we do not have to artificially add parallelism and the related overhead but can use idle resources instead. The M- framework can be seen as a solution to the classical "work unbalance" flaw of tree-based distributed search algorithms.

We analysed and defined distributed cooperation (Aggregation) with respect to two orthogonal principles *diversity* and *emulation*. Each principle was applied without overhead within the limited scope of each agent's knowledge. This is the second special advantage of using Aggregation in DisCSP: it yields no communicational costs and preserves privacy because processes are not related to search efforts but to agents instead. Our experiments identified the emulation-based *maxSupport* heuristic as the most promising one. Indeed, it is able to efficiently aggregate partial solutions which brings a large reduction in message passing operations.

Our present results greatly improve the applicability of DisCSP algorithms by providing greater efficiency and robustness to two classical tree search algorithms. In future work we would like to investigate how portfolios are best composed and how they could implement a more informed Aggregation (beyond agent's scope). The composition could be studied with different hand or system made portfolios or by dynamic adaptation during search. The latter could provide more resources to the most promising efforts. The former could take advantage of heterogeneous portfolios involving various tree- and local-search combined with some distributed consistency-enforcement method (e.g., [Ham99]). Finally, knowledge Aggregation could be easily improved at no cost by adding extra information to existing message passing operations (search effort, etc). This would give a more informed view of the distributed system which could be used by the Aggregation methods.

References

[BBMM05] C. Bessiere, I. Brito, A. Maestre, and P. Meseguer. Asynchronous backtracking without adding links: A new member in the ABT family. *Artificial Intelligence*, 161:7–24, 2005.

[CB04] T. Carchrae and J. C. Beck. Low knowledge algorithm control. In *Proc. AAAI'04*, 2004.

[FM02] S. Fitzpatrick and L. Meertens. Scalable, anytime constraint optimization through iterated, peer-to-peer interaction in sparsely-connected networks. In *Proc. IDPT'02*, 2002.

[GS97] C.P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In *Proc. UAI'97*, pages 190–197, 1997.

[GS01] C.P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[GSK98] C.P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proc. AAAI'98*, pages 431–438. AAAI Press, 1998.

[Ham99] Y. Hamadi. Optimal distributed arc-consistency. In *Proc. CP'99*, pages 219–233, 1999.

[Ham02] Y. Hamadi. Interleaved backtracking in distributed constraint networks. *International Journal on Artificial Intelligence Tools*, 11(2):167–188, 2002.

[HBQ98] Y. Hamadi, C. Bessiere, and J. Quinqueton. Backtracking in distributed constraint networks. In *Proc. ECAI'98*, pages 219–223, 1998.

[HH93] T. Hogg and B. A. Huberman. Better than the best: The power of cooperation. In *1992 Lectures in Complex Systems*, volume V of *SFI Studies in the Sciences of Complexity*, pages 165–184. Addison-Wesley, 1993.

[LBNA+03] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *Proc. IJCAI'03*, page 1542, 2003.

[Pug04] J. F. Puget. Some challenges for constraint programming: an industry view. In *Proc. CP'04, invited talk*, pages 5–9. Springer LNCS 3258, 2004.

[Ric76] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[YDIK92] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proc. ICDCS'92*, pages 614–621, 1992.

[ZM03] R. Zivan and A. Meisels. Synchronous vs asynchronous search on DisCSPs. In *Proc. EUMAS'03*, 2003.