Multi-Directional Distributed Search with Aggregation

Georg Ringwelski and Youssef Hamadi
t-georgr@4c.ucc.ie, youssefh@microsoft.com

February 25, 2005

Technical Report
MSR-TR-2005-27

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

1 Introduction

In many application domains constraint-based methods in tree-search are the technology of choice to solve NP-complete problems today. However, when actually applying the algorithms out-of-the-box, i.e. without further customization, we have often experienced inacceptable performance. This results from various well-investigated factors including bad modelling and the choice of a wrong labelling strategy. This is particularly present for the state-of-the-art search algorithms for distributed constraint satisfaction (DisCSP). This paper 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.

It is well known that clever heuristics for variable and value selection can lead to good performance when solving certain problems[4]. However, they all remain heuristics and we do not yet really understand what makes them good. The risk of having chosen just the wrong heuristic will always be present. In a preliminary experiment we compared the performance of four different variable-ordering heuristics in distributed search on random problems. One of these was a well-known "clever" heuristic, which turned out to be the best choice for 59% of the considered problems. In the remaining 41% of the tests other, blind or even "anti-clever" heuristics were the best. This example makes clear, what we have also experienced in industrial projects where domain specific knowledge was used by experts to find good heuristics: we can never predict which heuristic will be best for a particular problem instance. Thus, ideally we would not have to make a choice for a strategy at all and rather be able to use an algorithm "out-of-the-box" which finds the best strategy itself [11]. This is implemented in algorithm portfolios which execute several strategies in parallel and let them compete for being the first to finish.

A special motivation to apply competition in distributed search is that in contrast to earlier portfolio-approaches [9, 5, 6] we do not have to add parallelism (and its associated overhead) to the algorithm as it is already inherent in the distribution. In order to evaluate this potential we set up an experiment where we measured the idle time of agents in the distributed search algorithms IDIBT [8] and ABT [13] and a portfolio of 10 IDIBT searches in parallel. We used a multi-threaded simulator with random message delays and agent-activation. In this simulator each agent imposes a thread and processes messages whenever it is scheduled to the CPU. We could observe that with the plain algorithms (ABT,IDIBT) the agents run most of the time idle and do not actively work on finding a solution. The average idle times (10-100 samples) of the agents in some classes of problems are shown in Table 1. These idle times can be used "for free" to perform further computations in concurrent search efforts which makes the portfolio-approach particularly interesting for distributed problems. As can be seen in the figure the idle time is reduced when a portfolio of 10 searches is used (M-IDIBT, M-ABT)

In this paper we define a notion for the risk we have to face when choosing a variable-ordering and present the new "M-" framework[1] for the execution of

[1] M stands for Multi-Directional. The name is derived from "Bidirectional" search. "M-"

problem class	idle time of agents			
	ABT	IDIBT	M-IDIBT	M-ABT
easy random	87%	92%	47%	56 %
hard random	92%	96%	59%	39%
n-queens	91%	94%	52%	48%
hard quasigroups	87%	93%	58%	28%

Table 1: Idle times of agents in DisCSP.

distributed search. We apply the framework in two case studies where we define the algorithms "M-ABT" and "M-IDIBT" which improve their counterparts ABT and IDIBT significantly. With these case studies we can show the benefit of competition and cooperation for the underlying distributed search algorithms. We expect the "M-" farmework to be similarily beneficial for other DisCSP algorithms. "M-" uses a portfolio of searches with different variable ordering heuristics in parallel which compete for being the first to finish and furthermore cooperate by exchanging gained knowledge. Cooperation of distributed searches is implemented with the aggregation of knowledge within agents and thus yields no extra communcation. 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 refer to related work and in Sections 4 and 5 we present the new "M-" framework. Section 6 describes our case studies and shows an empirical evaluation of them. The last Section summarizes the results and outlines some ideas for future work.

2 Risks in Search

In [6] "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. With random value selection for example it is high and with a completely deterministic algorithms it will be close to zero. In order to prevent confusion we will refer to this risk as to the Randomization-Risk (R-Risk) in the rest of the paper.

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

Reducing the R-Risk leads in many cases to tradeoffs in performance [7]. Such that the reduction of this risk is in general not desirable. For instance, we would

searches in multiple directions, namely agent topologies, at the same time.

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 the expected performance is worse. Thus, for this example we should refer to the larger range as to a "chance" rather than to a risk.

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 [14]. We measured the R-Risk in DisCSP in a preliminary experiment, where randomness emerged from distribution only. We solved binary DisCSPs with the IDIBT and ABT algorithms with random message delays and inpredictable 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 with our simulator in 297–5374 milliseconds IDIBT applied 100 times took 1640–1984 milliseconds. The R-Risk resulting exclusively from distribution was thus 807 for ABT and 96 for IDIBT.

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 variable ordering in IDIBT in a preliminary experiment on small, fairly hard random problems (15 variables, 5 values, denisty 0.3, tightness 0.4). 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. A typical range of run time for one (average) problem instance was 0.29 – 54.7 seconds with IDIBT in this experiment. This range was achieved with applying each heuristic 10 times to the problem. The S-Risk, including the inevitable R-Risk was thus 84754 with the four considered variable-orderings on that random DisCSP. This value is close to the average for the 30 considered problems with this tightness. In the 10-queens problem we tried random, lexicographic and middle-first variable ordering and lexicographic value selection with ABT and IDIBT. Each algorithm was run with each ordering 100 times and the range of runtime was 140–5374

milliseconds for ABT and 141–4781 for IDIBT. The S-Risk of the mentioned variable orderings is thus 818 for ABT and 848 for IDIBT.

3 Related Work

The benefit of cooperating searches executed in parallel was first investigated for CSP in [9]. 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 its sender has found and the receiver could use them by trying to do the same. 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 we use message passing instead of the (for DisCSP inpracticable) global blackboard. Finally DisCSP agents do not have a global view to 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 with ILOG solver was investigated [5, 6]. No cooperation between the processes was used here. They calculated in the first paper statistically that both risk and runtime can be very low with uniform portfolios when each algorithm can use its own processor. From this they concluded that using a uniform portfolio is better than using different algorithms in parallel. This conclusion was wrong for the following two reasons: first, they ignored that also the worst observed cases used a uniform portfolio; and second they didn't capture the S-Risk. In the newer paper they 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 [7]. 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. Another difference is that we distinguish R-Risk and S-Risk. From Gomes and Selman's results it becomes clear that using uniform portfolios may be very beneficial, but also that this may be the worst thing to do. Thus, the selection of a uniform portfolio, i.e. deciding to use one algorithm exclusively, yields the risk of actually doing the worst possible.

Recent work on constraint optimization [2] has shown that letting multiple search algorithms compete and cooperate can be very beneficial without haveing 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 in distributed problem solving as no global views to current states are available until a solution is found and in this case the (satisfaction) problem is already solved.

4 Multi-Directional Distributed Search

By a direction in search we refer to 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. For each constraint a *directed* connection between two agents 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.

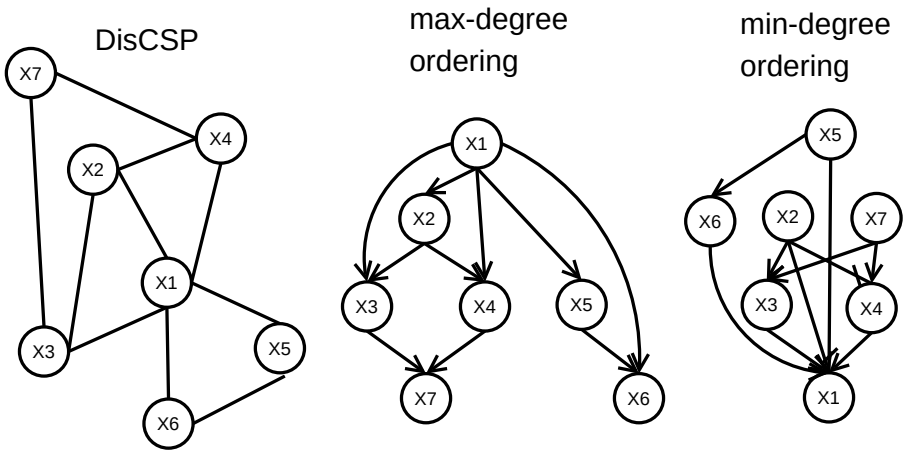

Figure 1: DisCSP and agent topologies implied by variable orderings

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. The idea of multiple contexts in DisCSP was first used to employ parallel search in IDIBT [8]. 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 X4 for example, would contain two contexts. In the one which related to maxDegree it would store X1 and X2 as higher prioritized adjacent agents an in the other it would store

X2 and X7. In ABT od IDIBT it would thus address backtracking-messages to agents X1 or X2 in one search effort and to X2 or X7 in the other. In Figure 2 we show how an agent hosting variable X4 from Figure 1 could employ the two described variable orderings. It hosts two different current values, one for each search and two different agent-views which contain its beliefs about the values of higher-priority agents. The set of these higher-priority agents depends on the chosen topology and thus on the chosen variable ordering.

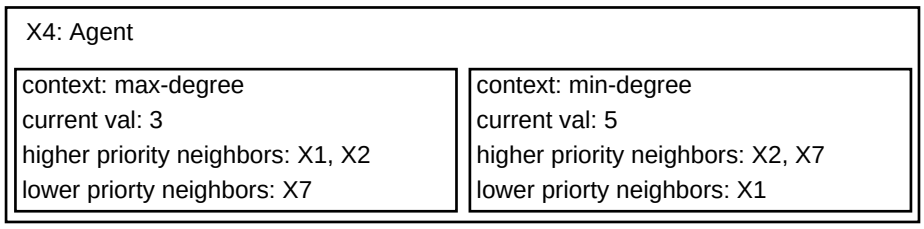

Figure 2: Two contexts in one agent hosting one variable

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 [8].

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 heuristiscs we should include them but the use of many of them will reduce the risk of bad performance for every single problem instance (cf. experiement 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 tradeoffs and overheads for this are investigated in this paper.

The tradeoff in space is linear in the number of appplied orderings. Thus, it clearly depends on the size of the data structures that need to be duplicated for the contexts.

It turned out in our experiemnts that this extra space requirement is very small in our implementations M-IDIBT and M-ABT. We could observe that the extra memory needed with a portfolio of size ten applied to IDIBT is typically only about 5–10%. For ABT the extra memory when using 10 instead of one context differed depending on the problem. For easy problems, where few nogoods need to be stored the extra memory consumption was about 5–20%. For

hard problems we could observe up to 1000% more memory usage of the portfolio. This clearly relates to the well-known space-tradeoff of nogood-recording.

The tradeoff in computational costs will be described in detail in the Evaluation Section. Unlike the described monolithic portfolio approaches, we do not need a processor for every heuristic or agent in order to achieve a speedup. In DisCSP we can do the additional work in times when the agents would run idle otherwise. (cf. Table 1). The load-balancing is performed dynamically by the agents and does not have to be set a priori by the designer of the portfolio. We have observed in our experiments that we can handle a portfolio of 10 heuristics managed in in parallel in 81 agents with about 15MB memory usage on one processor.

5 Aggregation

Besides the idea of letting randomized algorithms compete to become "as good as the best" the "M-" frameowrk 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 addition the views could be enriched by adding information for aggregation-purposes. This could be communicated in messages the agents send anyway or even in extra messages for this purpose. Such methods are, however, not considered in this paper.

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 2. 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.

	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

Table 2: Methods of aggregation.

6 Algorithms

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

6.1 M-IDIBT

IDIBT [8] is an asynchronous tree-based search algorithm to solve binary, distributed Constraint Satisfaction Problems (DisCSP). A DisCSP [13] consists of a set of variables with associated domains and a set of constraints which define allowed combinations of values for the variables. A solution to a DisCSP is a variable assignment that satisfies all constraints. Each variable imposes an agent which can communicate with other agents exclusively by message passing. The agents try to find a solution to the problem. The IDIBT algorithm is a self-stabilizing protocol for such agents. It leads to a state where either it was proven that no solution exists or where a solution is represented by the values selected in each agent.

IDIBT performs tree-based search on a topology of agents that is determined by the structure of the problem. Whenever a constraint exists between two

variables, then a directed link is imposed between their associated agents. The direction of these links defines the applied variable ordering for the search. For instance, for the well-known maxDegree heuristic each link will point to the variable which is in the scope of fewer constraints. Furthermore for the soundness of the algorithm it is necessary to pre-process the topology with the DisAO algorithm [8]. This adds extra links between some pairs of agents where no constraint exists.

In the context of this work we have found a new way to omit this pre-processing and still achieve soundness. This is done by dynamically adding the required links during the execution of the search in a similar way as it is done in ABT [13]. Whenever a backtrack-message is sent to an agent which is not already stored to be adjacent and has a higher priority, then a link is imposed from the receiver of this message to its sender.

Lemma 1 *Dynamic linking preserves the correctness of IDIBT.*

Proof 1 *A proof can be obtained from the authors. The idea is that dynamic linking captures all cases where additional links are necessary. This is whenever a btSet-message is sent to a non-parent agent.*

IDIBT interleaves parallel search and distributed search by allowing for the concurrent management of multiple contexts in each agent. Thus each agent can actively participate in several search-efforts at the same time. These efforts can implement the parallel exploration of different parts of the search tree or the parallel solving of same (sub-)problems. However, the applied search heuristics (variable-ordering and value selection) are static and identical in all parallel searches. Thus IDIBT can only be used to apply uniform portfolios of heuristics and parallel search in DisCSP.

In order to allow for variation of applicable heuristics we extended IDIBT with functionality for dynamic value selection. This is implemented by storing the values that were already considered since the last backtrack. Whenever an agent is to pick a new value it chooses one which is not stored in this list. When an agent sends a backtrack-message it deletes the list of used values and thus allows for their re-use in combination with other values of higher prioritized variables.

Lemma 2 *Dynamic Value selection preserves the correctness of IDIBT with dynamic linking.*

Proof 2 *A proof can be obtained from the authors. The idea is that the value selection functions return valid values in exactly the same cases as the static functions in IDIBT do.*

6.2 M-ABT

This algorithm incorporates ABT [13] in the "M-" framwork. For this we implemented contexts by duplicating the local storage of current value agent-view

and nogood-store. Furthermore, every message additionally carries the id of its related search. No other changes were made to the original algorithm.

7 Empirical Evaluation

For the empirical evaluation of the "M-" framework we solved more than 180000 DisCSPs with M-IDIBT and M-ABT. In a first set of experiments we solved random binary problems with 15 variables, 5 values each, density 0.3 and tightness varying between 0.2 and 0.8. The sample size (per problem instance) was 20-100 and for each tightness we used 30 different problem instances. 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). All tests were run in a multi-threaded simulator where each agent implements a thread using random message delays and unpredictable thread-scheduling. The simulator was used with a single-processor Windows PC with a Pentium 4, 1.8 GHz.

7.1 Portfolio size

In Figure 3 we show the median numbers of messages sent and the runtime to find one solution by different sized portfolios on fairly hard instances (tightness 0.4) of random problems (sample size 300). No aggregation was used in these experiments. The best known variable-ordering (maxDegree) was used in each portfolio and thus also for the plain algorithms. In the larger portfolios only blind orderings and more instances of maxDegree were added. It can be seen that with increasing portfolio-size, the absolute numbers of mc rise. There is more communication between agents when more searches are executed in parallel. 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. With the considered problems, using one processor only, this point is reached with size 10. With larger portfolios no further speedup can be achieved wich would make up the communication and compuational overhead.

In Figure 4 we concentrate on the run time (same experimental setting as before) with different compositions of the portfolios. Heterogeneous portfolios are those we used before and homogeneous portfolios use the maxDegree variable ordering in all searches. It can be seen that heterogeneous portfolios are much more beneficial. This results from the fact that we can only make used of the chances resulting from the R-Risk with homogeneous portfolios. With heterogeneous portfolios we can make use of the potential of the S-Risk, which is always higher that that of the R-Risk (see Figure 6).

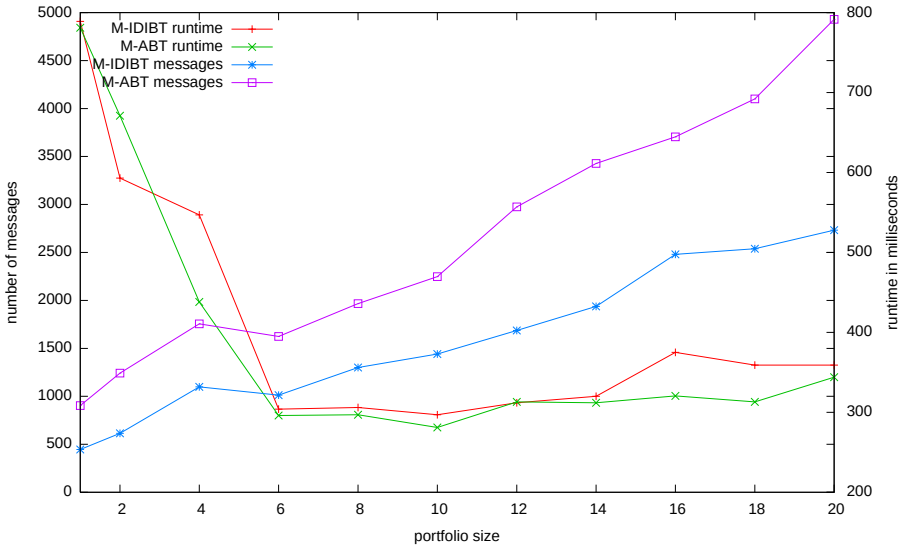

Figure 3: Communication and runtime in portfolios.

7.2 Risk

The Selection-Risk is defined as the standard deviation of the performance of different algorithms. Since our algorithms are distributed, the S-Risk cannot exclude the R-Risk. We reduced the randomness as much as possible by using deterministic value selection. In order to capture the overall risk (subsuming S-Risk and R-Risk) we used random variable orderings. This would eliminate the effects we get from knowledge about heuristics. Thus, we chose a low-knowledge approach [2] in this experiement. The reason for this is to get a statistically relevant evaluation of the S-Risk instead of another comparison of the expected performance of different heuristics. In a portfolio of size n we thus used n different, randomly generated variable orderings all using lexicographic value selection. Each portfolio was applied 100 times each to one hard random problem instance. The standard-deviation of the runtime with these portfolios is shown in Figure 5 on a logarithmic scale. It can be seen that the risk can be reduced significantly with portfolios. With a portfolio of size 20, for instance, the risk of IDIBT can be reduced to $\frac{1}{344}$ and the risk of ABT to $\frac{1}{727}$.

The total risk (or chance of improvement) we get with larger portfolios clearly relates to the composition of the portfolio. In Figure 6 we show the overall risk (S-Risk plus inevitable R-Risk) of portfolios of size up to ten. The decrease in the overall risk is much more significant when heterogeneous portfolios are used. For the homogeneous portfolios it is still present (note the log-scale) but not as significant.

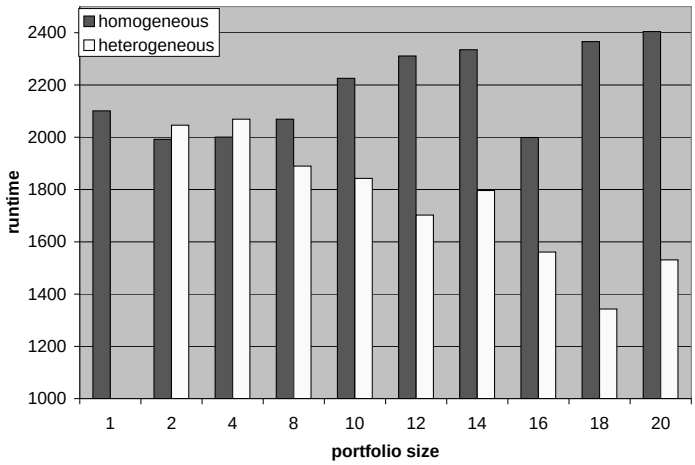

Figure 4: Runtime in homeogeneous and heterogeneous portfolios.

7.3 Diversity in portfolios and performance

In the above experiments we could observe that diversity in the portfolios is advantageous in both, performance and risk. To make this more clear we thus computed the correlation between these outcomes and the degree of difference of the used heuristics. This was measured as the average of the pairwise Hamming Distance of the used variable orderings. We could, however not observe, that the performance correlates to the hamming distance. In figure 7 we show the performances of 600 tests with two random varaible orderings. Using random orderings we obtained different hamming distances which we used to order the samples on the x-axis. The performance (y-axis) does not seem to correlate to that at all.

7.4 Aggregation

The benefit of Aggregation, which is implemented with the different value selection heuristics presented in Table 2 is shown in Table 3. Each column in the table shows the median values of 600 samples solved with a portfolio of size 4 applied to 30 different hard random problems. It can be seen that only maxBt and maxUsed do not perform better than random value selection. The best methods (wrt. smc) were maxSupport and minMaxUsed. The first selects the value that supports the most constraints wrt. the current agent-views and

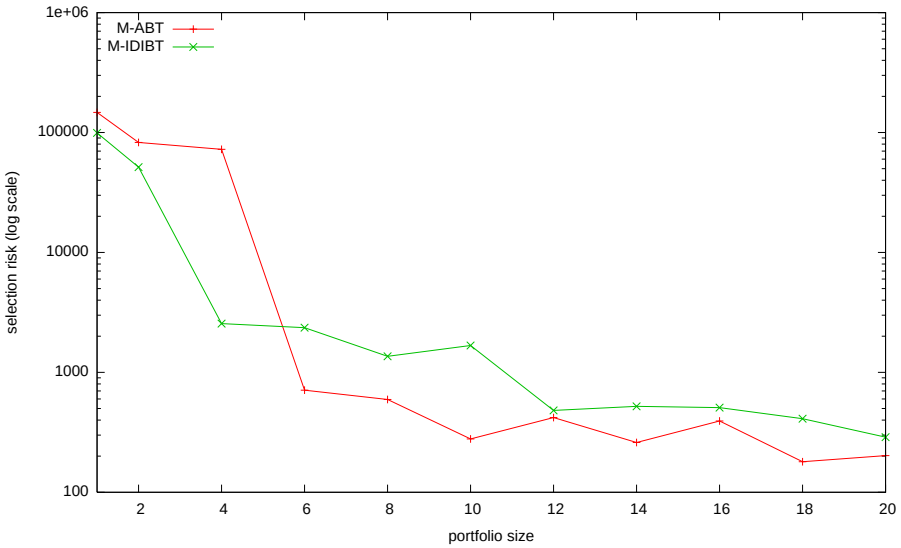

Figure 5: S-Risk including the R-Risk emerging from distribution.

the second is a composition of *minUsed* and *maxUsed*. It switches from the diversity-principle to emulation as soon as the number of backtracks is larger than half the size of the variable domain. The runtime and ccc results show that the computation of support is rather costly. We did not use an efficient (incremental) algorithm for this such that, given the smc, there is potential for speedup in *maxSupport*.

	smc	ccc	t
minUsed	425	2478	**2294**
maxUsed	446	2562	2483
minMaxUsed	**421**	**2410**	2296
minBt	429	2484	2499
maxBt	452	2623	2534
maxSupport	**421**	3312	2650
random	447	2639	2654

Table 3: Performance of aggregation methods.

7.5 Approaching real problems

In order to check the applicability of the "M-" framework we investigated how it scales in larger and more structured problems. For this experiment we used the well-known quasigroups completion problem cf. [6] in a straightforward model (N^2 variables, one variable per agent, no symmetry breaking, binary

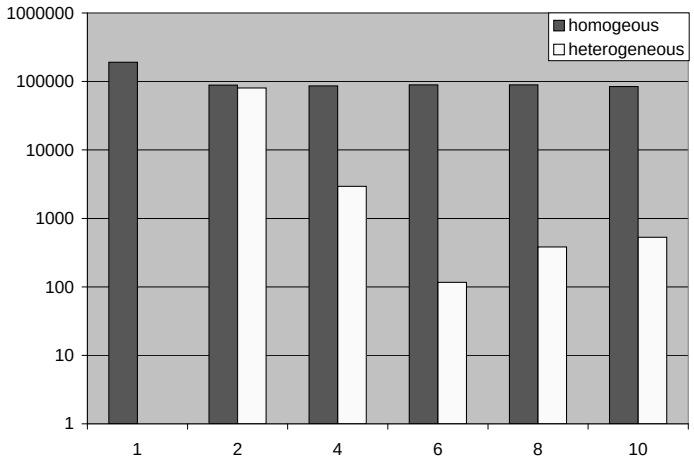

Figure 6: Risk in homeogeneous and heterogeneous portfolios.

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. Figure 8 shows the run time of distributed search algorithms on problems of different orders on a logarithmic scale. All algorithms used the *domain/degree* variable ordering (besides other orderings where applicable) and *minMaxUsed* value selection (where applicable). For each order we show the median runtime to solve 20 different problems (once each). We used a timeout of two hours whithin which we could solve all 20 instances of the presented results. Furthremore M-IDIBT 10 was able to solve 8 instances of order 9, M-IDIBT 6 and IDIBT 5 within the time limit. M-ABT could additionally not solve many of the larger instances because of memory problems. Using one thread per agent and 10 searches we had for instance for order 8 to store 640 nogood-stores in one process. Solving the hardest problems the nogood stores would potentially contain large portions of the search space. This memory problem of ABT was addressed in [1], but we didn't implement this method in our simulator. From the successful tests it can be seen that portfolios improve the performance of the algorithms significantly. In the problems of order 8 a portfolio of 10 IDIBT was 71 times faster than the regular IDIBT and twice as fast as with a smaller portfolio of size 5. Furthermore, portfolios seem to become more and more beneficial in larger problems as the portfolios of size 10 seems to scale better than smaller portfolios.

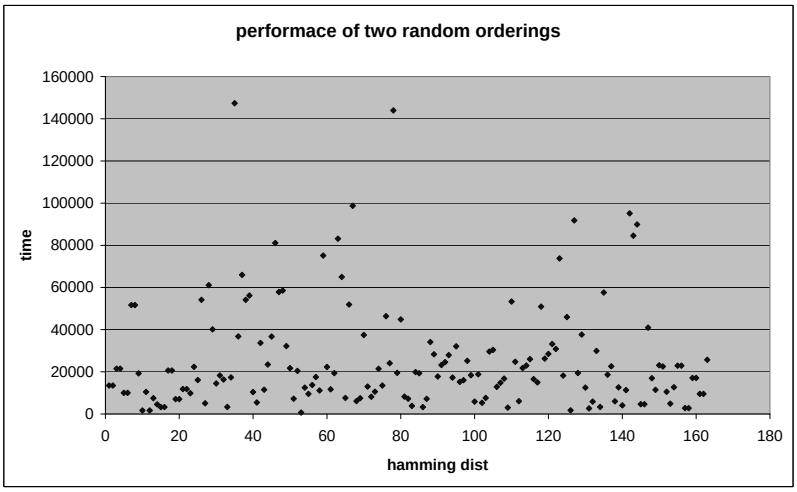

Figure 7: Hamming distance of orderings and performance.

8 Conclusion and Future Work

The use of heterogeneous portfolios of variable-orderings in distributed CSP is very beneficial. It improves the performance and reduces the risk of poor performance. With this technique we could 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.

Randomness is known to potentially improve the performance of search. However, in tradional approaches the tradeoff for this is the risk of very poor performance for some problem instances. With our portfolio approach we reduce this tradeoff by reducing the likelyhood of choosing the "wrong" random heurtistic. The cost is shifted to the computational effort we have to perform in parallel. However, in distributed algorithms this comes to a wide extent for free as agents would run more idle otherwise. The extra mamory consumption is linear in the size of the portfolio and was negligible in all our experiments.

In future work we will dynamically adapt the portfolio during search in order to provide more resources to the most promising efforts. This will be performed with the prioritization of messages to be processed. When an agent believes that a certain search effort is promising it will preferably process messages which are relevant for this effort. This will allow to leverage the load-balancing dynamically and find an ideal portfolio.

With the implementations of M-IDIBT and M-ABT we have provided a

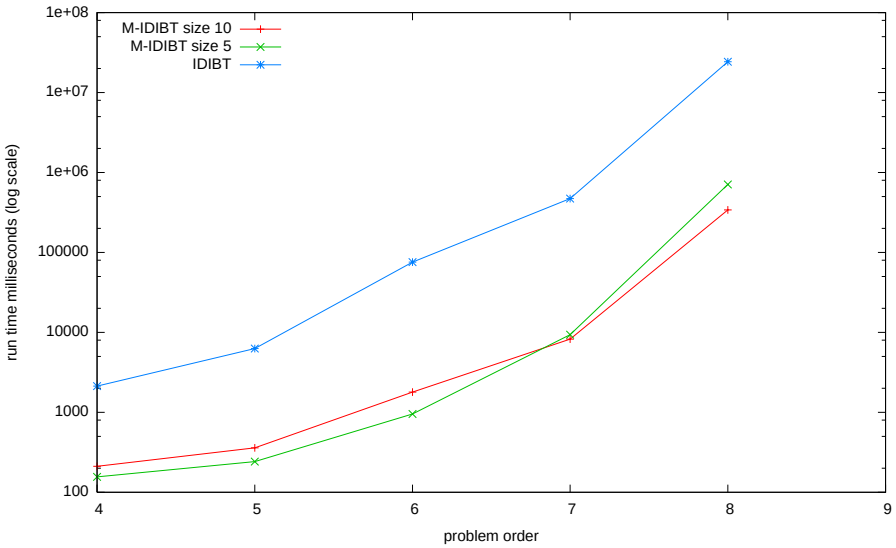

Figure 8: Median performance on quasigroup completion problems with 42% pre-assigned values.

case study of the benefit we get from competition and cooperation in tree-based distributed search. The same could be implemented on top of any other distributed algorithms. We could even combine multiple different algorithms, as opposed to just different topologies. In future work we will investigate the benefit and possible aggregation-methods of portfolios which contain for example also distributed local-search algorithms or consistency-enforcement.

The implementation of aggregation in a heuristic i.e. as value selection function, may be a first step to a new approach to search in CSP which is derived from Bidirectional Search as it is known in Planning cf. [10]. Instead of just using it as a heuristic, aggregation could also be implemented in an exact way. For problems that satisfy certain structural requirements we could already implement an algorithm in which agents can compose partial solutions and thus make the traversal of parts of the search-tree for some searches redundant.

Once DisCSP has found an algorithm which scales well enough we intend to integrate it in distributed systems of constraint solvers to manage real-world distributed problems cf. [3, 12]. In such settings each agent will host a powerful constraint solver and a few variables which are linked to other agents by (binary) equality-constraints. DisCSP will then be used to find solutions to the linked variables which extend to the local problems.

References

[1] 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.

[2] Tom Carchrae and J Chirstopher Beck. Low knowledge algorithm control. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI04)*, 2004.

[3] Y. L. Chong and Y. Hamadi. Distributed algorithms for log-based reconciliation. Technical Report MSR-TR-2004-104, Microsoft Research, Cambridge, UK, December 2004.

[4] Ian P Gent, Ewan MacIntyre, Patrick Prosser, Barbara M Smith, and Toby Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proc. CP96*, pages 179–193. Springer LNCS 1118, 1996.

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

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

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

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

[9] Tad Hogg and Bernardo A. Huberman. Better than the best: The power of cooperation. In Lynn Nadel and Daniel Stein, editors, *1992 Lectures in Complex Systems*, volume V of *SFI Studies in the Sciences of Complexity*, pages 165–184. Addison-Wesley, Reading, MA, 1993.

[10] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, PaloAlto, CA, 1980.

[11] Jean-Francois Puget. Some challenges for constraint programming: an industry view. In *Principles and Practice of Constraint Programming - CP2004, invited talk*, pages 5–9. Springer LNCS 3258, 2004.

[12] Georg Ringwelski and Armin Wolf. Global production planning in multiple facilities with the discs library. In *Proc. 19th Workshop on (Constraint) Logic Programming*, 2005.

[13] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *International Conference on Distributed Computing Systems*, pages 614–621, 1992.

[14] Roie Zivan and Amnon Meisels. Synchronous vs asynchronous search on discsps. In *Proceedings EUMAS*, 2003.