Exploring Hybrid Iterative Approximate Best-Response Algorithms for Solving DCOPs

Mihaela Verman, Philip Stutz, Robin Hafen, and Abraham Bernstein

Department of Informatics, University of Zurich,
Binzmühlestrasse 14, 8050 Zürich, Switzerland
{verman, stutz, bernstein}@ifi.uzh.ch, rhafen@student.ethz.ch
http://www.ifi.uzh.ch

Abstract. Many real-world tasks can be modeled as constraint optimization problems. To ensure scalability and mapping to distributed scenarios, distributed constraint optimization problems (DCOPs) have been proposed, where each variable is locally controlled by its own agent. Most practical applications prefer approximate local iterative algorithms to reach a locally optimal and sufficiently good solution fast. The Iterative Approximate Best-Response Algorithms can be decomposed in three types of components and mixing different components allows to create hybrid algorithms.
We implement a mix-and-match framework for these algorithms, using the graph processing framework SIGNAL/COLLECT, where each agent is modeled as a vertex and communication pathways are represented as edges. Choosing this abstraction allows us to exploit the generic graph-oriented distribution/optimization heuristics and makes our proposed framework configurable as well as extensible. It allows us to easily recombine the components, create and exhaustively evaluate possible hybrid algorithms.

Keywords: Distributed Constraint Optimization Problems, Hybrid Algorithms, Multi-Agent System as Graph Processing

1 Introduction

Many real-world problems, such as scheduling, and positioning or frequency selection for sensor networks, can be modeled as constraint optimization problems (COPs). Distributed constraint optimization (DCOP) algorithms have been proposed to distribute the control of variables to different software agents [20, 24]. Zhang et. al [37] distinguish between distributed complete algorithms that find a global optimum but are exponential and local iterative algorithms that only reach local optima, but do so avoiding exponential complexity. One subclass of local iterative algorithms, identified by Chapman et al. [4], is the approximate best-response algorithms, where agents can only observe the states of their neighbors. Chapman et al. propose a unifying theoretical framework for this class, which brings together local search algorithms and adaptive learning heuristics proposed in the Game Theory literature. They show that algorithms coming from both literatures share the same properties and that they can each be split into different components. This decomposition enables the creation of hybrid algorithms [5] and is the reason for which our software framework is strictly targetting the class of *local iterative approximate best-response algorithms*.

In this paper *we model DCOPs in a graph processing framework*, where agents are represented as active vertices and two variables/vertices that share a constraint can communicate their values to one another via edges. On the other hand, agents are also seen as a mix between different types of components, as described by Chapman et al. By only looking at the utility on the vertex, we can support constraints with multiple variables. Modeling the DCOP in this manner enables us to *take advantage of optimization heuristics available in distributed graph processing frameworks* such as SIGNAL/COLLECT [27, 29] and would also allow for the processing of larger problems. As a side-effect of this modeling approach we can also detect algorithm convergence for some algorithms in a distributed fashion, speeding up algorithm termination, and having a way to precisely determine convergence.

In summary, the contributions of this paper are the following: *First*, we provide a framework implementation for Local Iterative Best-Response Algorithms[1] based on a distributed graph processing framework simulating the agents. *Second*, we provide implementations for components of well known DCOP algorithms— the Distributed Stochastic Algorithm (DSA) with two of its variants ($\overline{A}$ and $\overline{B}$) [32, 1], a variant of Distributed Simulated Annealing (DSAN) [1], Joint Strategy Fictitious Play with Inertia (JSFPI) [18], Regret Matching [8], and Weighted Regret Matching with Inertia (WRMI) [2]. We easily combine components into hybrid algorithms, and evaluate these. Since we rely on a graph processing abstraction, the algorithms can easily be run both synchronously and asynchronously. *Third*, we introduce automatic convergence detection, again leveraging the capabilities of the underlying graph-processing framework. *Last*, we extend Chapman et al.'s [5] comparison of algorithms by providing a way to easily and exhaustively mix and match the components.

2 Related work

Distributed Constraint Optimization Problems A *Constraint Satisfaction Problem* (CSP) consists of: a set of variables , a set of domains from which each variable is allowed to take a value, and a set of constraints over subsets of the variables, which all need to be satisfied by an assignment of values to the set of variables. A *Constraint Optimization Problem* (COP) [22] assigns a utility function for the satisfaction or violation of each constraint with the given values of the variables over which the constraint was defined. The goal is to find the variable assignment that will maximize the global utility function, typically defined as the sum of utilities over all constraints.

A *Distributed Constraint Satisfaction Problem* (DCSP) [35] is produced when each variable is controlled by an agent. Analogously, a *Distributed Constraint Optimization Problem* (DCOP) [13, 21, 9, 10] shares the same characteristic as a DCSP with the added utility functions defined for COPs. As Chapman et al. [4] underline, we can define a private utility function for each agent, which will depend on the satisfaction of all constraints in which the variable (controlled by the agent) is involved (typically the

[1] The framework is open source, implemented in Scala, and can be found on github.com/elaverman/cuilt on the branch "optmas"

sum of utilities over all these mentioned constraints). Thus, the private utility of the agent will depend on its own state (the value of its variable) and on the states of all neighboring agents. The neighborhood relation is defined between two agents for which there exists a constraint over a subset that contains both their variables.

Iterative Approximate Best-Response Algorithms for DCOPs Chapman et al. [4] categorize algorithms for DCOPs into three main classes, and the one they focus on is represented by the approximate best-response algorithms, where agents can only be aware of their immediate neighbors' states. In [4] and [5], a theoretical unifying framework is proposed for iterative approximate best-response algorithms, which come from both game theory and computer science backgrounds. Based on the demonstration that DCOPs can be formulated as potential games, they prove that the algorithms from both literatures are equivalent in functionality and can be used for solving DCOPs and, more generally, for finding Nash equilibria in potential games.

This theoretical framework decomposes the algorithms into three components, making them easily comparable and more pluggable, in order to create new hybrid algorithms (Chapman et al. [5] introduce several hybrid algorithms based on the observed qualities of the components). An algorithm contains:

the state evaluation, which consists of updating an algorithm-specific **target function** to evaluate prospective states

the decision rule, which represents how the agent decides which action to take next by using the already computed target function from the state evaluation step; most algorithms use either stochastic or $argmax$ functions;

the adjustment schedule, which refers to the order in which the agents execute their processes and is mostly left unspecified, but it can either be parallel (purely parallel: also called flood, or parallel random: also called "with Inertia"), or preferential.

There are implemented frameworks for algorithms for Distributed Constraint Optimization Problems (such as FRODO [12], DisChoco2 [33], DCOPolis[31] and AgentZero [14]), but to our knowledge, none is tailored for local iterative best-response algorithms or takes advantage of the modularity described by Chapman et al. Our framework seeks to model exactly that, and we implement it inside a graph processing framework that comes with several other benefits.

The SIGNAL/COLLECT *framework* [27, 29] is a framework for distributed large-scale graph processing implemented in Scala.[2] The programming model is vertex-centric with vertices that communicate with each other via signals that are sent along directed edges. Each vertex collects the received signals to update its state, while the edges send signals to their respective target vertices to inform them about state changes. An algorithm is specified by providing the graph structure, initial states, as well as the SIGNAL and COLLECT functions for the vertices and edges respectively.

Consider the example of computing the Single-Source Shortest Path in a graph that only has positive edge weights. First, we initialize the state of the source vertex with 0

[2] SIGNAL/COLLECT is open source and more details about it can be found on the website: www.signalcollect.com.

and of the other vertices with infinity. With the SIGNAL function, vertices “message” their neighbors their own state plus the weight of the connecting edge. The COLLECT function updates a vertex’s state to the minimum of the current state and all incoming messages (or signals). In the end, every vertex will have as its state the length of the shortest path from the source to itself. Other examples of common algorithms that can be modeled in SIGNAL/COLLECT are the computation of PageRank, label propagation, or cellular automata. The framework has also been used as a basis for more complex systems, such as a triple store [30, 28], a system for fraud detection [26], and an implementation of Probabilistic Soft Logic [15].

We chose SIGNAL/COLLECT not only because DCOPs can map well to graph abstractions, but also because of the capabilities and unique features of this graph processing framework. The framework executes algorithms expressed in this model in parallel on the same machine, or distributed over a cluster, which, in the future, would enable us to look at large scale problems. It also allows both synchronous and asynchronous scheduling of the SIGNAL/COLLECT operations. Scheduling is optimized with scoring functions that can implement heuristics such as “only collect when a signal was received” or “only signal if the state has changed”. The framework allows to run aggregation operations over the graph, which offer insight and control over all the vertices in the graph. Convergence detection can either be based on detecting a change in scores or can be based on the result of global aggregations over the vertices. The scoring and convergence detection enables us to implement automatic algorithm termination.

3 An extensible Framework for Local Iterative Best-Response Algorithms

In this section, we show how our framework is designed and shortly discuss the four approximate best-response algorithms we implemented as examples.

Variables are represented as vertices; the neighborhood (two variables sharing at least one constraint) relationship is described by edges. The utility function of the “agent” responsible for a variable is dependent on the state of the adjacent vertices.

The algorithm will be specified by the way in which the different components are mixed and matched. To add different components, we use Scala traits, which can be easily combined. In Figure 1, we can observe the base Algorithm trait, with the types it needs and the methods that it requires. Implementations for the other traits in the diagram then get mixed and need to cover all the required methods of the Algorithm. The agent’s state is completely encoded in a State type. The state can have different implementations, specified by implementations of the StateModule, and dependent on the information that needs to be stored by an agent.

The methods are each provided by implementations of the several modules that extend Algorithm: the AdjustmentSchedule, DecisionRule and TargetFunction, providing the functionality described by Chapman et al. [4], and the added TerminationRule and Utility modules.

A specific algorithm has methods for creating specific vertices and edges, which can then be added to a graph. The SignalCollectAlgorithmBridge provides the implementations for these methods and for the vertex and edge. The DcopVertex, defined on the

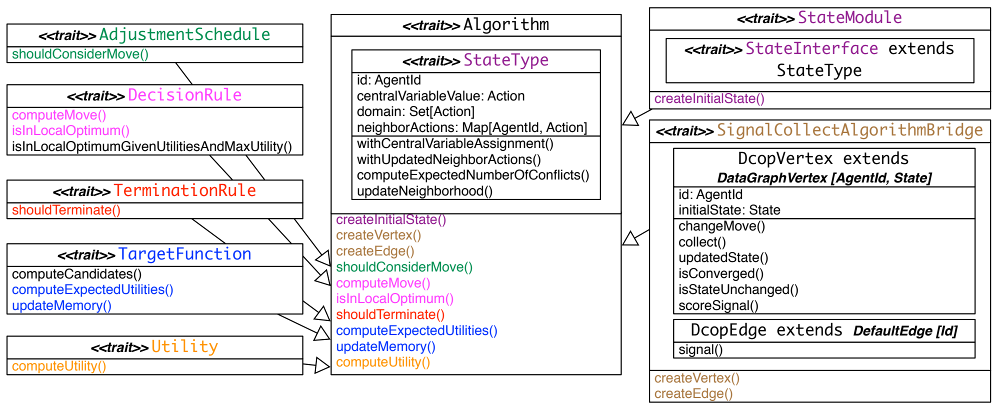

Fig. 1. Simplified diagram of the Algorithm trait and the traits defining the modules.

bridge, ties up together all the components provided by other modules, and describes, through its COLLECT method, the behaviour of an agent at every step. It extends one of the default types of vertices defined in SIGNAL/COLLECT, the DataGraphVertex. The SIGNAL method is defined on the DcopEdge that extends the DefaultEdge provided by the graph processing framework and, in our implementation, forwards the variable value to the neighbors.

A collect step is composed by: *first* getting the new messages from the neighbours and computing the current configuration and the target function and possibly the memory update (equivalent with the state evaluation phase), *second* determining if an update will actually take place by using the adjustment schedule, *third*, if the update should take place, then the new state will be created. The way to create this state is determined by the method *withCentralVariableAssignment*, defined on the state implementation being used. This will replace the variable assignment of the old state and potentially update any other dependent fields on the state.

This model of DCOPs allows to map the actual algorithm implementations from Chapman et. al's [4] framework to SIGNAL/COLLECT. The **state evaluation** and the **decision rule** phases are modeled in the COLLECT function. The **adjustment schedule**, in addition to being modeled in the COLLECT function, is also impacted by the choice of either the synchronous (for the flood or parallel random schedules) or asynchronous execution modes for the graph.

For preferential adjustment schedules such as those used in the Maximum Gain Messaging algorithm [16, 34], the COLLECT function would be implemented for the two stages (when the vertex computes the gain it has from changing strategy, and when it changes its state if it has the maximum gain out of all its neighbors). Signaling would be done accordingly. The sequential random schedule, which gives the possibility of changing the state to only one agent at a time, does not fit the needs for scalability unless it may be converted into an asynchronous schedule.

Algorithm 1 DSAN-TD: collect (agent i, time t)

```
candState := $getRandomState$
currUtil := $U_i[s_i(t-1), s_{-i}(t-1)]$
$\Delta$ := $U_i$[candState, $s_{-i}(t-1)$] $-$ currUtil
$p_t$ := **if** $(\Delta = 0)$ **then** $\gamma$ **else** $e^{\frac{\Delta t^k}{const}}$
**if** $(\Delta > 0 \vee (\Delta \leq 0 \wedge \text{random}(0,1) \leq p_t))$ **then**
    **return** candState
**else**
    **return** $s_i(t-1)$
**end if**
```

As a side-effect of this modeling approach, we can also detect algorithm convergence for some algorithms in a distributed fashion, by adjusting the SCORESIGNAL function, further speeding up algorithm termination. The SCORESIGNAL function takes into account if the state changed or not and if the shouldTerminate method on the decision rule returns true.

To illustrate the ease of this mapping, we provide implementations (through the recombination of components) of four algorithms from the category of iterative approximate best-response algorithms for DCOPs (as highlighted by [4]).

The Distributed Stochastic Algorithm (DSA) [32] is one of the baseline algorithms usually used for evaluation. An agent only takes into account the states most recently received from its neighbours, which it considers will be repeated, and given those, it selects the candidate state that would maximize the utility function.

Arshad et al. [1] provide asynchronous versions of DSA. However, we slightly adapted this version, in order to be the same as the synchronous one, and the only difference between the two is determined by the Execution mode.

The decision rule then uses the difference between the utilities of the candidate state and the past state. Specifically, DSA-A chooses the candidate state over the past state, with a given probability p, only if the candidate is strictly better than the past state. DSA-B follows the same procedure except it can also choose the candidate state with a probability p when it is as good as the past state, but there are still conflicts. Our implementation comprises of the modified variants, DSA-$\overline{A}$ and DSA-$\overline{B}$, presented in [1]. Compared to the original variants presented in [36], these modifications randomly choose a maximizing state when multiple ones exist.

Distributed Simulated Annealing with Termination Detection (DSAN-TD) is a modification of the Distributed Simulated Annealing algorithm [1]. DSAN first selects a candidate at random. If the candidate state improves the utility, then it is selected. Else, the candidate is chosen (i.e., exploration) with a probability that decreases over time, usually $e^{\frac{\Delta}{\tau}}$, where τ is a temperature function over time.

As presented in a companion paper [7], the original algorithm exhibits oscillations when running on under-constrained problems. To address this issue, as presented in Algorithm 1, we need a fixed small switching probability γ in the case where the Δ between the utility of the candidate and current state is 0. However, because in

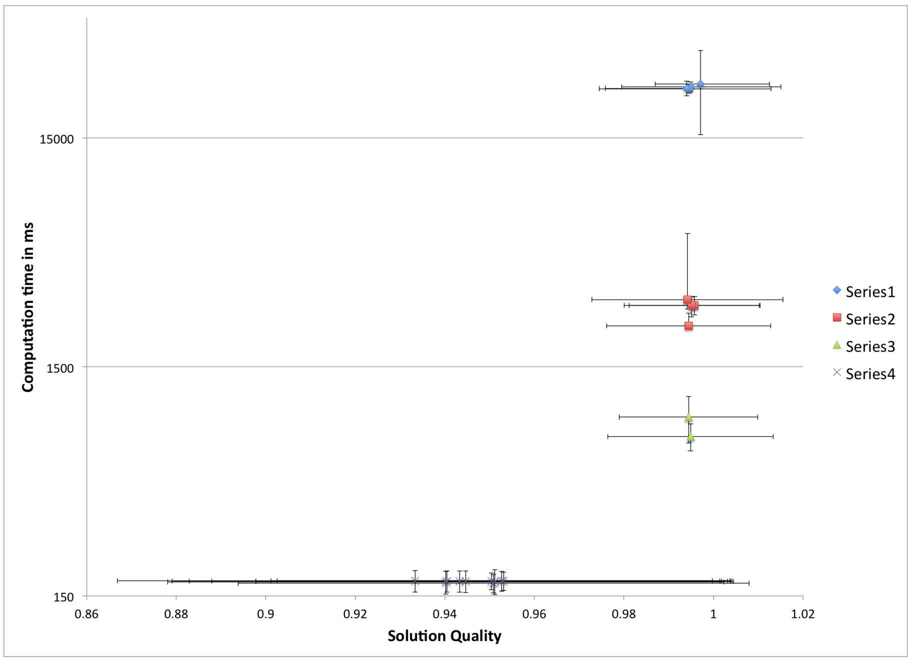

Fig. 2. Solution quality (X axis) vs. computation time in ms (logarithmic Y axis) for the top 10 performing combinations in terms of average solution quality and the top 10 in terms of computation time. The algorithms and their series can be seen in Table 1.

our framework we need to automatically detect termination, contrary to [7], γ is not fixed, but decreases over time. In our experiments we used $\gamma = e^{\frac{maxNegativeDeltat^k}{const}}$. $maxNegativeDelta$ is a parameter that can be set, and it is also used in detecting termination: Each vertex needs to have given a best response and have γ smaller than a certain ϵ, which we fixed at 0.001.

Joint Strategy Fictitious Play with Inertia (JSFPI) [18] proposes to first compute the average utility over time for each candidate state. The candidate state with the maximum expected utility is then chosen. We also implemented the fading memory variant described in the same paper, where the past possible states' utilities are discounted by a discount factor ρ.

Weighted Regret Matching with Inertia (WRMI, also called Generalized Regret Monitoring With Fading Memory and Inertia in its original paper) [2] is a regret-based game theory algorithm. The target function first computes the difference between the candidate's and current state's utilities, again discounted over time. The positive differences are called regret. The algorithm picks the candidate with a probability proportional to its regret. In our implementation, if all regrets are equal to zero, then the previous state is picked as candidate. We also implemented Regret Matching [8], which computes the average regrets over time and does not use a discount factor. Due to the probabilistic decision rule, termination is detected in this case when for all agents, the values of their target functions converge.

Besides the components of these algorithms, we also added the ϵ-Greedy Decision Rule [4], which explores with a fixed probability ϵ, or uses the argmax function otherwise.

In the next section we present the evaluations of the described algorithms.

4 Experiments

So far we suggested modeling COPs as graph computations and provided implementations of four approximate best-response algorithms based on the SIGNAL/COLLECT graph processing model. Thus, we demonstrated SIGNAL/COLLECT's suitability for implementing such problems. We also proposed an adjustment to the DSAN algorithm, DSAN-TD, that facilitates convergence and we discussed termination detection. In this section, we provide through our experiments empirical evidence for the suitability of SIGNAL/COLLECT's abstraction for the discussed class of algorithms, and we evaluate combinations of algorithm components, demonstrating how it is possible to automate creating hybrid algorithms as Chapman *et. al* suggest [5]. To evaluate our algorithms we chose the *vertex coloring problem*, typically used in the literature to benchmark DCOP algorithms.

For our hybrid algorithms evaluation, we generated algorithms by taking all possible combinations of the following components:

Target functions:
- ◦ MemoryLessTargetFunction – the utility function, like in DSA,
- ◦ AverageExpectedUtilityTargetFunction – the average utility over previous steps, like in JSFPI,
- ◦ WeightedExpectedUtilityTargetFunction (ρ=0.2, 0.4, 0.6, 0.8) – the average utility over previous steps, like in Fading Memory JSFPI,
- ◦ AverageRegretsTargetFunction, – the average regrets over previous steps, like in Regret Matching,
- ◦ DiscountedAverageRegretsTargetFunction (ρ=0.2, 0.4, 0.6, 0.8) – the average regrets weighted by ρ over previous steps, like in WRMI.

Decision rules:
- ◦ With NashEquilibriumConvergence:
  - ◦ ArgmaxADecisionRule – which selects the argmax candidate state only if it is strictly better than the past state, as in DSA-$\overline{A}$,
  - ◦ ArgmaxBDecisionRule – which selects the argmax candidate state only if it is strictly better than the past state or as good as the past state and there are still conflicts, as in DSA-$\overline{B}$,
  - ◦ EpsilonGreedyDecisionRule (ϵ=0.001, 0.01, 0.1) – which explores with a fixed probability ϵ, or uses the argmax function otherwise.
- ◦ With SimulatedAnnealingConvergence – the algorithm terminates when there is a Nash Equilibrium and for all agents γ approaches 0, as in DSAN-TD:
  - ◦ SimulatedAnnealingDecisionRule (const = 1, 1000, k=2, $negDeltaMax$ = -0.01, -0.0001) – as in DSAN-TD

- ○ With DistributionConvergence – the algorithm terminates when the memory used for the target function computation by each agent has converged:
  - ○ LinearProbabilisticDecisionRule – probabilistic over the values with non-negative regret (regret$_i$) or the candidate state is chosen randomly if the regret values for all actions is 0, as in WRMI

Adjustment schedules:

- ○ ParallelRandomAdjustmentSchedule (changeProbability=0.2, 0.4, 0.6, 0.8 for synchronous runs and changeProbability=0.95 for asynchronous runs) – the probability for an agent to switch to the candidate state regardless of its quality is given by the changeProbability parameter,
- ○ FloodAdjustmentSchedule – all agents change their state to the candidate state.

No. and Series	Combination	Exec.	Solution quality		Time (ms)	
			avg	std	avg	std
	Highest solution quality					
1 (1)	DiscountedAverageRegrets-ArgmaxB-Floodrho0.8	sync	0.9971	0.0101	25776.5	10276.8
2 (2)	MemoryLess-SimulatedAnnealing-FloodnegDeltaMax-0.01k2.0const1000.0	sync	0.9958	0.0147	2777.4	265.6
3 (2)	MemoryLess-SimulatedAnnealing-ParallelRandomnegDeltaMax-0.01k2.0const1000.0p0.8	sync	0.9951	0.0151	2783.0	248.5
4 (3)	MemoryLess-SimulatedAnnealing-ParallelRandomnegDeltaMax-0.01k2.0const1000.0p0.95	async	0.9949	0.0185	744.4	100.0
5 (1)	WeightedExpectedUtility-SimulatedAnnealing-FloodnegDeltaMax-1.0E-4k2.0const1000.0rho0.6	sync	0.9949	0.0154	25012.1	1228.5
6 (1)	MemoryLess-SimulatedAnnealing-ParallelRandomnegDeltaMax-1.0E-4k2.0const1000.0p0.6	sync	0.9947	0.0202	24562.0	1059.3
7 (2)	MemoryLess-SimulatedAnnealing-FloodnegDeltaMax-1.0E-4k2.0const1000.0	async	0.9944	0.0183	2266.5	303.6
8 (3)	WeightedExpectedUtility-SimulatedAnnealing-FloodnegDeltaMax-0.01k2.0const1000.0rho0.8	async	0.9944	0.0155	905.5	207.9
9 (2)	DiscountedAverageRegrets-epsGreedy-ParallelRandomeps0.1p0.6rho0.4	sync	0.9942	0.0213	2949.0	2779.3
10 (1)	MemoryLess-SimulatedAnnealing-FloodnegDeltaMax-1.0E-4k2.0const1000.0	sync	0.9940	0.0182	24648.2	1815.3
	Lowest computation time					
1 (4)	MemoryLess-epsGreedy-Floodeps0.001	async	0.9509	0.0571	170.1	15.5
2 (4)	WeightedExpectedUtility-ArgmaxA-Floodrho0.4	async	0.9402	0.0621	172.5	19.7
3 (4)	WeightedExpectedUtility-ArgmaxA-Floodrho0.6	async	0.9511	0.0534	173.4	21.4
4 (4)	WeightedExpectedUtility-ArgmaxB-Floodrho0.4	async	0.9447	0.0567	173.9	18.7
5 (4)	MemoryLess-epsGreedy-Floodeps0.01	async	0.9504	0.0527	174.0	14.5
6 (4)	WeightedExpectedUtility-epsGreedy-Floodeps0.001rho0.2	async	0.9404	0.0614	174.1	18.3
7 (4)	WeightedExpectedUtility-epsGreedy-Floodeps0.001rho0.6	async	0.9433	0.0604	174.1	18.7
8 (4)	WeightedExpectedUtility-epsGreedy-Floodeps0.001rho0.8	async	0.9527	0.0515	174.5	17.2
9 (4)	WeightedExpectedUtility-ArgmaxB-Floodrho0.6	async	0.9531	0.0506	174.6	16.1
10 (4)	DiscountedAverageRegrets-ArgmaxA-Floodrho0.2	async	0.9333	0.0664	174.6	18.7

Table 1. Statistics for the top combinations in terms of solution quality and computation time. The numbers in the parantheses in the first column represent the corresponding Series from Figure 2.

In the case of the Parallel Random adjustment schedule, for the asynchronous mode we ran it with a degree of parallelism $p = 0.95$. The reason for choosing that specific degree of parallelism for the asynchronous mode is that the asynchronous mode does not shield from thrashing behaviour. In preliminary experiments, many runs on the bigger graphs experienced non convergence, even in cases where the synchronous variants were able to converge. This, furthermore, leads to the detection of a Nash Equilibrium, even though the run was not converged. As an example, if we take two vertices that can both choose either red or blue, in the event that they are scheduled to collect almost in parallel, a thrashing behaviour will occur, but the belief of each vertex is that it already gave a best response. However, convergence is only detected when no more changes occur. Therefore, we reduced the degree of parallelism, in order to reduce such behaviour.

The graphs had 40 vertices and were constructed like the ones in [5], with a chromatic number of 3, 4 and 5 and an average edge density of 3, constraints with utility 1. We had 5 graphs for each configuration, and all the algorithms were run 5 times on these graphs. The machines that we used for this evaluation have 128 GB RAM and two E5-2680 v2 at 2.80GHz processors, with 10 cores per processor.

Table 1 and Figure 2 show the top 10 combinations in terms of solution quality and convergence speed. We can observe four clusters. The bottom cluster in Series 4 is represented by all the top 10 convergence speed algorithms. They all ran asynchronously, with a Flood adjustment schedule, and with either argmax or epsilon greedy adjustment schedules. They were all hybrids, algorithms 2-4 being modifications of Fading Memory JSFPI, with a flood schedule, and algorithm 10 being a modification of WRMI with a flood schedule instead of a parallel random one.

The cluster in Series 1, with highest utility, but lowest convergence speed from the top 10 in terms of quality, contains combinations that use the Simulated Annealing schedule, and also a combination of Discounted Average Regrets, Argmax-B and a Flood schedule. Algorithms 1, 5 and 10 are hybrids, whilst algorithm 6 is DSAN-TD. None of these runs was asynchronous, and the improved quality comes with a high cost for computation time compared to the algorithms in Series 4.

In Series 3, with good speed of convergence, we have two Simulated Annealing combinations run in asynchronous mode, algorithm 4 being DSAN-TD, and algorithm 8 being a hybrid.

Series 2, between Series 1 and 3 when it comes to speed of convergence, contains two hybrids of DSAN-TD with Flood schedule (algorithms 2 and 7), a pure DSAN-TD (algorithm 3), and a hybrid of Discounted Average Regrets, Epsilon Greedy and a Parallel Random schedule.

To summarize, the only non-hybrid algorithm that appeared in the top 10 algorithms in terms of solution quality was DSAN-TD. The rest were hybrids, most of them using the Simulated Annealing decision rule. In the case of computation time, the top algorithms were asynchronous hybrids with Flood schedule, most of them using the Average Weighted Expected Utility target function from Fading Memory JSFPI and either argmax or Epsilon greedy decision rules. From the experiment, we see that hybrid algorithms can lead to good results no matter if the aim is reduced computation time or improved quality, and that running the algorithms asynchronously seems to positively impact speed of convergence.

5 Limitations and future work

As our next step, we intend to evaluate the hybrid algorithms on different types and scales of problems and investigate larger problems using the distributed version of SIGNAL/COLLECT on a cluster. This will also enable us to detect changes in the behaviour of the algorithms with regard to graph scale and structure.

In addition, we would like to go beyond synthetic data sets and evaluate the applicability of our approach to real-world constraint problems that contain more complex utility functions, as well as constraints between multiple variables. That would also enable us to verify and improve the generalizability of the framework.

It would also be desirable to explore an alternative modeling approach, where both constraints and variables are each represented as a vertex resulting in a bipartite graph. This approach, which would simplify the mapping of multi-ary constraints, has already been implemented in the context of local message-passing algorithms [25] and can be implemented in the SIGNAL/COLLECT framework as shown by [19], and by [23] in the context of loopy belief propagation. Another direction of extension would be to allow for seamless switching of algorithms for a certain agent.

Another very important step which we are already addressing is to implement more components, and to enable automatic mixing of components based on a given set of dependencies between different types of components.

6 Conclusions

Chapman et al. [4, 5] have proposed a way to create new hybrid algorithms. In this paper, we introduce a software framework for local iterative approximate best-response algorithms that takes advantage of newly emerging graph processing frameworks such as SIGNAL/COLLECT and enables the exploration of the entire space of hybrid algorithms. Our implementation comes with several benefits: it is configurable and extensible by allowing us to easily create and mix components into hybrid algorithms that can then be evaluated in a straight-forward fashion, it has automatic convergence detection, and it allows us to also exploit the advantages of asynchronously executing algorithms.

In our evaluation, we show that hybrid algorithms can play an important role when it comes to both speed of convergence and solution quality.

We believe that our findings and our implementation enable more thorough explorations of new hybrid algorithms. Using a graph processing framework for the implementation will also allow in the future to apply local iterative algorithms to big real-world constraint problems.

Acknowledgments. We would like to thank the Hasler Foundation for supporting under grant number 11072 the SIGNAL/COLLECT project, which we used for implementing our framework, and the Dinu Patriciu Foundation for the ”Open Horizons” scholarship for Mihaela Verman.

References

1. Arshad, M., Silaghi, M.: Distributed simulated annealing and comparison to DSA. In: Proceedings of the fourth international workshop on distributed constraint reasoning (2003)
2. Arslan, G., Marden, J., Shamma, J.: Autonomous vehicle-target assignment: A game-theoretical formulation. Journal of Dynamic Systems, Measurement, and Control 129, 584 (2007)
3. Černỳ, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of optimization theory and applications 45(1), 41 51 (1985)
4. Chapman, A., Rogers, A., Jennings, N., Leslie, D.: A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems. Knowledge Engineering Review 26(4), 411 444 (2011)
5. Chapman, A., Rogers, A., Jennings, N.: Benchmarking hybrid algorithms for distributed constraint optimisation games. Autonomous Agents and Multi-Agent Systems 22, 385414 (2011),
6. Farinelli, A., Vinyals, M., Rogers, A., Jennings, N.: Distributed Constraint Handling and Optimization. In: Weiss, G. (ed.) Multiagent Systems (Intelligent Robotics and Autonomous Agents) (2013)
7. Flueckiger, A., Verman, M., Bernstein, A.: Improving Approximate Algorithms for DCOPs Using Ranks, International Workshop on Optimisation in Multi-Agent System (2016)
8. Hart, S., Mas-Colell, A. 2000. A simple adaptive procedure leading to correlated equilibrium. Econometrica 68, 1127 1150
9. Hirayama, K., Yokoo, M.: Distributed partial constraint satisfaction problem. In: G. Smolka (Ed.), Principles and Practice of Constraint Programming. pp. 222 236 (1997)
10. Hirayama, K., Yokoo, M.: An approach to over-constrained distributed constraint satisfaction problems: Distributed hierarchical constraint satisfaction. In: Proceedings of International Conference on Multiagent Systems (2000)
11. Kirkpatrick, S., Vecchi, M., et al.: Optimization by Simulated Annealing. In: Science 220(4598), 671 680 (1983)
12. Léauté, T., Ottens, B., Szymanek, R.: FRODO 2.0: An open-source framework for distributed constraint optimization. In: Proceedings of the IJCAI09 Distributed Constraint Reasoning Workshop (DCR09). pp. 160 164. Pasadena, California, USA (July 13 2009), http://frodo2.sourceforge.net
13. Liu, J., Sycara, K.: Exploiting problem structure for distributed constraint optimization. In: Proceedings of International Conference on Multi-Agent Systems (1995)
14. Lutati, B., Gontmakher, I., Lando, M., Netzer, A., Meisels, A. and Grubshtein, A.: AgentZero: A Framework for Simulating and Evaluating Multi-agent Algorithms. In Agent-Oriented Software Engineering pp. 309 - 327. Springer Berlin Heidelberg (2014)
15. Magliacane, S., Stutz, P., Groth, P., Bernstein, A.: FoxPSL: An Extended and Scalable PSL Implementation. In: AAAI Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches. AAAI Spring Symposium, AAAI Press, Palo Alto, California (2015)
16. Maheswaran, R., Pearce, J., Tambe, M.: A family of graphical-game-based algorithms for distributed constraint optimization problems. Coordination of largescale multiagent systems pp. 127 146 (2006)
17. Maheswaran, R.T., Tambe, M., Bowring, E., Pearce, J.P., Varakantham, P.: Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1. pp. 310 317. IEEE Computer Society (2004)

18. Marden, J.R., Arslan, G., Shamma, J.S.: Joint strategy fictitious play with inertia for potential games. IEEE Transactions on Automatic Control 54(2), 208220 (2009)
19. Mazlami, G.: Scaling message passing algorithms for distributed constraint optimization problems in Signal/Collect. Bachelor Thesis, (2013)
20. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. In: Artificial Intelligence Journal. vol. 161, pp. 149 - 180. Elsevier (2005)
21. Parunak, V., Ward, A., Fleischer, M., Sauter, J., Chang, T.: Distributed component-centered design as agentbased distributed constraint optimization. In: Proceedings of the AAAI Workshop on Constraints and Agents (1997)
22. Schiex, T., Fargier, H., Verfaillie, G., et al.: Valued constraint satisfaction problems: Hard and easy problems. In: International Joint Conference on Artificial Intelligence. vol. 14, pp. 631 639. Citeseer (1995)
23. Schurgast, S.:Markov Logic Inference on Signal/Collect. Masters thesis, University of Zurich, Department of Informatics (2010)
24. Shoham, Y., Leyton-Brown, K.: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge Univ Pr (2009)
25. Stranders, R., Farinelli, A., Rogers, A., Jennings, N.: Decentralised coordination of mobile sensors using the max-sum algorithm. In: Proceedings of the 21st international jont conference on Artifical intelligence. pp. 299 304. Morgan Kaufmann Publishers Inc. (2009)
26. Strebel, D.: Scalable forensic transaction matching: and its application for detecting patterns of fraudulent financial transactions. Master thesis, University of Zurich, Faculty of Economics (Dec 2013)
27. Stutz, P., Bernstein, A., Cohen, W.: Signal/collect: Graph algorithms for the (semantic) web. The Semantic WebISWC 2010 pp. 764 780 (2010)
28. Stutz, P., Paudel, B., Verman, M., Bernstein, A.: Random-Walk TripleRush: Asynchronous Graph Querying and Sampling. In: 24th International World Wide Web Conference (2015)
29. Stutz, P., Strebel, D., Bernstein, A.: Signal/Collect: Processing Large Graphs in Seconds. Semantic Web Journal (2015)
30. Stutz, P., Verman, M., Fischer, L., Bernstein, A.: TripleRush: A Fast and Scalable Triple Store. In: 9th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2013). p. 50 (2013)
31. Sultanik, E., Lass, R., Regli, W.: DCOPolis: A Framework for Simulating and Deploying Distributed Constraint Reasoning Algorithms, Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)
32. Tel, G.: Introduction to distributed algorithms. Cambridge Univ Pr (2000)
33. Wahbi, Mohamed and Ezzahir, Redouane and Bessiere, Christian and Bouyakhf, El Houssine: DisChoco 2: A Platform for Distributed Constraint Reasoning, Proceedings of the IJCAI'11 workshop on Distributed Constraint Reasoning (2011)
34. Yokoo, M., Hirayama, K.: Distributed breakout algorithm for solving algorithm for solving distributed constraint satisfaction and optimization problems. In: Proceedings of the 2nd International Conference on Multiagent Systems (1996)
35. Yokoo, M., Ishida, T., Durfee, E., Kuwabara, K.: Distributed constraint satisfaction for formalizing distributed problem solving. In: Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on. pp. 614 621. IEEE (1992)
36. Zhang, W., Wang, G., Wittenburg, L.: Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance. In: Proceedings of AAAI Workshop on Probabilistic Approaches in Search (2002)
37. Zhang, W., Wang, G., Zhao, X., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks, In: Artificial Intelligence Journal. vol. 161, pp. 55 - 87. Elsevier (2005)