Constant cost of the computation-unit in efficiency graphs

Marius C. Silaghi[1], **Robert N. Lass**[2], **Evan A. Sultanik**[2], **William C. Regli**[2], **Toshihiro Matsui**[3] **and Makoto Yokoo**[4]
[1]Florida Tech, [2]Drexel University, [3]Nagoya Institute of Technology, [4]Kyushu University

Abstract

This article identifies and corrects a commonly held misconception about the scalability evaluation of distributed constraint reasoning algorithms. We show how to ensure a constant cost for the *computation-unit* in graphs depicting the number of (sequential) computation-units at different problem sizes. This is needed for a meaningful evaluation of scalability and efficiency, specially for distributed computations where it is an assumption of the measurement. We report empirical evaluation with ADOPT revealing that the computation cost associated with a constraint check (commonly used – and assumed constant – in ENCCCs evaluations) actually varies with the problem size, by orders of magnitude. This flaw makes it difficult to interpret such skewed graphs. We searched for methods to fix this problem and report a solution. We started from the hypothesis that the variation of the cost associated with a constraint-check is due to the fact that the most inner cycles of some common constraint solvers like ADOPT do not consist of constraint checks, but of processing search contexts (i.e., other data structures). We therefore propose *computation-units* based on a basket of weighted constraint-checks and context processing operations. Experimental evaluation shows that we obtain a constant cost of the computation-unit, proving the correctness of our hypothesis and offering a better methodology for efficiency and scalability evaluation.

Introduction

This article identifies and solves a wrong assumption with the scalability evaluation of distributed constraint reasoning algorithms. One of the major achievements of computer science consists of the development of the complexity theory for evaluating and comparing the efficiency of algorithms in a scalable way (Garey & Johnson 1979). Complexity theory proposes to evaluate an algorithm in terms of the number of times it performs *the operation in the most inner loop* or *the most expensive operation*. This number is analyzed as a function of the size of the problem. While such metrics do not reveal how much actual time is required for a certain instance, they allow for interpolating how the technique scales with larger problems. The assumption that computation speed doubles each few years makes a constant factor irrelevant from a long perspective.

For simple algorithms such as centralized sorting and graph traversal, *the most inner loop* is easily detected, and its operation is typically identified as the most expensive one. Identifying the most expensive operation and the most inner loop is not always trivial in distributed algorithms. Constraint reasoning researchers have long used either the *constraint check*, the *visited search-tree node*, or the *semijoin operation* as the *basic operation* in classical algorithms (Zhang & Mackworth 1991; Kondrak 1994). An advantage of the *constraint check* and *semijoin operation* for centralized problem solving is that cost associated with them is typically independent of the problem size, and that they often are really part of the most inner loop operations.

Evaluating distributed computing With distributed computing, the evaluation is particularly sensitive to the poor choice of the computation-unit. Main reasons are:

- the most inner loop is not easily revealed by the event-driven program structure, and it may consist of validating incoming data or local data, rather than of constraint checks or semijoin operations,
- the relative ratio between cost (latency) of messages varies by 4-6 orders of magnitude between multi-processors and remote Internet connections, and
- while the cost of a centralized computation can be expected to reduce over years, the cost (latency) of a message between two points is not expected to decrease significantly (in contrast with the other computation costs), since the limits of the current technology are already dictated by the time it takes light to traverse that distance over optical cable. Future increase in bandwidth can eventually remove only the congestion, leading to constant latency.

Indeed, the minimal time it can theoretically take a message to travel between two diametrically opposed points on the Earth is:

$$\frac{\pi * R_{Earth}}{speed_{light}} = \frac{3.14 * 6.378 * 10^6 m}{3 * 10^8 m/s} \approx 67ms.$$

Since the optical cables do not travel on a perfect circle around the Earth, it is reasonable to not expect significant improvements beyond the current some 150ms latency for such distances (other than possible elimination of congestion events due to increased bandwidth).

We start introducing a simple framework for unifying various versions of *logic time* systems. These previous methodologies are presented in the unifying framework. In particular, we show how to verify that a metric respects the assumption that its *unit (ordinate)* has the same meaning at different *evaluation points (abscissae)*, condition which with ADOPT we found not to be respected by the state of the art, but is respected by our proposal.

Previous research saying that problems of certain types/sizes are harder than problems of other types, based on counting common computational units, might have compared apples with oranges. The way in which algorithms scale with problem size might have also been significantly skewed, by orders of magnitude. We show how to verify if this happened, and how to avoid it.

Framework

Distributed Constraint Optimization (DCOP) is a formalism shown to model multi-agent scheduling problems, oil distribution problems, auctions, or distributed control of red lights in a city (Modi & Veloso 2005; Walsh 2007; Petcu, Faltings, & Parkes 2007).

DEFINITION 1 (DCOP). *A distributed constraint optimization problem (DCOP), is defined by a set A of agents $\{A_1, ..., A_n\}$, a set X of variables, $\{x_1, ..., x_n\}$, and a set of functions (aka constraints) $\{f_1, ...f_i, ..., f_m\}$, $f_i : X_i \rightarrow \boldsymbol{R}_+$, $X_i \subseteq X$, where only some agent A_j knows f_i.*
The problem is to find $argmin_x \sum_{i=1}^{m} f_i(x_{|X_i})$.

DCOPs restricting the output of the functions f_i by defining them as $f_i : X_i \rightarrow \{0, \infty\}$, are called Distributed Constraint Satisfaction Problems (DisCSPs).

Evaluation for MIMD.

Some of the early works on distributed constraint reasoning were driven by the need to speed up computations on multiprocessors, in particular (multiple instruction multiple data) MIMD architectures (Zhang & Mackworth 1991; Collin, Dechter, & Katz 2000; Kasif 1990), sometimes even with a centralized command (Collin, Dechter, & Katz 2000). However, their authors pointed out that those techniques can be applied straightforwardly for applications where agents are distributed on Internet.

The metric proposed by Zhang and Mackworth is based on Lamport's logic clocks described in the Definition 6.1 and in the Algorithm 18 in (Zhang & Mackworth 1991).

Certain authors use random values for the logic latency of a message (Fernàndez *et al.* 2002) and therefore we allow this in our unifying framework by specifying a number series generator (NSG) R_L from which each message logic time (logic latency) is extracted with a function $next()$. A *logic time system* we will use here is therefore parametrized as $LT\langle R_L, E, T\rangle$ where E is a vector of types of local events and T a vector of logic costs, one for each type of event. For measurements assuming a constant latency of messages set to a value L, the R_L parameter used consists of that particular number, **L**, (written in bold face).

Some reported experiments (Yokoo *et al.* 1992) use simultaneously several logic time systems, $LT^1\langle R_L^1, E^1, T^1\rangle, ..., LT^N\langle R_L^K, E^K, T^K\rangle$ (see Algorithm 1). Each agent A_i maintains a separate logic clock, with times LT_i^u, for each $LT^u\langle R_L^u, E^u, T^u\rangle$. Also, to each message m one will attach a separate tag $LT^u(m)$ for each maintained logic time system $LT^u\langle R_L^u, E^u, T^u\rangle$. This is typically done in order to simultaneously evaluate a given algorithm, and set of problems, for several different scenarios (MIMD, LAN, remote Internet).

```
% $R_L$ is the number series generator from which message latencies are extracted using function $next()$
% $E = \{e_1, ..., e_k\}$ is a vector of $k$ local events
% $T = \{t_1, ..., t_k\}$ is a vector of (logic) costs for events $E$
when event $e_j$ happens do
    $LT_i = LT_i + t_j;$
when message $m$ is sent do
    $LT(m) = LT_i + next(R_L));$
when message $m$ is received do
    $LT_i = max(LT_i, LT(m));$
```

Algorithm 1: Lamport's logic time (LT) maintenance for A_i. Parameters $\langle R_L, E, T\rangle$ unify previous versions.

A common metric used to evaluate simulations of DCR algorithms is given by the *logic time to stability* of a computation. The logic time to stability is given by the highest logic time of an event occurring before *quiescence* is reached (Zhang & Mackworth 1991); *Quiescence* of an algorithm execution is the state where no agent performs any computation related to that algorithm and no message generated by the algorithm is traveling between agents.

Uses of logic time for multiprocessors.

The operation environment targeted by (Zhang & Mackworth 1991) consists of a network of transputers. The metric employed there with simulations for a constraint network with ring topology is based on the logic time system $LT\langle\mathbf{1}, \{semijoin\}, \{1\}\rangle$, where the number series generator **1** outputs the value 1 at each call to $next()$. Note that the single local event associated there with a cost is the $semijoin$. Their results depict *logic time to stability* vs *problem size as (log scale) number of variables,* and *logic time* vs. *number of processors (aka agents) at a given size of the DisCSP* distributed to those agents (see Entries LTS1 and LTS2 in Table 1).

A theoretical analysis of the time complexity of a DisCSP solver is presented in (Collin, Dechter, & Katz 2000). Logic time analysis is presented there under the name *parallel time*, targeting MIMD multiprocessors, where each value change (aka *visited search-tree node* in regular CSP solvers) has cost 1. Note that the obtained metric is $LT\langle\mathbf{0}, \{value\text{-}change\}, \{1\}\rangle$, where message passing is considered instantaneous. A sequential version of the same algorithm is also evaluated in (Collin, Dechter, & Katz 2000) using the logic time $LT\langle\mathbf{0}, \{value\text{-}change, privilege\text{-}passing\}, \{1, 1\}\rangle$. The term coined in (Kasif 1990) for a similar theoretical analysis is *sequential time*.

NB	coordinate axis (Oy)	ordinates axis (Ox)	example usage
LTS1	(logic) time to stability (latency=0)	log – ring size –	(Zhang & Mackworth 1991)
LTS2	speedup – size 800 –	number of processors	(Zhang & Mackworth 1991)
TSL	number of time steps (aka ENCCCs)	message delay (time steps)	(Yokoo *et al.* 1992)
ECL	(equivalent) checks (aka ENCCCs)	checks/message (w. lat. 0)	(Silaghi, Sam-Haroud, & Faltings 2000a)
NCT	NCCCs (ENCCCs latency=0)	(constraint) tightness	(Meisels *et al.* 2002)
ECT	ENCCCs (at fix checks/message)	(constraint) tightness	(Chechetka & Sycara 2006)
ST	seconds	constraint tightness	(Hamadi & Bessière 1998)
CT	#checks	constraint tightness	(Hamadi & Bessière 1998)
MT	#messages	constraint tightness	(Hamadi & Bessière 1998)
CBR	checks	constraint tightness	(Davin & Modi 2005)

Table 1: Summary of the systems of coordinates used for comparing efficiency of distributed constraint reasoning.

Evaluation for applications targeting the Internet.

Distributed constraint reasoning algorithms targeting the Internet had to account for the possibly high cost of message passing between agents on remote computers. As mentioned earlier, the theoretical lower bound on latency between diametrically opposed point on the Earth is 67ms, eight orders of magnitude larger than a basic operation on a computer (of the order of 1ns).

One of the first algorithms specifically targeting Internet is the Asynchronous Backtracking in (Yokoo *et al.* 1992). That work used simultaneously a set of different logic times, $LT^1, ..., LT^{25}$, where LT^i is defined by parameters

$$LT^i\langle \mathbf{i}, \{constraint\text{-}check\}, \{1\}\rangle \qquad (1)$$

(Yokoo *et al.* 1992) reports the importance of the message latency in deciding which algorithm is good for which task. A curve in their type of graph (see Entry TSL in Table 1) reports several metrics, but for a single problem size/type. The *time steps* introduced in (Yokoo *et al.* 1992) correspond to the cost associated with a constraint check. A similar results graph is used in (Silaghi, Sam-Haroud, & Faltings 2000b) having as axes checks vs checks/message, i.e., the logic time cost for one message latency when the unit is the duration associated with a constraint check (see Entry ECL in Table 1). This last graph also reports logic time for the latency $L = 0$

$$LT^0\langle \mathbf{0}, \{constraint\text{-}check\}, \{1\}\rangle, \qquad (2)$$

which corresponds to simulation of execution with agents placed on the processors of a MIMD with very efficient (instantaneous) message passing (similar to (Collin, Dechter, & Katz 2000), but using the constraint check as the logic unit). This particular metric is sometimes referred to as *the number of non-concurrent constraint checks*.

Cycles.

After the work in (Yokoo *et al.* 1992), most DCOP research focused on agents placed on remote computers with problem distribution motivated by privacy (Yokoo *et al.* 1998). Due to the small ratio between the cost of a constraint check and the cost of one message latency in Internet, the standard evaluation model selected in many subsequent publications completely dropped the accounting of constraint checks. A common assumption adopted for evaluation is that local computations can be made arbitrarily fast (local problems are assumed small and an agent can make his local computation on arbitrarily fast supercomputers). Instead, message latency between agents is a cost that cannot be circumvented in environments distributed due to privacy constraints. Such a metric used in (Yokoo *et al.* 1998) is:

$$cycles = LT\langle \mathbf{1}, \emptyset, \emptyset\rangle$$

This metric is sometimes referred to as *number of sequential messages (SMs)* or *longest causal chain of messages*. The original name for this metric is *cycles*, based on the next theorem (known among some researchers but not written down in this context).

THEOREM 1. *In a network system where all messages have the same constant latency L and local computations are instantaneous, all local processing is done synchronously, only at time points kL (in all agents).*

PROOF. One assumes that all agents start the algorithm simultaneously at time L, being announced by a broadcast message, which reaches all agents at exactly time L (due to the constant time latency). Each agent performs computations only either at the beginning, or as a result of receiving a message.

Since each computation is instantaneous, any message generated by that computation is sent only at the exact time when the message triggering that computation was received. It can be noted that (**induction base**) any message sent as a result of the computation at the start will be received at time L, since it takes messages L logic time units after the start to reach the target.

Induction step: All the messages that leave agents at time kL, will reach their destination at exactly time $(k + 1)L$ (due to the constant latency L). Therefore the observation is proven by induction. □

As a consequence of this observation, any network simulation respecting these assumptions (that local computations are instantaneous and that message latencies are constant) can be performed employing a loop, where at each cycle each agent handles all the messages sent to it at the previous cycle. $LT\langle \mathbf{1}, \emptyset, \emptyset\rangle$ is given by the total number of cycles of this simulator. However, the assumption that local computations can be considered instantaneous with respect to latency does not apply to some algorithms performing significant local processing (Petcu & Faltings 2005).

NCCCs and ENCCCs.

Researchers voiced concerns[1] about the lack of accounting for local computation in SMs. A subsequent re-introduction of logic time in the form of the metric in Equation 2 is made in (Meisels *et al.* 2002), proposing to build graphs with axes labeled *NCCCs (non-concurrent constraint checks)* versus problem type (Entry NCT in Table 1).

REMARK 1. *One infers the assumption that the cost associated with a constraint check (NCCC) is the same at different problem sizes!*

In this approach the cost of a message is typically restricted to only 0, reporting solely constraint checks, as in (Collin, Dechter, & Katz 2000).

However, the importance of the latency of messages has also been rediscovered recently and logic time cost for message latency is reintroduced in (Chechetka & Sycara 2006) under the name Equivalent *Non-Concurrent Constraint Checks (ENCCCs).* ENCCCs are computed using the Equation 1. Current ENCCCs usage in graphs typically differs from earlier usage of the metric by being depicted versus *constraint tightness* or versus *density of constraint-graph* (with a label specifying the value of the logic latency L, i.e. the number of checks/message-latency). Each graph depicts the behavior of several problem types for one message latency, rather than the behavior of one problem type for several message latencies (Entry ECT in Table 1).

REMARK 2. *One infers the assumption that the cost associated with a constraint check (ENCCC) is the same at different problem sizes! More exactly, it assumes that the ratio between message latency and the time taken for a constraint check is constant.*

Evaluations not related with the logic time.

Three other important metrics (not based on logic time) for evaluating DCOPs algorithm were introduced in (Hamadi & Bessière 1998) in conjunction with a DisCSP solver.

- the total running time in seconds (Entry ST of Table 1);
- the total number of constraint checks for solving a DisCSP (or DCOP) with a simulator (see Entry CT of Table 1), and
- the total number of exchanged messages (Entry MT of Table 1).

Cycle-based runtime (CBR) gives the ENCCCs on a modified version of the algorithm, which adds synchronizations before sending each message (Davin & Modi 2005).

Reporting the experimental setting for reproduction by peer researchers is done by giving the specification of the used computers, the distribution of agents to computers and the type of the network. The time used per constraint check (averaged over all problems) can also be reported (Silaghi & Yokoo 2007b; Lass *et al.* 2007), to help determining the ENCCCs' constraint-check to latency ratio relevant to a given application scenario. Congestion can lead to a large variance in the value of latency, but one can provide an analysis for a future where increase in bandwidth would remove congestion. The ENCCCs at such a latency/checks ratio are called *operation point* ENCCCs (ENCCC-OPs), shown in Figure 1. Equivalent message latencies in the operating point (EML-OP) show the number of (equivalent) message delays and are also proportional with the *logic simulated seconds*.

[1]Debate at the CP 2001 conference.

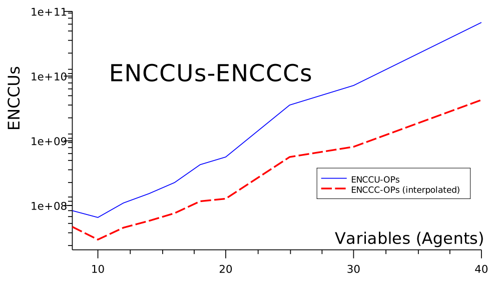

Figure 1: ADOPT performance: operating point ENCCUs.

Constant Computational Unit

We studied the common DCOP solver ADOPT (Modi *et al.* 2005) in more detail and we observed that the cost associated with a constraint check is not constant over different problem sizes, as assumed (see earlier Remarks). This can be seen following the ms/check curve in the graph in Figure 2. In order to verify whether a chosen computation-unit (e.g. the constraint check) has a constant cost/meaning at different problem sizes in DCOPs, we follow the following methodology.

Assumption Verification.

1. Compute the total execution time in seconds, t_p, for solving each complete test set of problems at size p using a simulator (or a real execution where message latencies are factored out, and where local computation times are summed up).
2. Compute the total number of basic computation-units $\#CU_p$ (e.g. constraint checks, $\#CC_p$), at each problem size p (Hamadi & Bessière 1998).
3. Compute the cost in seconds that should be associated with a computation-unit by computing the ratio $t_p/\#CU_p$. Verify that the computation unit was correctly selected by checking that this ratio is constant with the problem size.

We note that for the previous ways of evaluating DCOP solvers on a given machine and programming language, this cost depends on the problem size p, varying as much as an order of magnitude. For example, our C simulator for ADOPT on the problems in the Teamcore data-set uses between 3 to 28 microseconds associated with each constraint check on a Linux PC at 700MHz. The smaller value was found at problems with 8 agents and 8 variables and the larger one at problems with 40 agents and variables.

$$LT^i\langle \mathbf{i}, \{constraint\text{-}check, nogood\text{-}inference, nogood\text{-}validity, nogood\text{-}applicability\}, \{1,3,2,2\}\rangle, \forall i > 0 \qquad (3)$$

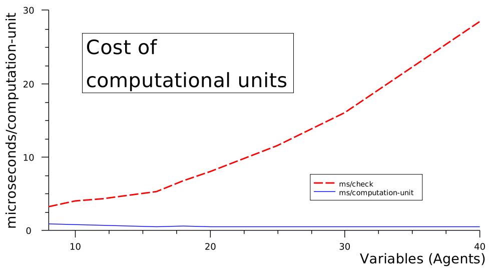

Figure 2: The running time associated with a computation-unit for two metrics: checks and CUs.

Our hypothesis is that the failure of the assumption of constant time constraint checks is due to the fact that checks are not representative for the most inner loop(s) operations in ADOPT. We try to identify other operations that weighted will lead to a computation unit (CU) which is closer to constant across problem sizes. Having constant cost across problem sizes is a good indication of the fact that a logic metric represents the most inner loop(s) of an algorithm.

Accounting for context, auxiliary data, and nogood processing.

Certain DCOP algorithms are not based on checking constraints repeatedly, but rather they compile information about constraints into contexts and new entities, such as *nogoods*. Afterward, these techniques work by performing inferences on such entities. Contexts, agent views, and nogoods are a kind of constraints themselves. In such algorithms it makes sense to attribute costs to the different important operations on these entities such as *nogood inference*, *nogood validity check*, and *nogood applicability check*. A new method for computing logic times at various message latencies in nogood-based techniques is the Equation 3, where the coefficients of different nogood handling operations are selected based on a perceived complexity for those operations. The nogood inference operation is typically the most complex of these operations as it accesses two nogoods to create a third one (suggesting a logical cost of 3). Nogood-validity and nogood-applicability both typically involve the analysis of a nogood and of other data, local assignments and remote assignments, to be compared with the nogood (hence a logical cost of 2). In some implementations these costs do not have an exact value if the sizes of nogoods vary within the same problem. In our experiments with ADOPT, all the messages and context (nogood) have the same size. A constraint check for binary constraints is cheaper than the verification of an average-sized nogood, and is given the logical cost of 1. Experimentation reported here shows that the CU obtained this way is practically constant across problem sizes.

Why cost associated with checks varies with the problem size.

The cost associated with a constraint check (as measured above) consist of an aggregation of the costs of all other operations executed by DCOP algorithm in preparation of the constraint check and in processing the results of the constraint check. Typically there are several data structures to maintain and certain information to validate, and these data structures may be larger with large problem sizes than with small problem sizes.

In certain situations, algorithms change their relative behavior in situations that are close to the operating point. Then precise measurements are important, and it makes sense to try to tune the logic time associated with each operation, in order to reduce the variation of the meaning of a unit of logic time with the problem size. One can approach this problem by trying many different combinations, or trying a hill climbing approach that tunes successively each of the parameters. One has to run complete sets of experiments for each of these possible costs (which is computationally expensive). A valuable future research direction consists in finding an efficient way of tuning these parameters.

We selected weights as described earlier for operations related nogood processing of a valued nogood-based implementation of ADOPT (Silaghi & Yokoo 2006), and were able to obtain such a constant cost for the obtained computational unit as reported in the Experiments section.

Generality of our conclusions.

Note that we do not claim (and do not think) that the computation unit proposed in Equation 3 would necessarily fit any algorithm, other than ADOPT. Our contribution is to detect the problem with current practice, and to examplify how to find computation units which render experimental results both verifiable by peers and meaningful from the scalability perspective.

When two different algorithms are compared, this may be done for one of the following two goals:

- **To find which technique scales better with the problem size.**

  For this goal, each algorithm should be analyzed (and depicted) as a function of a computation unit that is appropriate for it (Figure 1), where only *the order of growth* of the curves is relevant, not the relative position.

- **To find which algorithm is better on a given set of problems.**

  For this goal, the efficiency of each algorithm should be analyzed in terms of a general metric, such as the EML-OP or *logic simulated seconds* described above (see Figure 3). Those two metrics allow to comparable different algorithms in a way that is verifiable by peers.

Experiments

The experiments are based on a sample of Teamcore random graph coloring problems with 10 different sizes, ranging between 8 agents and 40 agents, with graph density 30%. The results are averaged over 25 problems of

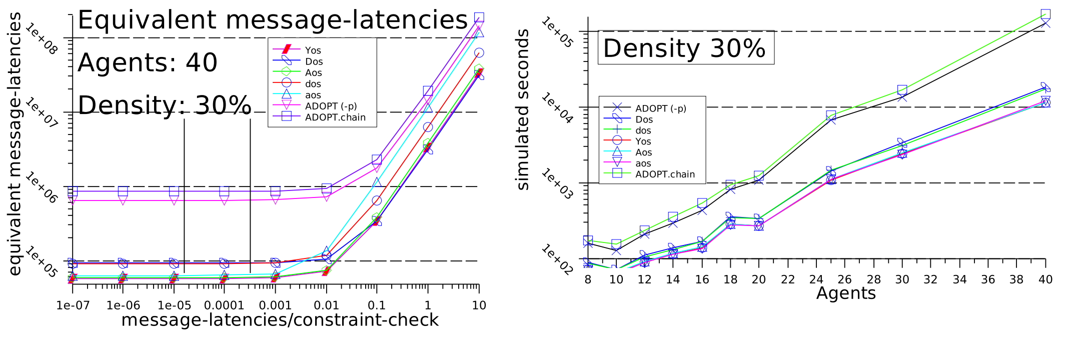

Figure 3: Example of graph showing a) EML-OPs; b) logic simulated times; for the versions of ADOPT in (Silaghi & Yokoo 2007a). The vertical bars in the EMLgraph show the range of the operating point.

p (agents)	8	10	12	14	16	18	20	25	30	40
t_p (total seconds)	0.1404	0.1528	0.3012	0.5516	1.0068	2.5708	4.1176	47.7112	174.06	3767.38
$\#CC_p$ total checks	43887	38279	70279	116080	191501	381415	516835	4.1*10^6	10.9*10^6	132*10^6
$\frac{microseconds(t_p)}{check(\#CC_p)}$	**3.199**	**3.992**	**4.286**	**4.752**	**5.257**	**6.74**	**7.967**	**11.47**	**15.98**	**28.4**
L_p= $\frac{checks}{latency(200ms)}$	62518	50103	46666	42088	38041	29672	25103	17437	12519	7041.1
(10^6) ENCCC L=10^4	7,94	6,32	10,5	14,8	21,6	41,8	54,1	343	694	6594
(10^6) ENCCC L=10^5	79	63	105	148	216	417	541	3429	6939	65880
$\#CU_p$ (10^6)	0.173	0.205	0.473	1.02	2.067	5.210	8.673	100	396	8521
$\frac{microseconds(t_p)}{comp-unit(\#CU_p)}$	**0.81**	**0.743**	**0.636**	**0.54**	**0.487**	**0.493**	**0.475**	**0.476**	**0.439**	**0.442**
ENCCU L=10^5 (10^6)	79	63	105	148	216	418	542	3434	6953	66096

Table 2: Sample re-evaluation of ADOPT with our method. Columns represent problem size.

each size (Modi *et al.* 2003). The targeted application scenario consists of remote computers on Internet. The catalog message latency for our scenario is 200ms, varying between 150ms and 250ms (Neystadt & Har'El 1997; Kopena *et al.* 2004).

Following the steps of the discussed method we report the following results using ENCCCs evaluation:

1. Simulated ADOPT with randomized latencies is implemented in C++ and runs on a the 700MHz node of a Beowulf (Linux Red Hat). The total time in seconds is given in the second row of Table 2.

2. The total number of constraint checks $\#CC_p$ for each problem size is given in the third row of Table 2.

3. The cost in (micro)seconds associated with each constraint check is computed as $t_p/\#CC_p$. It is given in the fourth row of Table 2.

The message-latency/constraint-check ratio (L_p) is computed by dividing the average latency found at Step 1 (200ms) by the items in the 4^{th} row. The results are given in the 5^th row of Table 2.

We report results with logic time systems where the checks/latency ratio are $L = 100,000$ and where it is $L = 10,000$. In is now possible to re-run the experiments with all the L_p values found in our table. Here we report the results of the closest L, which is 10,000 for most problem sizes (one also can use $L = 100,000$ for problems with 8 and 10 agents), see the 6^{th} and 7^{th} rows of Table 2. We also interpolate the time between the predictions based on $L = 10,000$ and $L = 100,000$, function of the predicted L_p at each problem size (obtaining the graph in Figure 1).

It is remarkable that the cost associated with constraint checks varies so strongly with the problem size even for the same implementation of the same algorithm. We felt that it is good to verify this observation on a different implementation, and in particular on a LAN solver. We ran a set of experiments using the JAVA based DCOPolis platform (Lass *et al.* 2007). Here the agents are distributed on five HP-TC4200 tablet PCs with 1.73Ghz Intel Pentium M processors and 512M of RAM connected via Ethernet to a Netgear FS108 switch, isolated from the Internet and running Ubuntu Linux (see Figure 4), and reveal similarly large variance.

The next part in Table 2 gives the ENCCU, i.e. ENCCCs with the computational unit based on counting context processing as described in Equation 3. The cost of a CU is given in the last but one row. The CUs/latency ratio at latency 200ms ratio is $5 * 10^5$. Note that the cost of CUs is practically constant across problem sizes, with an increase at very small problems (reflecting the predictable impact of the overhead due to initializing data structures at the beginning of the computation). The overhead producing variability at low problem sizes may be taken into account adding additional costs for initializations of data structures. This shows that with ADOPT, the selected computation-unit sat-

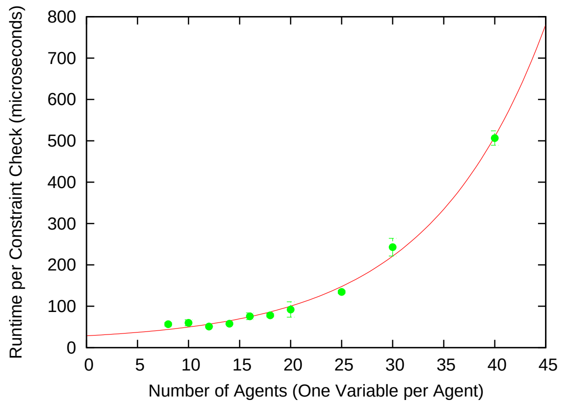

Figure 4: Results with DCOPolis on a LAN

isfies the assumption of constant time better than the constraint checks. The impact of the CU choice on ENCCUs becomes stronger at $L < 10^3$.

Conclusion

We started introducing a framework for enabling an unified representation of different logic clocks-based metrics used for efficiency evaluation of DCOPs. We show that all major metrics used in the past for evaluating algorithms for DCOPs fit into this framework. We identify a wrong assumption with currently common metrics. Then we propose a methodology to analyze whether the computational unit was well selected to have a common cost for all problem sizes.

We discuss remarkable experimental observations showing that state of the art metrics for evaluating ADOPT fail to satisfy an essential assumption, namely that the meaning of the metric used for comparison is the same at different points (problem sizes). We found that the cost associated with constraint checks can vary widely with the size of the problem even for the same implementation of the same algorithm. The conclusion is that some of the current DCOP research on scalability of algorithms with the problem size may *compare apples to oranges*, and may be strongly skewed. We discuss the possible explanations, their implications, and how the issue can be fixed. A solution we propose is based on selecting *computational units* that account for a weighted count of several common operations, such as context validation, and inferences. Experimentation shows that such a selection leads to a closer to constant cost of the computational unit across problem sizes, which is an important assumption normally expected from common evaluation graphs.

Appendix

Random message latencies.

Some researchers report that the introduction of random latencies has a strong impact on the efficiency for certain DCOP algorithms (Fernàndez *et al.* 2002). We have extensively experimented and we have found random latencies to have only around 5% impact (in both directions) on the number of sequential messages for the ADOPT (Modi *et al.* 2003) algorithm. In order to allow other researchers to replicate such experiments we propose to publish the seed with which we initialized the used random number generator (e.g., in our case the C library random number generator with seed 10000), as well as the equation used to distribute the latency uniformly in the range of expected latencies for the targeted application (in our case uniformly between 150ms and 250ms).

In Table 3 we show data for comparing between randomized versus constant latencies in simulation.

References

Chechetka, A., and Sycara, K. 2006. No-commitment branch and bound search for distributed constraint optimization. In *AAMAS*.

Collin, Z.; Dechter, R.; and Katz, S. 2000. Self-stabilizing distributed constraint satisfaction. *Chicago Journal of Theoretical Computer Science*.

Davin, J., and Modi, P. J. 2005. Impact of problem centralization in distributed COPs. In *DCR*.

Fernàndez, C.; Béjar, R.; Krishnamachari, B.; and Gomes, C. 2002. Communication and computation in distributed CSP algorithms. In *CP*, 664–679.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W.H.Freeman&Co.

Hamadi, Y., and Bessière, C. 1998. Backtracking in distributed constraint networks. In *ECAI'98*, 219–223.

Kasif, S. 1990. On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks. *Artificial Intelligence* 45(3):275–286.

Kondrak, G. 1994. A theoretical evaluation of selected backtracking algorithms. Master's thesis, Univ. of Alberta.

Kopena, J.; Nail, G.; Peysakhov, M.; Sultanik, E.; Regli, W.; and Kam, M. 2004. Service-based computing for agents on disruption and delay prone networks. In *AAMAS*, 1341–1342.

Lass, R. N.; Sultanik, E. A.; Modi, P. J.; and Regli, W. C. 2007. Evaluation of cbr on live networks. In *DCR Workshop at CP*.

Meisels, A.; Kaplansky, E.; Razgon, I.; and Zivan, R. 2002. Comparing performance of distributed constraints processing algorithms. In *AAMAS02 DCR Workshop*, 86–93.

Modi, P., and Veloso, M. 2005. Bumping strategies for the multiagent agreement problem. In *AAMAS*.

Modi, P.; Tambe, M.; Shen, W.-M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. In *AAMAS*.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *AIJ* 161.

Neystadt, J., and Har'El, N. 1997. Israeli internet guide (iguide). http://www.iguide.co.il/isp-sum.htm.

p (agents)	8	10	12	14	16	18	20	25	30	40
SMs (random)	793.2	631.48	1045.12	1480.36	2158	4172.56	5411.36	34284.2	69383.64	658731.36
SMs	736.16	602.76	1020.24	1438.64	2109.24	4125	5480.84	34067.2	68976.36	650307.08
total time (sec)	0.104	0.098	0.2776	0.4872	0.9636	2.55	4.1612	49.3452	181.2136	3813.5016
total checks	44637	40537	76936	126979	215303	428259	592419	4729413	12523233	151307539
μs/check	2.33	2.417	3.608	3.837	4.475	5.95	7.024	10.4	14.47	25.2
checks/latency	85841.9	82728.9	55429.8	52125.9	44687.2	33588.9	28473.5	19168.7	13821.5	7935.36

Table 3: ADOPT. First row is for randomized latency. Remaining rows are with constant message latency.

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization. In *IJCAI*.

Petcu, A.; Faltings, B.; and Parkes, D. C. 2007. M-dpop: Faithful distributed implementation of efficient social choice problems. *submitted to JAIR*.

Silaghi, M.-C., and Yokoo, M. 2006. Nogood-based asynchronous distributed optimization. In *AAMAS*.

Silaghi, M.-C., and Yokoo, M. 2007a. ADOPT-ing: Unifying asynchronous distributed optimization with asynchronous backtracking. `http://www.cs.fit.edu/~msilaghi/papers/ADOPTing-detailed.pdf`.

Silaghi, M.-C., and Yokoo, M. 2007b. Dynamic DFS tree in ADOPT-ing. In *Twenty-Second AAAI Conference on Artificial Intelligence (AAAI)*.

Silaghi, M.; Sam-Haroud, D.; and Faltings, B. 2000a. Asynchronous search with aggregations. In *Proc. of AAAI2000*, 917–922.

Silaghi, M.-C.; Sam-Haroud, D.; and Faltings, B. 2000b. Asynchronous search with aggregations. In *Proc. of AAAI2000*, 917–922.

Walsh, T. 2007. Traffic light scheduling: a challenging distributed constraint optimization problem. In *DCR*.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1992. Distributed constraint satisfaction for formalizing distributed problem solving. In *ICDCS*, 614–621.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE TKDE* 10(5):673–685.

Zhang, Y., and Mackworth, A. K. 1991. Parallel and distributed algorithms for finite constraint satisfaction problems. In *Third IEEE Symposium on Parallel and Distributed Processing*, 394–397.