Distributed Simulated Annealing and comparison to DSA —trade-offs between 'anytime' and marginally improved quality, local effort and communication*—

Muhammad Arshad† and Marius C. Silaghi
Florida Institute of Technology
{marshad,msilaghi}@cs.fit.edu

Abstract

Distributed Constraint Satisfaction is a framework for modeling and solving distributed problems. Research on the topic intensified during the last ten years when mainly complete techniques were thoroughly explored. A revival of attention for Distributed Stochastic Algorithms (DSA) was marked by the work of (Fabiunke 1999; Fitzpatrick & Meertens 2001; Zhang, Wang, & Wittenburg 2002). Remarkably, it was proven that Distributed Stochastic Search is a competitive technique. Here we review current DSAs and their intrinsic problems. We also describe solutions and a new algorithm, Distributed Simulated Annealing (DSAN). We show theoretically how these solutions can improve the robustness of DSA. Experimental evaluation on synchronous versions shows that with current heuristics only DSAN competes with the previous winner, DSA-$\overline{B}$. DSAN offers marginally better quality solutions than DSA-$\overline{B}$ for hard and over-constrained problems. For easy problems DSAN may not reach the solution if the temperature doesn't descend to zero.

For very remote agents, when only the cost of communication matters, DSA-$\overline{B}$ has a better 'anytime' behavior than DSAN. The local effort of an agent in DSA-$\overline{B}$ is higher than in DSAN with a factor linear in the domain size.

Introduction

Constraint satisfaction has proven to be a successful paradigm for approaching combinatorial problems like resource allocation, scheduling, or planning in centralized settings. A constraint satisfaction problem (CSP) is given by:

- a set of *variables* $\{x_1, x_2, ..., x_n\}$,
- a set of *domains*, $\{D_1, D_2, ..., D_n\}$, associated with the variables, and
- a set of *constraints*, $C_1, C_2, ..., C_k$, each of them involving a subset of the set of variables,

The *solution* to a CSP is an assignment of values from the corresponding domains to each variable such that the combination of assigned values is allowed by each constraint. A *distributed* CSP (DCSP) arises when variables are distributed among agents so that each variable can only be assigned values by a single agent (Yokoo *et al.* 1992). This is the definition exploited in our technique, even if DSA/DSAN can be easily extended to other frameworks of distributed CSPs, notably where assignments for each variable can be proposed by several agents (Silaghi, Sam-Haroud, & Faltings 2000).

Distributed Constraint Satisfaction can model and solve naturally distributed problems. Research on DCSPs has accelerated during the last years when complete techniques as well as techniques based on tight organization were thoroughly explored. Attention was drawn to Distributed Stochastic Algorithms (DSA) by the work of (Fabiunke 1999; Fitzpatrick & Meertens 2001; Zhang, Wang, & Wittenburg 2002). Remarkably, it was reported in (Zhang & Wittenburg 2002) that Distributed Stochastic Search performs better on coloring graphs compared with Distributed Breakout, previously known as particularly efficient. Many researchers directed their attention towards DSA. Five versions of DSA are compared in (Zhang, Wang, & Wittenburg 2002): DSA-A, DSA-B, DSA-C, DSA-D, and DSA-E. DSA-B was found to be the best in this family.

Here we review existing DSAs and address their intrinsic problems. DSA has no chance to get out of local minima. We also describe solutions and show theoretically how these solutions can improve the robustness of DSA. Experimental evaluation on synchronous versions shows that with current heuristics only DSAN competes with the previous winner, DSA-$\overline{B}$. DSAN offers marginally better quality solutions than DSA-$\overline{B}$ for hard and over-constrained problems. For easy problems DSAN may not reach the solution if the temperature doesn't descend to zero.

For very remote agents, when only the cost of communication matters, DSA-$\overline{B}$ has a better 'anytime' behavior than DSAN. The local effort of an agent in DSA-$\overline{B}$ is higher than in DSAN with a factor linear in the

*Most content of this paper was taught or discussed in AI classes at FIT during the 2002 Fall Semester, and the first version was available on Nov 8, 2002. Grigore Borota worked on a partial implementation of DSAN in December 2002. We thank anonymous reviewers of FLAIRS03 for encouragements, despite the fact that one errouneously believed DSAN would have been already proposed and studied in (Aarts & Korst 1989).

†Undergraduate student at FIT.

Cycle	A_1(NY,LA,W)	A_2(D,LA,P,W)	A_3(W)	A_2(LA,W,D)
1	NY	D	W	LA
	$(\Delta = 1 > 0) => W, LA(p)NY(1-p)$	$(\Delta = 1 > 0) => W, LA(p)D(1-p)$	$(\Delta = 0) => W$	$(\Delta = 1 > 0) => W, D(p)LA(1-p)$
	NY	LA	W	D
2	NY	LA	W	D
	$(\Delta = 1 > 0) => W, LA(p)NY(1-p)$	$(\Delta = 1 > 0) => D, W(p)LA(1-p)$	$(\Delta = 0) => W$	$(\Delta = 1 > 0) => LA, W(p)D(1-p)$
	LA	LA	W	W
3	LA	LA	W	W
	$(\Delta = 1 > 0) => W(p)LA(1-p)$	$(\Delta = 1 > 0) => W(p)LA(1-p)$	$(\Delta = 0) => W$	$(\Delta = 1 > 0) => LA(p)W(1-p)$
	W	LA	W	LA
4	W	LA	W	LA
	$(\Delta = 1 > 0) => LA(p)W(1-p)$	$(\Delta = 1 > 0) => W(p)LA(1-p)$	$(\Delta = 0) => W$	$(\Delta = 1 > 0) => W(p)LA(1-p)$
	W	W	W	W
5	W	W	W	W
	$(\Delta = 0) => W$	$(\Delta = 0) => W$	$(\Delta = 0) => W$	$(\Delta = 0) => W$
	W	W	W	W

Figure 1: Successful trace of the problem with DSA-B. For each cycle, the first row show the result of the exchange stage, the second row show the parameters for the (stochastic) decision step, and the third row show the outcome of the decision i.e. the chosen value.

domain size.

Distributed Stochastic Algorithm

The properties of distributed algorithms that express ideal behaviors are: uniformity, simplicity, robustness, and efficiency. Distributed stochastic search algorithms (DSA) offers three of these properties. Uniformity is the property of a distributed algorithm where all processes have equal priority for every act. This also implies that they do not need identities to distinguish one another for breaking ties. An uniform algorithm does not need a central authority and it is simple to launch or to restart (Tel 1999). There are very few known uniform algorithms as most algorithms are *almost uniform*, meaning that all but one of the processes are identical (Collin, Dechter, & Katz 2000).

Algorithm 1: The algorithm performed by all agents for DSA

```
procedure DSA do
    Randomly choose a value;
    while (no termination condition is met) do
        if (a new value is assigned) then
            send the new value to neighbors;
        end
        collect neighbors' new values, if any;
        select and assign the next value (See Figure 2)
    end
end do.
```

Second, DSA is a very simple algorithm (Zhang, Wang, & Wittenburg 2002). The single procedure of DSA is shown in Algorithm 1. The agents start picking random values for their variables. Then they enter a loop until the termination condition is met. At the beginning of each cycle, each agent sends its variable value to its neighboring agents. The value is sent only in the first cycle or if it was changed at the end of the previous cycle. Simultaneously each agent listens and gets the changes of the states, i.e. value assignments, of the neighbors. After all exchanges were made, each agent decides whether to change the current value for its variable. The decision can be taken either deterministically or stochastically. The values are changed according to a min-conflict behavior, aiming to reduce the number of constraints that are not satisfied.

The existing versions of DSA are synchronous. This means that the agents start each cycle in a synchronized way and also take a decision only after making sure that all exchanges of the current cycle have been terminated, e.g. by an additional synchronization. Note that each DSA cycle has two stages: value exchange, respectively decisions for changing values. These synchronizations actually mean that no message sent by an agent in a certain stage of a cycle reaches another agent in another cycle or even in another stage of the same cycle.

The existing versions of DSA differ in the way an agent decides the next value. The decision is based on its current state and the values received from the neighboring agents. The state of an agent is quantified by its current value and by the number of conflicts it knows between its value and other values. An agent only changes its value if this does not increase the number of conflicts it knows. If there exists a value that improves or maintains state quality, the agent may or may

not change to the new value. The decision is based on a stochastic scheme, depending on the chosen strategy. Figure 2 (Zhang, Wang, & Wittenburg 2002) shows the schemes defining the five existing versions of the DSA algorithm. C stands for conflict, Δ is the best possible conflict reduction between two steps, v is the value for which Δ was obtained. p is the probability to change the current value. It was notice that p has a relation to the intensity of the computation and therefore it is called *the degree of parallel executions*. "-" in the Figure 2 means that the value is not changed. $\Delta > 0$ implies that there exists a conflict, since an improvement is possible.

Here are the differences between the five versions:

A Whenever the current state can be improved, the change is made stochastically. Otherwise no change is made.

B DSA-B is like DSA-A but agents also change their values if they know conflicts and changing the values does not increase the number of conflicts.

C DSA-C allows agents to change their value in the same conflict conditions as DSA-B. Additionally. DSA-C allows the agents to change their value if there is no conflict and they introduce no conflict by their change.

D DSA-D is a version of DSA-B where the probability to move when $\Delta > 0$ is 1.

E DSA-E is a version of DSA-C where the probability to move when $\Delta > 0$ is 1.

Algorithm	$\Delta > 0$	C, $\Delta = 0$	no C, $\Delta = 0$
DSA-A	v with p	-	-
DSA-B	v with p	v with p	-
DSA-C	v with p	v with p	v with p
DSA-D	v	v with p	-
DSA-E	v	v with p	v with p

Figure 2: Alternatives of existing DSA differ in their value selection. Here is the exhaustive list of their details (Zhang, Wang, & Wittenburg 2002).

DSA-B has the best performance (Zhang, Wang, & Wittenburg 2002). This can be explained by the fact that agents may change their current state even if it does not bring improvements directly. Even if the improvements are not made directly, the change may enable a neighbor to find a descent out of the local minima. The agents may have been in a global state where two or more changes were required for an improvement and they do not require any increase of the conflicts in the intermediary steps. DSA-C tries to extend this behavior for states with no conflict but no improvement was found. DSA-D and DSA-E make an improvement whenever they can, but this was proven not to be wise. The fact that all agents make many changes in parallel with the two last versions, activates the conditions foreseen in (Collin, Dechter, & Katz 2000) to lead to long/infinite loops and used as argument against uniform techniques.

An asynchronous version of DSA

Algorithm 2: An asynchronous version of DSA

```
procedure DSA do
    Randomly choose a value;
    while (no termination condition is met) do
        if (a new value is assigned) then
            send the new value to neighbors;
        end
        set alarm;
        at alarm collect neighbors' new values
             received since the last alarm, if any;
        select and assign the next value;
    end
end do.
```

As mentioned before, one can easier understand DSA by analyzing it as a synchronous algorithm. Nevertheless, asynchronous versions can be obtained very easily. The Algorithm 2 gives an example where synchronizations are replaced with a timeout. It is assumed that an additional thread collects incoming messages while the agent is found in the second stage.

Local Minima

This section highlights an issue obvious for all AI researchers but that escapes students, mainly due to the fact that it isn't mentioned in other reports on DSA.

Let us now analyze the following problem with the best of the previous techniques, DSA-B: Four agents, A_1, A_2, A_3, and A_4 want to meet and will choose a city. Each of them has the constraint of wanting to be with the others: $x_1 = x_2$, $x_1 = x_3$, $x_1 = x_4$, $x_2 = x_3$, $x_2 = x_4$, $x_3 = x_4$. In fact A_1 only evaluates the constraints $x_1 = x_2$, $x_1 = x_3$, $x_1 = x_4$. Evaluating any of the other constraints could only switch us from the last column in Figure 2, no C, $\Delta = 0$, to the previous column: C, $\Delta = 0$. (Note that this would lead to behaving according to DSA-C instead of DSA-B.) Similarly, A_2 only evaluates: $x_1 = x_2$, $x_2 = x_3$, $x_2 = x_4$. A_3 evaluates: $x_1 = x_3$, $x_2 = x_3$, $x_3 = x_4$. A_4 evaluates $x_1 = x_4$, $x_2 = x_4$, $x_3 = x_4$.

Agent A_1 can only go to New York (NY), Los Angeles (LA), or Washington DC (W) i.e. $x_1 \in \{NY, LA, W\}$. Agent A_2 can only go to Dallas (D), Los Angeles (LA), Pittsburgh (P), or Washington DC (W) i.e. $x_2 \in \{D, LA, P, W\}$. Agent A_3 can only go to Washington DC (W) i.e. $x_3 \in \{W\}$. Agent A_4 can only go to Los Angeles (LA), Washington DC (W), and Dallas (D) i.e. $x_4 \in \{LA, W, D\}$.

In Figure 1 is given a successful trace of DSA-B for our problem. As it can be seen, 4 cycles with DSA-B and acceptable outcomes of the random generators used for randomizing the decisions lead to convergence.

Cycle	A_1(NY,LA,W)	A_2(D,LA,P,W)	A_3(W)	A_2(LA,W,D)
1	NY	D	W	LA
	$(\Delta = 1 > 0) => W, LA(p)NY(1-p)$	$(\Delta = 1 > 0) => W, LA(p)D(1-p)$	$(\Delta = 0) => W$	$(\Delta = 1 > 0) => W, D(p)LA(1-p)$
	NY	LA	W	D
2	NY	LA	W	D
	$(\Delta = 1 > 0) => W, LA(p)NY(1-p)$	$(\Delta = 1 > 0) => D, W(p)LA(1-p)$	$(\Delta = 0) => W$	$(\Delta = 1 > 0) => LA, W(p)D(1-p)$
	LA	LA	W	**LA**
3	LA	LA	W	LA
	$(\Delta < 0) => LA$	$(\Delta < 0) => LA$	$(\Delta = 0) => W$	$(\Delta < 0) => LA$
	LA	LA	W	LA

Figure 3: Less successful trace of the problem with DSA-$\overline{B}$. A local minima and deadlock is reached after 2 cycles with an unfortunate chance.

With this example, it happens that sometimes two distinct values offer simultaneously the best $\Delta > 0$. In this case we change the value with probability p and the next value is chosen with equal probability among those offering the improvement Δ. To distinguish this version of the DSA-B algorithm, we denote it by DSA-$\overline{B}$. Similarly one can get DSA-$\overline{A}$, DSA-$\overline{C}$, DSA-$\overline{D}$, DSA-$\overline{E}$.

But let us notice how fragile our success was. In Figure 3 is given a trace where the outcome of the used random generators in cycle 2 moves us into a bad minima. It is therefore clear that for many problems, DSA can converge to suboptimal minima with high probability.

Improvements to DSA

DSA1/DSA2 There are two obvious improvements to be brought to the DSA family. The first is to allow the agents to move up the hill, i.e. to increase the number of conflicts (even when an improvement exists) with an acceptable low probability. This version is called DSA1. The strategies for DSA1 are shown in Figure 4. They correspond to the mentioned strategies for DSA, but a small probability allows agents to get out of local minima. For example, in the trace of Figure 3 the agent A_4 has a chance ($p_2/2$) to change its proposal back to W and later the trace can develop as in Figure 1. Another alternative, DSA2, consists of allowing the agent to move towards more conflicts only when $\Delta \leq 0$ (see Figure 5).

Distributed Simulated Annealing (DSAN) The drawback of both DSA1 and DSA2 is that the probability p_2 has to be chosen very low to avoid that the system leaves too easily the global optima. But choosing it too low reduces the chance of passing any sufficiently high barrier towards the global/better optima. This is not a lost war, and a principled solution has been found much time ago (Kierkpatrick, Gelatt, & Vecchi 1983; Aarts & Korst 1989). The solution is to follow the example of physical annealing by simulating it with the distributed search. Note that Distributed Simulated Annealing (DSAN) is closer to the real phenomena than the description in (Kierkpatrick, Gelatt, & Vecchi 1983). All that has to be changed to DSA is to use a decreasing schedule of temperatures $T = \{t1, t2, ...\}$ in each agent and to move in cycle i to a randomly selected next value with the famous probability e^{Δ/t_i} when $\Delta < 0$, or with probability 1 otherwise. Here Δ is the improvement that would be brought by the new value (see Algorithms 3, 4).

In the described experiments we have choosen a schedule of temperatures given by $ti = const/i^k$, where we used $k = 2$ and $const = 1000$, 1000 being also the maximum number of iterations that we run. Following our experiments we conclude that a sequence of subsequent iterations with temperature zero should follow at the end. This schedule also starts with a too high temperature which leads to poor starts, therefore lower values for const should be recommended.

Note that extensive work exists in the field of Simulated Annealing. Still, the only non-centralized versions that we have found in literature so far were limited to parallelizations that let several processors pursue the same problem with different heuristics (initial values and formulations, random number generators, seeds, etc.) (Aarts & Korst 1989).

Extensions

An unlimited number of other alternatives is possible (e.g. using the strategies DSA1/DSA2 where p_2 decreases according to a schedule of temperatures) and their merit remains to be explored. We will call the techniques obtained this way: DSAN1, DSAN2

We have seen a large number of alternatives for improving DSA. All of them allow to get out of the minima that put DSA-B in trouble. Given its experimental success, we expect that Distributed Simulated Annealing (DSAN) and its versions will remain among the most important distributed techniques from the point of view of efficiency.

We also designed a hybrid DSAN-DSA that alternates between the two algorithms. According to exper-

Algorithm	$\Delta > 0$	C, $\Delta = 0$	no C, $\Delta = 0$	C, $\Delta < 0$
DSA-A1	gv with p_1/g, ov with p_2/o	ov with p_2/o	ov with p_2/o	ov with p_2/o
DSA-B1	gv with p_1/g, ov with p_2/o	gv with p_1/g, ov with p_2/o	ov with p_2/o	ov with p_2/o
DSA-C1	gv with p_1/g, ov with p_2/o	gv with p_1/g, ov with p_2/o	gv with p_1/g, ov with p_2/o	ov with p_2/o
DSA-D1	gv	gv with p_1/g, ov with p_2/o	ov with p_2/o	ov with p_2/o
DSA-E1	gv	gv with p_1/g, ov with p_2/o	gv with p_1/g, ov with p_2/o	ov with p_2/o

Figure 4: Alternatives of DSA1 algorithms. gv is any of the values resulting in the best Δ. ov is any of the other values than the current value or an gv. g is the number of gv values and o is the number of ov values. p_1 and p_2 are two probabilities.

Algorithm	$\Delta > 0$	C, $\Delta = 0$	no C, $\Delta = 0$	C, $\Delta < 0$
DSA-A2	gv with p_1/g	ov with p_2/o	ov with p_2/o	ov with p_2/o
DSA-B2	gv with p_1/g	gv with p_1/g, ov with p_2/o	ov with p_2/o	ov with p_2/o
DSA-C2	gv with p_1/g	gv with p_1/g, ov with p_2/o	gv with p_1/g, ov with p_2/o	ov with p_2/o

Figure 5: Alternatives of DSA2 algorithms. The notation is similar to the one for DSA1 strategies.

Algorithm 3: The algorithm performed by all agents for DSAN

```
procedure DSAN do
    Randomly choose a value;
    i←0;
    while (no termination condition is met) do
        i←i + 1;
        if (a new value is assigned) then
            send the new value to neighbors;
        end
        collect neighbors' new values, if any;
        select randomly a value and adopt it with
                probability e^(Δ/t_i) when Δ≤0,
                or with probability 1 otherwise;
    end
end do.
```

Algorithm 4: An asynchronous version of DSAN

```
procedure DSAN do
    Randomly choose a value;
    i←0;
    while (no termination condition is met) do
        i←i + 1;
        if (a new value is assigned) then
            send the new value to neighbors;
        end
        set alarm;
        at alarm collect neighbors' new values
             received since the last alarm, if any;
        select randomly a value and adopt it with
                probability e^(Δ/t_i) when Δ≤0,
                or with probability 1 otherwise;
    end
end do.
```

Algorithm 5: The hybrid DSAN-DSA algorithm.

```
procedure DSAN-DSA do
    choose K∈ [1..10], k ∈ [2..4], and MaxIterationsNr (approx. 1000);
    for (i=0;i<MaxIterationsNr;i++) do
        DSA-step();
        if (quality did not improve 1% in last MaxIterationsNr*0.1 steps) then
            store solution; break;
        end
    end
    for (;i<MaxIterationsNr*0.8;i++) do
        temperature←K*0.64/i^k;
        DSAN-step();
    end
    for (;i<MaxIterationsNr*0.9;i++) do
        temperature←0;
        DSAN-step();
    end
    for (;i<MaxIterationsNr*0.9;i++) do
        DSA-step();
    end
end do.
```

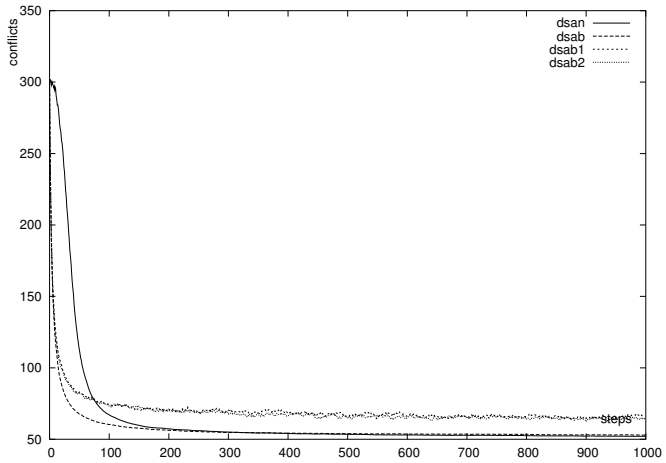

Figure 6: DSAN vs DSA-$\overline{B}$ vs DSA-B1 vs DSA-B2 averaged on 100 problem instances with tightness 20%.

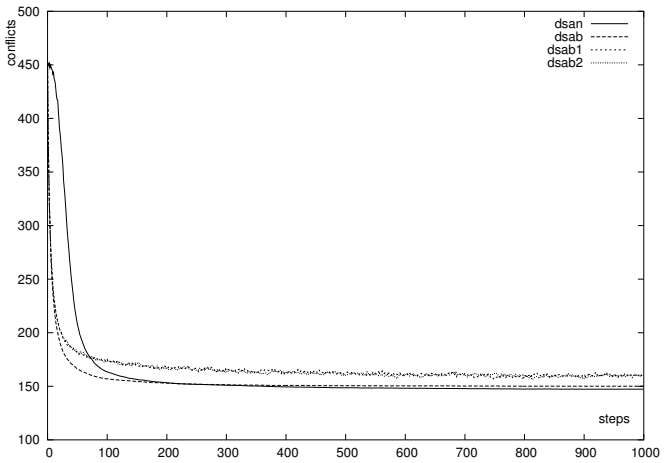

Figure 7: DSAN vs DSA-$\overline{B}$ vs DSA-B1 vs DSA-B2 averaged on 100 problem instances with tightness 30%.

iments, hybrids between DSAN and DSA, (e.g. DSAN-DSA, DSAN1, DSAN2) are expected to inherit all good properties of both DSAN and DSA. Running first DSA will offer its good 'anytime' value. When DSA's convergence is detected, the best solution can be stored and DSAN can be launched to lead to better solution quality. Whenever DSAN is stopped, DSA can be relaunched to lead to the neighbor local minima.

Experiments

A set of tests has been run where we are comparing DSAN, DSA1, DSA2, and DSA-B for different values of p, p_1, and p_2. For DSAN with $I = 1000$ iterations, the schedule of temperatures at iteration i was chosen as given by the function $t_i = \frac{I}{i^2}$.

We have generated randomly CSPs with 100 variables and 10 values per domain. The density arcs was choosen of 30% of the total number of possible arc. The tightness, probability of a tuple being feasible for an existing arc, was between 10% and 50%. We tested 100 problem instances randomly generated for each of the tightness values 10%, 20%, 30%, 40%, 50$ (500 randomly generated problem instances).

Each problem was solved with DSAN, DSA-$\overline{B}$, DSA-B1, DSA-B2 and MaxIterationsNr was always set to 1000. p and p_1 were set to 20$ while p_2 was set to .5%.

Figures 6-9 show that DSA-$\overline{B}$, DSA-$\overline{B}$1, DSA-$\overline{B}$2 prove a very good anytime behavior descending quickly in about 100 steps almost to their final solution. DSA-$\overline{B}$1 and DSA-$\overline{B}$2 have a very irregular curve due to their probability p_2 of jumping out of the best solution and consistently lie significantly higher than the curve of DSA-$\overline{B}$. DSA-$\overline{B}$ gets trapped into a local minima after 300-500 steps.

DSAN starts much worse than than DSA, mainly also due to its high initial temperature. It approached DSA only after 100 steps, this is when the temperature is

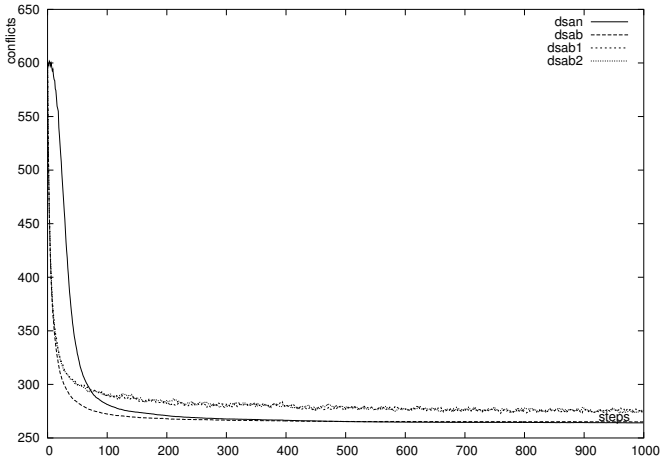

Figure 8: DSAN vs DSA-$\overline{B}$ vs DSA-B1 vs DSA-B2 averaged on 100 problem instances with tightness 40%.

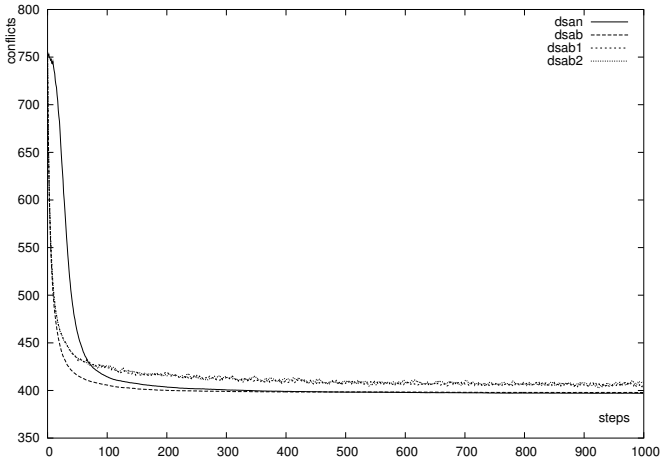

Figure 9: DSAN vs DSA-$\overline{B}$ vs DSA-B1 vs DSA-B2 averaged on 100 problem instances with tightness 50%.

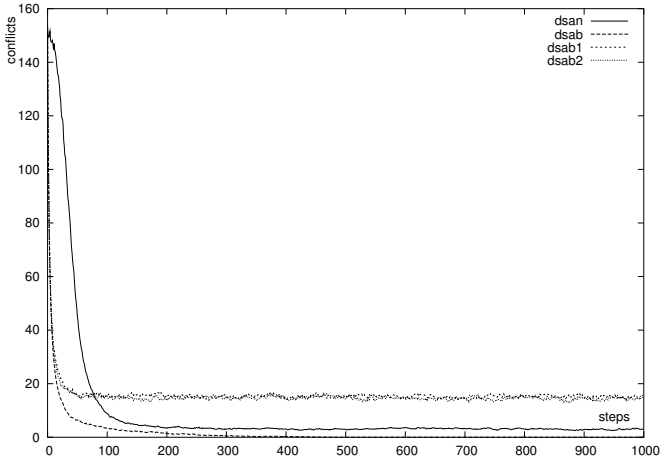

Figure 10: DSAN vs DSA-$\overline{B}$ vs DSA-B1 vs DSA-B2 averaged on 100 easy problem instances with tightness 10%.

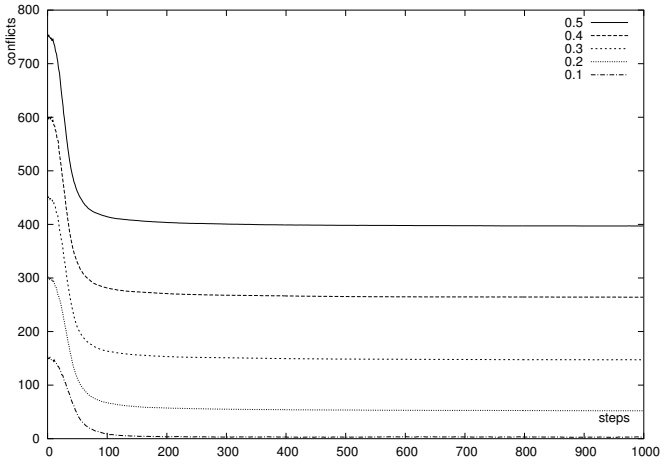

Figure 11: DSAN averaged on 100 problem instances for each tightness 10% to 50%.

close to 1. However soon after that, at about 200 steps, it reaches better solutions that DSA-$\overline{B}$. The improvement in the solution offered by DSAN vs. the one of DSA-$\overline{B}$ is visible in hard regions of phase transition between easy and hard problems, which in our case is at tightness of 20-30. There the improvement is of about 2% less conflicts with DSAN than with DSA-$\overline{B}$.

In simulator, DSA-$\overline{B}$, DSA-$\overline{B}$1, and DSA-$\overline{B}$2 are all about an order of magnitude slower in time than DSAN (10 being the domain size), meaning that in simulator DSAN solved problems much quicker than DSA-$\overline{B}$. This measure has no influence on distributed runs where the agents are remote and the cost of messages is high compared to local constraint checks.

Further tests

By the time of the presentation of this article we plan to also describe the behavior or DSAN-DSA, DSAN1, DSAN2, as well as DSAN for temperature schedules $1/i^2$.

Conclusions

Distributed Stochastic Search Algorithm (DSA) is an important new family of algorithms that has been quickly imposed due to its simplicity and efficiency. The algorithm is uniform, i.e. all the agents perform identical functions and there is no need of names to break ties (Fabiunke 1999; Fitzpatrick & Meertens 2001; Zhang, Wang, & Wittenburg 2002). We discovered that even the best among the versions of DSA has a high risk to get caught in local minima. Moreover, none of the existing DSA variants have any means or chance to get out of such minima. We find that versions offering chances to get out of such minima are possible and we propose a few such variants. The centralized techniques already classified such techniques as more promising than the strict descent. For example, Simulated Annealing (Kierkpatrick, Gelatt, & Vecchi 1983) is so important that it became a field by itself. Distributed Simulated Annealing may have the same destiny.

References

Aarts, E., and Korst, J. 1989. *Simulated Annealing and Bolzmann Machines*. John Wiley & Sons.

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

Fabiunke, M. 1999. Parallel distributed constraint satisfaction. In *PDPTA*, 1585–1591.

Fitzpatrick, S., and Meertens, L. 2001. An experimental assessment of stochastic, anytime, descentralized, soft colourer for sparse graphs. In *Symp. on Stohastic Algorithms: Foundations and Applications*, 49–64.

Kierkpatrick, S.; Gelatt, C.; and Vecchi, M. 1983. Optimization by simulated annealing. *Science* 220:671–680.

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

Tel, G. 1999. *Multiagent Systems, A Modern Approach to Distributed AI*. MIT Press. chapter Distributed Control Algorithms for AI, 539–580.

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

Zhang, W., and Wittenburg, L. 2002. Distributed breakout revisited. In *Proc. of AAAI'2002*.

Zhang, W.; Wang, G.; and Wittenburg, L. 2002. Distributed stochastic search for constraint satisfaction and optimization: Parallelism, phase transitions and performance. In *PAS*.