Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

Distributed Breakout: Beyond Satisfaction

Steven Okamoto and **Roie Zivan** and **Aviv Nahon**
Industrial Engineering and Management Department
Ben-Gurion University of the Negev
Beer-Sheva, Israel
{okamotos,zivanr,nahona}@bgu.ac.il

Abstract

The Distributed Breakout Algorithm (DBA) is a local search algorithm that was originally designed to solve DisCSPs and DisMaxCSPs. Extending it to general-valued DCOPs requires three design choices: manner of modifying base costs (multiplicative weights or additive penalties); definition of constraint violation (non-zero cost, non-minimum cost, and maximum cost); and scope of modifying cost tables during breakout (entry, row, column, or table).

We propose Generalized DBA (GDBA) to span the 24 combinations in the three dimensions. In our theoretical analysis we prove that some variants of GDBA are equivalent for certain problems, and prove that other variants may find suboptimal solutions even on tree topologies where DBA is complete. Our extensive empirical evaluation on various benchmarks shows that in practice, GDBA is capable of finding solutions of equal or significantly lower cost than alternative heuristic approaches (including DSA).

1 Introduction

Distributed constraint satisfaction and optimization problems (DisCSPs and DCOPs) are common frameworks for representing multiagent coordination problems that are distributed by nature, such as target tracking in sensor networks [Zhang *et al.*, 2005] and scheduling meetings in offices [Maheswaran *et al.*, 2004b]. Because they are NP-hard, considerable research effort has been devoted to developing incomplete algorithms for finding good solutions quickly [Yokoo and Hirayama, 1996; Maheswaran *et al.*, 2004a; Zhang *et al.*, 2005; Basharu *et al.*, 2005; Farinelli *et al.*, 2008; Smith and Mailler, 2010]. Local search algorithms, e.g., DSA [Zhang *et al.*, 2005] and MGM [Maheswaran *et al.*, 2004a], are simple incomplete algorithms with a naturally distributed structure. Although they commonly offer no (or little) quality guarantees, they were empirically found to produce high quality solutions [Yokoo and Hirayama, 1996; Maheswaran *et al.*, 2004a; Zhang *et al.*, 2005; Zivan *et al.*, 2014]

The Distributed Breakout Algorithm (DBA) is one such local search algorithm, originally proposed to solve DisCSPs [Yokoo and Hirayama, 1996]. Agents in DBA associate weights with constraints and employ greedy local search to reduce the weighted sum of violated constraints. Agents try to escape local minima via a *breakout* mechanism that increases these weights.

DCOPs generalize DisCSPs by replacing hard constraints with cost functions that factor a global objective function. Initial attempts to adapt DBA to DCOPs focused on minimizing the number of constraint violations in over-constrained DisCSPs (also known as DisMaxCSPs) [Hirayama and Yokoo, 1997; Wittenburg and Zhang, 2003]. DBA was subsequently used with more general-valued DCOPs (i.e., problems in which a violated constraint incurs a numeric cost) [Zhang *et al.*, 2005], but the details of this adaptation were omitted. This is unfortunate, because many details of DBA become unclear with general-valued cost functions. In DisCSP an assignment can either violate or satisfy a constraint. What does it mean for a general-valued DCOP constraint to be violated, when each combination of assignments incurs some cost? In DisCSP the weight of a constraint is increased by one. For a DCOP should this be interpreted as an increment to the constraint cost (e.g., increasing a base cost of 3 to 4, then to 5, etc.) or as an increment to a multiplicative factor of the constraint cost (e.g., increasing a base cost of 3 to 6, then to 9, etc.)? In DisCSP a constraint is a single joint assignment to a set of variables, and the scope of the weight increase is exactly that single joint assignment. In DCOP, constraints are arbitrary functions from joint assignments to costs, so what set of joint assignments should be affected by an increase?

In this paper we make three major contributions. First, we formalize the three design choices required to adapt DBA to general-valued DCOPs: the *manner* of computing effective costs based on true costs and modifiers (i.e., the "weights" of the original DBA), the *definition* of constraint violation, and the *scope* of changes to the modifiers during breakouts. For each choice we consider alternatives that are consistent with the original DBA: three definitions, two manners, and four scopes, resulting in a space of 24 variants parameterized along three dimensions. We propose the Generalized Distributed Breakout Algorithm (GDBA) to span this space.

Second, we establish theoretical properties of GDBA. We establish equivalences between some variants on important classes of problems. We also prove limitations of GDBA on general-valued DCOPs. Although DBA is complete and sound on graph coloring DisCSPs with tree topologies, we prove that DCOPs defined on the same topologies may not be solved optimally by some variants of GDBA.

Our third contribution is an extensive empirical analysis of GDBA on three classes of DCOPs: random graphs with unstructured cost functions, weighted graph coloring, and meeting scheduling. Our results show that GDBA finds equal or better solutions than state-of-the-art competitors, in contrast to previous reported results [Zhang *et al.*, 2005].

2 Distributed Satisfaction and Optimization

DisCSPs and DCOPs have *agents* $\mathcal{A} = \{A_1, \ldots, A_m\}$; *variables* $\mathcal{X} = \{X_1, \ldots, X_n\}$ where each variable uniquely belongs to (is held by) one of the agents; finite *domains* $\mathcal{D} = \{D_1, \ldots, D_m\}$ where D_i specifies the values that X_i can take; and *constraints* describing relationships between variables. We make the standard assumption that each agent controls exactly one variable ($n = m$), and hence we use the terms "agent" and "variable" interchangeably. We further adopt a common assumption that all constraints are binary, involving only two agents. The undirected *constraint graph* has vertices $\mathcal{X}$ and an edge (X_i, X_j) if and only if X_i and X_j are constrained. Two variables are *neighbors* if they are adjacent in the constraint graph, and $\mathcal{N}_i$ denotes the set of neighbors of X_i. Each agent has exclusive control over the value assignment of its variable, and knowledge only of its neighbors and the constraints it has with its neighbors.

The set of constraints in a DisCSP is $\mathcal{C}$. Each constraint $C_l \in \mathcal{C}$ between X_{l1} and X_{l2} is a pair of values $C_l \in D_{l1} \times D_{l2}$ representing a *no-good*, i.e., a disallowed combination of values for X_{l1} and X_{l2}. If an assignment takes the values disallowed by a constraint, it is said to *violate* that constraint. The agents' goal is to choose a full assignment $\mathbf{x} = (x_1, \ldots, x_n) \in D_1 \times \cdots \times D_n$ such that no constraint is violated. DisMaxCSP generalizes DisCSP to minimize the number of violated constraints.

The set of constraints in a DCOP is $\mathcal{F}$. Unlike the no-good representation of DisCSP constraints, each DCOP constraint $F_{ij} \in \mathcal{F}$ is a function $F_{ij} : D_i \times D_j \to \mathbb{Z}_{\geq 0}$ mapping pairs of values of the constrained variables to costs. Constraints are symmetric so F_{ij} and F_{ji} are alternate names for the same constraint. The agents' objective is to choose a full assignment $\mathbf{x} = (x_1, \ldots, x_n) \in D_1 \times \cdots \times D_n$ minimizing the total cost $F(\mathbf{x}) = \sum_{F_{ij} \in \mathcal{F}} F_{ij}(x_i, x_j)$.

We refer to the DCOP binary constraints as two-dimensional *tables* with rows and columns indexed by values in the domains of the two constrained variables. We adopt the convention that when referring to a table F_{ij} from the perspective of a specific agent i, the values in D_i always index the rows. Thus, the tables from the perspectives of i and j for a constraint $F_{ij} \in \mathcal{F}$ are transposes of each other.

We assume that the tables are not uniform (otherwise they can simply be ignored by the agents).

3 Distributed Breakout Algorithm (DBA)

DBA is a synchronous local search algorithm designed for DisCSPs. Agents in DBA collectively maintain a full assignment $\mathbf{x}$ with each agent keeping a current value for its variable. Each agent also maintains a *weight*, initially 1, for each constraint it is involved in. The agents greedily update $\mathbf{x}$ in every iteration to minimize the sum of weights of violated constraints. Each agent calculates the maximal improvement in the sum of weights that it can achieve by unilaterally changing its assignment, then coordinates with its neighbors to ensure that only the agent with the greatest improvement (subject to deterministic tie-breaking) in its neighborhood is allowed to change its value.

To escape local minima, DBA modifies its objective function by increasing some of the weights. Detecting true local minima requires the full assignment $\mathbf{x}$, which is not generally known to a single agent, so agents instead detect a weaker condition, quasi-local minima, that can be computed using only information about their neighbors.

Definition 1 A_i is in a *quasi-local minimum (QLM)* if neither it nor any of its neighbors can unilaterally change to an assignment resulting in lower effective cost.

When an agent detects that it is in a QLM, it performs a *breakout* by increasing the weights of all violated constraints, eventually causing it to choose a new value. Neighboring agents will not necessarily be in QLMs at the same time, and hence one neighbor may break out while the other does not. This may lead to agents having different weights for the same constraint. (A later revision to DBA disallowed this [Hirayama and Yokoo, 2005], but we follow the original description [Yokoo and Hirayama, 1996].)

Algorithm 1: GDBA

Initialize cost modifiers to 0
Choose random value $x_i \in D_i$
Send x_i to all neighbors
while *termination condition not met* **do**
    Receive x_j from all neighbors
    $x'_i \leftarrow \arg\min_{d \in D_i \setminus \{x_i\}} \sum_{j \in \mathcal{N}_i} \text{EFFCOST}(d, j, x_j)$
    $\Delta_i \leftarrow \sum_{j \in \mathcal{N}_i} \text{EFFCOST}(x'_i, j, x_j) - \text{EFFCOST}(x_i, j, x_j)$
    Send Δ_i to all neighbors
    Receive Δ_j from all neighbors
    if $\Delta_i > 0$ **then**
        if Δ_i *is best improvement* **then**
            $v_i \leftarrow x'_i$
    else if *no neighbor can improve* **then**
        foreach $j \in \mathcal{N}_i$ **do**
            if $\text{ISVIOLATED}(x_i, j, x_j)$ **then**
                $\text{INCREASEMOD}(x_i, j, x_j)$
    Send x_i to all neighbors
Assign best x_i to X_i

Figure 1: Generalized DBA executed by A_i.

4 Generalized DBA (GDBA)

DBA was originally designed for solving DisCSPs but the general structure of the algorithm is compatible with DCOPs. In fact, agents in DBA implicitly try to solve a DisMaxCSP, searching for an assignment with cost 0. GDBA (pseudocode

Name	Description
M	Multiplicative manner.
A	Additive manner.
NZ	Non-zero constraint violation.
NM	Non-minimum constraint violation.
MX	Maximum constraint violation.
E	Entry scope.
C	Column scope.
R	Row scope.
T	Table scope.

Table 1: Summary of the design alternatives for GDBA.

in Figure 1) extends DBA to DCOPs with general constraint costs, where the original description may be interpreted in multiple ways, all of which are equivalent on DisMaxCSPs.

Each agent A_i uses a modifier function $M_{ij} : D_i \times D_j \to \mathbb{Z}_{\geq 0}$ for each constraint F_{ij}, analogous to the weights in DBA. As with weights, these modifiers are increased during breakout. The modifiers combine with the *base costs* of the DCOP constraints to yield local *effective cost functions* (EffCost), which generalize the sum of weights in DBA. They are initialized to 0 in line 1 and increased by IncreaseMod. A synchronous step is defined by every agent receiving messages from all of its neighbors. Each iteration requires one step (lines 5 – 8) to calculate the best local improvement Δ_i based on the values received from neighbors, and another step (lines 9 – 17) to decide if A_i can change its assignment. If A_i detects a QLM, it performs the breakout process described in lines 14 – 16.

Detecting an optimal solution in a DCOP is generally intractable (unless P=NP) and so the search continues even if the optimal solution has been encountered. We assume an upper limit on the number of iterations as the termination condition (line 4), as it is simple and predictable. GDBA can also use a method like the Anytime Local Search framework [Zivan *et al.*, 2014] to cache the best solution found during the search and assign it upon termination (line 18). Without such a method, the solution reported is the assignment held by the agents following the last iteration of the algorithm.

GDBA has the same time and space complexity of DBA. Each agent A_i requires $O(|\mathcal{N}_i| \cdot |D_i|)$ time in each step and $O(|\mathcal{N}_i| \cdot |D_i| \cdot \max_{j \in \mathcal{N}_i} |D_j|)$ space total. (Note that all agents execute each step in parallel.)

To complete the design of GDBA we next define the three subroutines EffCost, IsViolated, and IncreaseMod that implement the design alternatives summarized in Table 1.

Manner of cost increase defines how base costs and modifiers are combined to yield effective costs and is implemented in EffCost. We consider two manners. The first, *M*, uses modifiers as multiplicative factors to the base costs (line 20). This amplifies inherent differences in the base constraints, allowing for faster modification of the cost landscape while preserving some of the underlying problem structure. The second, *A*, uses modifiers as additive penalties to the base costs (line 21), allowing for finer-grained but slower modification of the cost landscape, and a risk of erasing inherent cost differences, resulting in an ill-fitting objective function.

Function EffCost(x_i, j, x_j)

19 **switch** *manner* **do**
20   **case** *M:* **do return** $F_{ij}(x_i, x_j) \cdot [M_{ij}(x_i, x_j) + 1]$
21   **case** *A:* **do return** $F_{ij}(x_i, x_j) + M_{ij}(x_i, x_j)$

Constraint violation is defined in IsViolated. We consider three possible definitions for constraint violations based on the base costs of the constraints. The use of base costs rather than effective costs is consistent with DBA, which only uses the presence, not the weight, of a no-good for determining constraint violation. It also helps to keep the cost landscape closer to the true objective, because only "bad" (i.e., violated) constraints are modified in breakouts. The three definitions are progressively more restrictive in defining violation, leading to less exploration but greater chance of convergence.

Function IsViolated(x_i, j, x_j)

22 $c \leftarrow$ EffCost(x_i, j, x_j)
23 **switch** *violation definition* **do**
24   **case** *NZ:* **do return** $c > 0$
25   **case** *NM:* **do return** $c > \min_{d_1 \in D_i, d_2 \in D_j} F_{ij}(d_1, d_2)$
26   **case** *MX:* **do return** $c = \max_{d_1 \in D_i, d_2 \in D_j} F_{ij}(d_1, d_2)$

A constraint is *NZ*-violated if it has a *non-zero* cost in an assignment (line 24). Some constraints may not have 0 cost for any assignment and hence will always be *NZ*-violated; although this will prevent convergence, it allows for more exploration that may result in better solutions. A constraint is *NM*-violated if its cost under the current assignment is *non-minimum* over all joint assignments (line 25), this is less explorative than *NZ* but more likely to converge because for every constraint there is always a joint assignment that does not *NZ*-violate it. Finally, a constraint is *MX*-violated if it takes the *maximum* cost over all possible joint assignments (line 26). Breakouts will only occur when an agent is trapped by its neighbors in its worst assignment; for other cases, GDBA with *MX* is not explorative and relies solely on greedy hill climbing for improvement. For every constraint there is always a joint assignment that does not *MX*-violate it.

Scope of cost increase specifies the elements in M that are incremented when the breakout is performed for a violated constraint. We consider four scopes named for the dimension in the table M that is altered and implemented in IncreaseMod. In DBA, the weight of a single no-good is incremented. The analogue in GDBA is to increment the single entry corresponding violated no-good; we call this the *E* scope. This is a very fine-grained modification that results in slower evolution of the cost landscape but more accurately captures the local minima, because an agent must actually be trapped in a QLM to modify the cost of a joint assignment. The *C* scope affects the costs in a column: all values in the domain of X_i given the neighbor's current value. With *M*

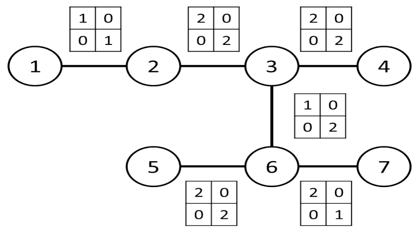


Figure 2: Counter-example to completeness of GDBA with maximum constraint violations.

manner this scales the relative importance of violated constraints for the agent's current value; as the agent is trapped in its current value, this eventually allows it to escape. (We prove in Proposition 3 that it is not very interesting for *A*.) The *R* scope affects the costs in a row: all entries for the current value of X_i for all value assignments of the neighbor. This is similar to a value penalty in DisPeL [Basharu *et al.*, 2005], trying to move the agent away from a value because it previously led to a QLM, even if the current context is different. Finally, the *T* scope affects the cost in the whole table: all entries in the modifier for a violated constraint are incremented, effectively making the constraint more important to A_i relative to its other constraints. *T* modifies the cost landscape the fastest and most drastically of all scopes.

Function INCREASEMOD(x_i, j, x_j)

```
switch scope do
  case E: do Increment C_ij(x_i, x_j)
  case C: do for d ∈ D_i do Increment C_ij(d, x_j)
  case R: do  for d ∈ D_j do Increment C_ij(x_i, d)
  case T: do
    for d_1 ∈ D_i, d_2 ∈ D_j do Increment C_ij(d_1, d_2)
```

5 Theoretical Results

We identify specific variants of GDBA with triples (*manner*, *definition*, *scope*) specifying the three dimensions. Because there are two manners, three violation definitions, and four scopes, there are a total of 24 GDBA variants. However, in this section we show that some of these may be equivalent for some classes of problems, i.e., in all iterations they compute the same effective costs and hence send the same messages and select the same assignments. When a dimension is restricted to a subset of the possible implementations, we represent it as a set in the triple notation. For example, (*M*, {*NZ*, *NM*}, *E*) is DBA with *M* manner, *E* scope, and either *NZ* or *NM* constraint violation. If a dimension may take any value, we represent it with "*". For example, (*, *NZ*, *E*) means DBA with *NZ* constraint violation, *E* scope, and any manner.

5.1 Equivalences

We start with DCOPs with binary-valued constraints:

Proposition 1 *If for all $F_{ij} \in \mathcal{F}$ the set $\{F_{ij}(d_1, d_2) \mid d_1 \in D_i, d_2 \in D_j\} = \{0, 1\}$, then (*, *, E) are equivalent.*

Proof: The equivalence of the three violation definitions follows directly from the assumption that all constraints have minimum cost of 0 and maximum cost of 1.

Equivalence of manners is shown by induction on the number of iterations. We show that $M_{ij}(d_1, d_2)$ is the same under both *M* and *A* for all M_{ij} and $d_1 \in D_i, d_2 \in D_j$ and that the effective costs are the same. In the first iteration all M_{ij} are initialized as zero and so the effective cost is F_{ij} for both manners and the base case is proven.

Assume next that the M_{ij} and effective costs are the same for both manners on iteration t. Thus the same value will be chosen for time $t+1$ and the same constraints will be broken out of at time t if A_i is in a QLM. Hence on iteration $t+1$, $M_{ij}(x_i, x_j)$ will be the same for both manners. There are now two cases. If $F_{ij}(x_i, x_j) = 1$ then the effective cost is $M_{ij}(x_i, x_j) + 1$ under both manners. If $F_{ij}(x_i, x_j) = 0$ then $M_{ij}(x_i, x_j) = 0$ because the scope is *E* and $F_{ij}(x_i, x_j)$ is not a violation. Thus under both manners the effective cost is 0 and equivalence is proven. $\square$

Corollary 1 *DBA is (*, *, E) on DisCSPs and DisMaxCSPs where every constraint can be satisfied and violated.*

Proposition 2 *With M manner and a fixed violation definition $V \in \{NZ, NM, MX\}$, the variants (M, V, $\{E, C, R\}$) are equivalent on graph coloring problems.*

Proof: In graph coloring, base costs $F_{ij}(x_i, x_j) > 0$ if and only if $x_i = x_j$, and so because of the *M* manner, the effective cost will also be non-zero if and only if $x_i = x_j$. This means that modifiers are irrelevant except for $M_{ij}(d_i, d_i)$ for $d_i \in D_i$. Constraint F_{ij} can only be violated (under any definition) when $x_i = x_j$, and for the scopes *E*, *C*, and *R*, $M_{ij}(d_i, d_i)$ is only incremented when $x_i = x_j = d_i$. Thus they are all equivalent.

Note that (M, V, T) is *not* equivalent to these, since for $d_i \in D_i$, the modifier $M_{ij}(d_i, d_i)$ can also be incremented when $x_1 = x_2 = d_2 \neq d_i$. $\square$

The next proposition shows that breakouts do not change the behavior of agents in some variants of GDBA.

Proposition 3 *(A,*,$\{C, T\}$) is equivalent to MGM, i.e., as if no breakouts are performed at all.*

Proof: We consider a variable X_i that is in a QLM and show that it does not change the choice of assignment after breakout. Let J be the set of j such that F_{ij} are currently violated, so that the breakout increments $M_{ij}(x_i, x_j)$ for all $j \in J$. Denote the effective cost before and after breakout as EFFCOST_t and EFFCOST_{t+1}, respectively. Then note that because we are in *C* or *T* scope,

$$\text{EFFCOST}_{t+1}(d, j, x_j) = \begin{cases} \text{EFFCOST}_t(d, j, x_j) + 1 & j \in J \\ \text{EFFCOST}_t(d, j, x_j) & \text{otherwise.} \end{cases}$$

Line 6 of Algorithm 1 computes the best alternative after breakout:

$$\begin{aligned}
&\underset{d\in D_i\setminus\{x_i\}}{\arg\min} \sum_{j\in\mathcal{N}_i} \text{EFFCOST}_{t+1}(d,j,x_j)\\
&= \underset{d\in D_i\setminus\{x_i\}}{\arg\min} \left(|\, J\, | + \sum_{j\in\mathcal{N}_i} \text{EFFCOST}_{t}(d,j,x_j) \right)\\
&= \underset{d\in D_i\setminus\{x_i\}}{\arg\min} \sum_{j\in\mathcal{N}_i} \text{EFFCOST}_{t}(d,j,x_j).
\end{aligned}$$

where the $|J|$ term is dropped because it does not affect the minimization. This is the same best alternative considered before the breakout, and hence the breakout does not change the agent's behavior. □

5.2 Limitations

We next show that GDBA may not optimally solve problems that can be solved by DBA when formulated as DisCSPs.

Proposition 4 *(*, MX, *) is not guaranteed to find optimal solutions, even when the constraint graph has a tree topology.*

Proof: Consider the DCOP in Figure 2 where every variable has domain $\{1, 2\}$. Note that there are two optimal solutions of cost 0 in which adjacent variables take alternating values.

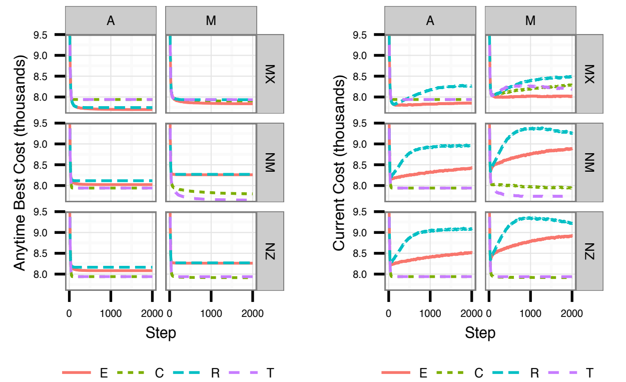
Let the starting assignment be $\mathbf{x} = (1, 2, 1, 2, 2, 1, 2)$ with cost 1, and consider Δ_i for each variable $X_i \in \mathcal{X}$. For X_1, changing to $x'_1 = 2$ incurs an increase in cost of 1 (i.e., $\Delta_1 = -1$), so X_1 cannot improve. For the other agents we get $\Delta_2 = \Delta_3 = -3$ and $\Delta_4 = \Delta_5 = \Delta_6 = \Delta_7 = -2$, so no agent can improve and every agent is in a QLM.

Every agent will try to breakout by increasing the cost of violated constraints. However, no constraint is MX-violated: F_{36} has maximum value of 2 when $v_3 = v_6 = 2$, so the current assignment of $v_3 = v_6 = 1$ does not MX-violate the constraint, and all other constraints have 0 cost. Thus no modifiers are increased and all subsequent iterations of the algorithm will maintain the suboptimal initial assignment. Because no effective cost is ever increased, the result holds for all manners and scopes of cost increase. □

Proposition 4 highlights the importance of problem formulation: the counter-example occurs precisely because GDBA tries to solve a *weighted* tree coloring problem. DBA optimally solves unweighted tree coloring [Zhang *et al.*, 2005], and by Corollary 1, so does (*, *MX*, *). Since trees can always be 2-colored, the optimal solution has 0 cost, and thus the weights could have been ignored.

6 Experimental Results

We tested GDBA on both abstract and realistic benchmark problems. Unless otherwise noted, all results reported are averages over 200 independent, randomly-generated instances with $n = 200$. We compared GDBA to six DCOP algorithms: DSA (type C, $p = 0.4$ and $p = 0.8$; settings of p from 0.1 to 0.9 were used and these performed best, each on different types of problems) [Zhang *et al.*, 2005], MGM and MGM2 [Maheswaran *et al.*, 2004a], Max-Sum with tie-breaking preferences [Farinelli *et al.*, 2008] and damping factor 0.5 [Tarlow *et al.*, 2010], DSAN [Arshad and Silaghi, 2004], and ADPOP ($maxDims = 2$) [Petcu and Faltings, 2005]. Descriptions of these algorithms are omitted for space.



(a) (b)

Figure 3: Anytime cost (a) and current cost (b) of GDBA on unstructured problems.

All algorithms were run for 2000 steps. For each algorithm we present the average anytime result following each step. Hence, non-monotonic algorithms (e.g., all GDBA versions and DSA) were implemented within the anytime framework proposed by [Zivan *et al.*, 2014]. In order to discuss the different levels of exploration caused by breakouts, we also present the cost of the current assignment following each step of GDBA.

We first considered entirely random constraint topologies: each pair of agents was constrained independently with probability $p_1 = 0.1$ (experiments on denser problems yielded generally similar results). Costs were random, with each entry in each table independently sampled from the discrete uniform distribution on $[1..10]$; zero was excluded to demonstrate GDBA's applicability to general-valued DCOPs. All domains had 10 values.

Figure 3(a) presents the anytime costs of the best solutions found by all GDBA variants. (*M*, *NM*, *T*) achieved the lowest cost overall, outperforming all other variants within 500 steps and continuing to slowly improve thereafter. With *MX* definition, *E* scope was best, while for the other definitions *C* and *T* scopes were best. *R* scope generally found poor solutions, except for (*A*, *MX*, *R*), which quickly found good solutions but did not continue to improve.

The results presented in Figure 3(b) demonstrate that breakouts do not always lead to better solutions. In particular, the poor performance of *R* scope is explained by breakouts causing inferior solutions to be explored, which also occurred with *E* scope with *NM* or *NZ* definitions.

E scope was most effective when paired with *MX* definition, particularly in the (*A*, *MX*, *E*) configuration. Although this led to worse solutions being explored on average (as seen by current cost increasing with step), it also allowed GDBA to continue to occasionally find better solutions (as seen by anytime costs continuing to slowly decrease). This used a relatively slow rate of exploration, since *MX* definition is the most restrictive violation definition (and hence the fewest assignments lead to breakouts) and *E* is the most restrictive scope

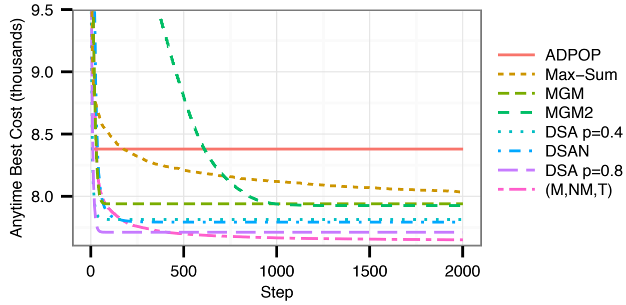


Figure 4: Anytime costs for each step of (M, NM, T) and competing heuristics on unstructured problems.

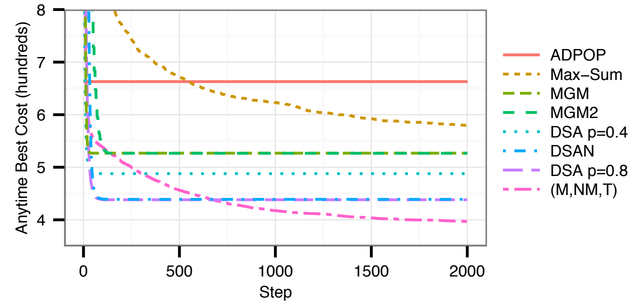


Figure 5: Anytime costs for each step of (M, NM, T) and competing heuristics on weighted graph coloring problems.

(and so each breakout results in the smallest modification to the cost landscape). For T scope, exploration through breakouts was really only useful in the (M, NM, T) configuration, as can be seen by the lower anytime costs vs. the current costs, but this balance between exploration and exploitation was the most successful overall.

Similar trends were observed in the graphs presenting the results on weighted graph coloring and meeting-scheduling. The figures were omitted for lack of space. The (M, NM, T) configuration dominated over all benchmarks.

Figure 4 plots the average anytime costs of (M, NM, T) and the competing algorithms at each step of execution on random unstructured problems. All differences at step 2000 are statistically significant for p-value < 0.01. DSA $p = 0.8$ performs the best out of all competing algorithms, quickly finding solutions of better quality on average than those found by the others; with $p = 0.4$ it finds considerably worse solutions. (M, NM, T) initially improves more slowly than DSA, which is not surprising considering that agents are only allowed to change values on every other step, while agents in DSA may change their values on every step if doing so leads to a local improvement. Despite this initial relative slowness, within 250 steps (M, NM, T) finds better solutions than all competing algorithms other than DSA, and within 500 steps it surpasses even DSA and continues to improve slowly (on average) when given more time.

We next considered coloring random graph topologies with $p_1 = 0.05$, $d = 3$, and random costs on [1..10]. Figure 5 presents the average anytime costs. Both DSA with $p = 0.8$ and DSAN very quickly find very good solutions (whose costs are not significantly different), although DSA finds its solutions slightly faster. DSA with $p = 0.4$ again finds much worse solutions. (M, NM, T) again takes longer to improve its solutions, but within 750 steps it finds solutions that are statistically better than all the competitors, including DSA. With more time, it continues to improve.

Our final set of experiments were on meeting scheduling problems using EAV representation [Maheswaran *et al.*, 2004b] with 200 people and 50 meetings to be scheduled in a 10-hour workday from 8:00 AM to 6:00 PM discretized into 15-minute time slots. Each meeting has a duration uniformly distributed on [1..4] time slots, and a desired attendance uniformly distributed on [2..6] people; the specific individuals are chosen uniformly at random. The required traveling time

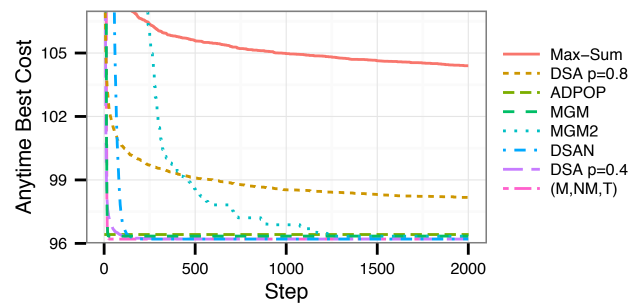


Figure 6: Anytime costs for each step of (M, NM, T) and competing heuristics on meeting scheduling problems.

between the locations of each pair of meetings is uniformly distributed on [1..2]. The preferences of the participants impose a cost that is uniformly distributed on [1..3] for each meeting and time. Two meetings scheduled so that common participants do not have enough time to attend both impose a conflict cost equal to the sum of the time slots missed by those participants in both meetings.

Figure 6 average anytime costs. Fast convergence suggests that breakouts are not effective in this setting, and that the bulk of (M, NM, T)'s improvement is due to greedy local search. Thus, it is not surprising that it behaves very similarly to MGM, which is just the local search component of DBA without a breakout mechanism. However, breakouts do confer a very small advantage to (M, NM, T), which finds better solutions than all other algorithms except DSAN and DSA (with $p = 0.4$), at a statistically significant level of p-value < 0.01; DSAN and DSA find solutions of exactly equal quality, albeit more slowly. DSA with the higher value of $p = 0.8$ did poorly on these problems, demonstrating that good DSA performance relies heavily on problem-specific parameter tuning; in contrast, (M, NM, T) performs well on different problems without needing such tuning.

7 Conclusion and Future Work

DBA is a local search algorithm designed to solve DisCSPs and DisMaxCSPs. Following early work comparing DSA and DBA, the common assumption of researchers was that DSA is superior, and therefore DSA was used for comparison in recent studies of incomplete DCOP algorithms.

We demonstrate in this paper that the performance of DBA when solving general-valued DCOPs is dependent on the design choices made. GDBA, generalizes DBA by allowing 24 combinations of 3 design choices. Our results demonstrate that GDBA, specifically the (*M*, *NM*, *T*) variant, is an effective algorithm for a wide range of general-valued DCOPs. This contrasts with DBA's well-known limitations on DisCSPs and DisMaxCSPs. Many other DCOP algorithms that originated as DisCSP algorithms may benefit from a similar rigorous study. For example, DSA-A and DSA-B differ in their behavior only when there are violated constraints; this invites investigation of violation definitions for DSA. (We used DSA-C, which does not require such definition.)

DBA, as a distributed implementation of a centralized method [Morris, 1993], is a bridge between DisCSPs and centralized CSPs. Future work should explore what other centralized approaches may be adapted to the distinct but related DisCSP and DCOP fields.

References

[Arshad and Silaghi, 2004] M. Arshad and M. C. Silaghi. Distributed simulated annealing. *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems, Frontiers in Artificial Intelligence and Applications series*, 112, November 2004.

[Basharu *et al.*, 2005] Muhammed Basharu, Ines Arana, and Hatem Ahriz. Solving DisCSPs with penalty driven search. In *Proceedings of the 20th National Conference on Artificial intelligence - Volume 1*, AAAI'05, pages 47–52. AAAI Press, 2005.

[Farinelli *et al.*, 2008] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *In: 7 th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08*, pages 639–646, 2008.

[Hirayama and Yokoo, 1997] Katsutoshi Hirayama and Makoto Yokoo. Distributed partial constraint satisfaction problem. In Gert Smolka, editor, *Principles and Practice of Constraint Programming-CP97*, volume 1330 of *Lecture Notes in Computer Science*, pages 222–236. Springer Berlin Heidelberg, 1997.

[Hirayama and Yokoo, 2005] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161:1-2:89–116, January 2005.

[Maheswaran *et al.*, 2004a] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *Proceedings of Parallel and Distributed Computing Systems*, pages 432–439, September 2004.

[Maheswaran *et al.*, 2004b] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling. In *AAMAS'04*, 2004.

[Morris, 1993] P. Morris. The breakout method for escaping from local minima. In *Proc. AAAI-93*, pages 40–45, 1993.

[Petcu and Faltings, 2005] Adrian Petcu and Boi Faltings. Approximations in distributed optimization. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP*, pages 802–806, 2005.

[Smith and Mailler, 2010] Melanie Smith and Roger Mailler. Getting what you pay for: Is exploration in distributed hill climbing really worth it? In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '10, pages 319–326, Washington, DC, USA, 2010. IEEE Computer Society.

[Tarlow *et al.*, 2010] Daniel Tarlow, Inmar E. Givoni, and Zemel. HOP-MAP: Efficient message passing with high order potentials. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 812–819, 2010.

[Wittenburg and Zhang, 2003] Lars Wittenburg and Weixiong Zhang. Distributed breakout algorithm for distributed constraint optimization problems – DBArelax. In *AAMAS'03*, pages 1158–1159, 2003.

[Yokoo and Hirayama, 1996] Makoto Yokoo and Katsutoshi Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi–Agent Systems*. MIT Press, 1996.

[Zhang *et al.*, 2005] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:1-2:55–88, January 2005.

[Zivan *et al.*, 2014] Roie Zivan, Steven Okamoto, and Hilla Peled. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence*, 212:1–26, July 2014.