DOI 10.12694/scpe.v16i4.1130
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 16, Number 4, pp. 403–421. http://www.scpe.org

MULTI-OBJECTIVE DISTRIBUTED CONSTRAINT OPTIMIZATION USING SEMI-RINGS

GRAHAM BILLIAU*, CHEE FON CHANG†, AND ADITYA GHOSE‡

Abstract. In this paper, we extend the Support Based Distributed Optimization (SBDO) algorithm to support problems which do not have a total pre-order over the set of solutions. This is the case in common real life problems that have multiple objective functions. In particular, decision support problems. These disparate objectives are not well supported by existing Distributed Constraint Optimization Problem (DCOP) techniques, which assume a single cost or utility function. As a result, existing Distributed COP techniques (with some recent exceptions) require that all agents subscribe to a common objective function and are therefore unsuitable for settings where agents have distinct, competing objectives. This makes existing constraint optimization technologies unsuitable for many decision support roles, where the decision maker wishes to observe the different trade-offs before making a decision.

Key words: Multi-objective Optimisation, Constraints, Semi-Rings.

AMS subject classifications. 65K10, 97H40

1. Introduction. The simple approach of modelling DCOPs, where the cost/utility of a solution is measured as a single real number, is quite restrictive. This approach can only represent problems where there is a total pre-order over the solutions. Many real world problems only have a partial order over the solutions. Examples of these problems include problems with multiple objectives and problems with qualitative valuations. To further complicate things, a problem may include a mix of qualitative and quantitive valuations.

Bistarelli's c-semiring framework [3] is very successful for modelling problems in the Constraint Optimisation Problem (COP) domain. The Distributed Constraint Optimisation Problem (DCOP) domain introduces several new modelling challenges over centralised problems. These challenges include agent responsibility, privacy, co-operating/competing agents and no global objective. Further, these challenges lead to classes of DCOP problems which have not been addressed in the literature so far. The existing modelling frameworks do not support these new classes of problems, so we are proposing a new modelling framework inspired by c-semirings.

We have identified three classes of problems within the DCOP domain which can not be adequately described using c-semirings; equitable problems, maximisation problems and 'committee' problems. For equitable problems, we specifically refer to problems with an objective of the form 'minimise the maximum difference between the best and worst criteria'. Examples of these problems include distributing rewards to a team, assigning jobs to workers and minimise the impact of flood mitigation. Some examples of maximisation problems include maximise profit within a supply chain, maximise the value of targets tracked in a sensor network and maximise the area a team of drones can search. Committee problems are when a group of agents must agree on one (or more) shared decisions. This includes problems such as deciding on a requirements specification or a CEO's bonus package.

Equitable problems are particularly interesting within a distributed system. This is because there may not be a single 'dictator' agent who defines the entire problem. Instead the problem may grow organically, as individual agents discover they share a common goal and agree to limited co-operation. While the agents may loosely agree on the common goal, each agent has other goals, which may conflict. The end result is that there may not be a single objective which all agents subscribe to. Further, individual agents may be selfish or altruistic, which determines how much weight they put on their local objectives. In these cases, a compromise (such as an equitable solution) is required.

Equitable problems often occur in the medical setting, where the objective is to balance the patients quality of life and their survival chances. To do this, the doctor must find a balance between the most aggressive treatment (maximum survival chances, minimal quality of life) and the least aggressive treatment (minimal

*Decision Systems Lab, University of Wollongong, NSW, Australia (gdb339@uow.edu.au)
†Centre for Oncology Informatics, University of Wollongong, NSW, Australia (cfchang@uow.edu.au)
‡Decision Systems Lab, University of Wollongong, NSW, Australia (aditya@uow.edu.au)

survival chances, maximum quality of life). Two examples of this include treating larynx cancer and methicillin-resistant staphylococcus aureus. Radiotherapy is an effective way to treat larynx cancer, but at high doses paralyses the patient's vocal cords. Similarly, amputation is the most effective way to treat staph infections, but then the patient looses the affected limb.

One possible way to represent an equitable problem is using a tuple of n real numbers to represent the utility for each agent. This can be represented using the semiring $\langle \Re \times \Re \times \cdots \times \Re, \oplus, \otimes \rangle$. The objective is to minimise the difference between the highest and lowest values in the tuple. Defining the $\oplus$ operator as follows captures this idea.

$$\oplus(A, B) = \begin{cases} A \; if \; (\max(A) - \min(A)) \leq (\max(B) - \min(B)) \\ B \; otherwise \end{cases}$$

where $A = \langle a_1, a_2, \ldots, a_n \rangle$ and $B = \langle b_1, b_2, \ldots, b_n \rangle$ are tuples of real numbers, so $\max(x)$ and $\min(x)$ return the largest or smallest value in the tuple respectively. The real numbers represent utility, so to get the total utility for an agent, the individual utilities are added together.

$$\otimes(A, B) = \langle a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n \rangle$$

This problem can't be represented as a c-semiring. A c-semiring requires a $\top$ value which is the absorbing element of $\oplus$ and the unit element of $\otimes$. It also requires a $\bot$ value which is the unit element of $\oplus$ and the absorbing element of $\otimes$. In this problem, there does not exist a unique 'best' value to use as $\top$ or a unique 'worst' value to use as $\bot$.

Egalitarian problems, a more common class of problems with a similar intuition also can't be represented as a c-semiring. By an egalitarian problem we specifically refer to problems with the objective "maximise the minimum utility" or "minimise the maximum cost". In these problems there is still no unique best or worst value. So long as one of the values is ∞, then any assignment to the other values will compare as the equal best (or worst) value. To represent such a problem using a c-semiring, the objective must be defined as "minimse the maximum cost, then minimise the next highest cost, ... then minimise the lowest cost". This objective still satisfies "minimise the maximum cost", though it also considers the other values.

Maximisation problems are often represented using valued constraints. They can also be represented using the semiring $\langle \Re, \max, + \rangle$. These problems can't be directly represented as c-semirings, as ∞ is the absorbing element for both max and $+$. In centralised settings, maximisation problems are commonly transformed into minimisation problems (represented by the c-semiring $\langle \Re_0^+, \min, +, 0, \infty \rangle$). This transformation involves first mapping $\Re$ to $\Re_0^-$ by subtracting ∞, then changing it to a minimisation problem by multiplying all the values by -1. Performing this transformation in practice requires knowing the maximum possible utility value. In dynamic or distributed settings, this is not possible. For dynamic settings, changes to the problem may result in a larger maximum possible utility value. For distributed settings, computing the largest possible utility value requires complete knowledge of the problem, which violates one of the assumptions of distributed problems. As the transformation is not applicable in these settings, the framework has to support the maximisation problem directly.

Finally we consider committee problems. This is a class of problems where a group of agents must agree on the answer to one (or several) decisions. Committee problems can be modelled as a standard DCOP by arbitrarily assigning agents control of decisions (variables). They could be modelled much more elegantly if agents are allowed to share control of variables. Problems of this form are usually modelled and solved as negotiation problems. If the committee problem is a sub-problem of a larger optimisation problem, then solving the larger problem requires a hybrid DCOP/negotiation algorithm or modelling the entire problem as a DCOP. Further, negotiation algorithms generally assume competitive agents, while for this work we assume cooperative, though not trusting, agents.

Committee problems are orthogonal to the utilitarian/equitable problems previously discussed. Whether a committee problem is a satisfaction, optimisation or even a multi-objective problem depends on the constraints and semiring chosen. If the agent's preferences are defined as valued constraints, then it is an optimisation problem.

We present a simple requirements engineering problem as an example of a committee problem. There are n candidate requirements (the variables, $\mathcal{X}$), which have been identified during the requirement elicitation process. Each candidate requirement can either be included (True) or excluded (False) from the specification (the domains, $D_1, \ldots, D_n$). In addition, there are m stakeholders (the agents, $\mathcal{A}$), each of whom has different preferences for which candidate requirements should be included (the constraints, $\mathcal{C}$). Finally, the objective is to maximise the total utility of the stakeholders (maximisation problem, $\mathcal{V}$).

In this setting all of the stakeholders know all of the candidate requirements and may propose a value for any of them. The stakeholders would like to keep their preferences private. Modelling this as a traditional DCOP problem would require arbitrarily assigning stakeholders control over the candidate constraints. Further, if we assume that each stakeholder has preferences regarding most of the constraints, then they will be forced to reveal their preferences to most of the other stakeholders. This is because most DCOP solvers require all agents involved in the constraint must know the details of the constraint. Recent work on asymmetric constraints [8] goes some way towards addressing this concern. By allowing multiple agents to share control of variables, this problem can be modelled naturally. The resulting model is also better at maintaining each agent's privacy.

These classes of problems may overlap, as shown in the following example. There are two agents {1, 2} sharing control of two variables {X, Y} with the same domain $\{-1, 0, 1\}$. The global objective is that the sum of the variables should be zero. Agent 1's objectives are to maximise X and minimise Y. Agent 2's objectives are to minimise X and maximise Y.

The (utilitarian) optimal solutions for this problem are:

- $X = 1, Y = -1$ (In favour of Agent 1)
- $X = -1, Y = 1$ (In favour of Agent 2)

There is one equitable solution:

- $X = 0, Y = 0$ (balance between both Agents)

A utilitarian solver will likely not return the equitable solution as a possible solution. While the equitable solution satisfies the global objective, it does not satisfy either of the local objectives.

In section 2 we describe the framework which we propose to model these classes of problems. Next we show the relationship between our framework and some of the established frameworks in section 2.3. We then detail an algorithm designed to solve problems expressed using this framework in section 3 and show the performance of the algorithm in section 4. Finally the conclusion summarises our contribution.

2. Semiring-based Distributed Constraint Optimization Problems. In this section, we will introduce our proposed framework, the Semiring-based Distributed Constraint Satisfaction/Optimization Problem (SDCOP). The proposed framework draws inspiration from Bistarelli et al. [3]. As such, there exists many commonalities. Bistarelli et al. [3] proposed the use of a *c-semiring* for comparing different solutions. The original formulation of a c-semiring was for use within the CSP domain. We view this approach to be too restrictive for use within the CSOP domain. Specifically, a c-semiring does not support optimization problems where the objective is to maximize utility. Objective functions of this form are required for some recent algorithms (specifically SBDO [2]). Thus, we propose the use of an idempotent semiring. The use of the semiring structure allows for two main benefits. Firstly, in multi-objective problems where no total pre-order over the solutions are prescribed, the use of an abstract structure (i.e. semirings) allows the framework to induce a partial ordering. Secondly, using an abstract comparison and aggregation operators allows the framework to support comparison and combination across both qualitative and quantitative domains.

2.1. Preliminaries: Idempotent Semirings. A semiring consists of a set of abstract values and two operators. The two operators allows for the comparison and combination of the prescribed abstract values.

DEFINITION 2.1. *A* **semiring** *[17] is a tuple $\mathcal{V} = \langle V, \oplus, \otimes \rangle$ satisfying the following conditions:*

- *V is a set of abstract values.*
- *$\oplus$ is a commutative, associative and closed operator over V.*
- *$\otimes$ is an associative and closed operator over V.*
- *$\otimes$ left and right distributes over $\oplus$.*

We shall call a semiring an **idempotent semiring** *if $\oplus$ is idempotent.*

DEFINITION 2.2. *[3] A c-semiring is a tuple $\mathcal{V} = \langle V, \oplus, \otimes, \bot, \top \rangle$ satisfying (for all $\alpha \in V$):*

- *V is a set of abstract values with $\bot, \top \in V$.*

- $\oplus$ *is defined over possibly infinite sets as follows:*
  - $\forall v \in V, \oplus(\{v\}) = v$
  - $\oplus(\emptyset) = \bot and \oplus(V) = \top$
  - $\oplus(\bigcup v_i, i \in S) = \oplus(\{\oplus(v_i), i \in S\})$ *for all sets of indices* S
- $\otimes$ *is a commutative, associative and closed binary operator on* V *with* $\top$ *as unit element* ($\alpha \otimes \top = \alpha$) *and* $\bot$ *as absorbing element* ($\alpha \otimes \bot = \bot$).
- $\otimes$ *distributes over* $\oplus$ *(i.e.,* $\alpha \otimes (\beta \oplus \gamma) = (\alpha \otimes \beta) \oplus (\alpha \otimes \gamma)$).

The idempotent property of the $\oplus$ operator can be used to obtain a partial order $\preceq_V$ over the set of abstract values V. Such a partial order is defined as: $\forall(v_1, v_2 \in V), v_1 \preceq_V v_2$ iff $v_1 \oplus v_2 = v_1$ (intuitively, $v_1 \preceq_V v_2$ denotes that v_1 is at least as preferred as v_2). The $\oplus$ operator enables comparisons between two semiring values while the $\otimes$ operator allows us to aggregate two semiring values.

This idempotent semiring structure is now capable of representing all the different constraint schemes. As a c-semiring is an idempotent semiring, all constraint schemes that can be represented as c-semirings can also be represented as idempotent semirings.

Bistarelli has shown that classic, fuzzy, probabilistic, weighted and set based constraints are all instances of a c-semiring [3]. Valued constraints with a maximization objective are not an instance of a c-semiring, but can be represented by the idempotent semiring $\langle \Re, \max, + \rangle$ where $\Re$ is the set of values, max is the comparison operator and + is the combination operator.

Multiple idempotent semirings may be combined into a single idempotent semiring in the same way that c-semirings are combined [3]. The semirings being combined may involve evaluations on multiple heterogeneous scales - both qualitative and quantitative. We leverage this property in handling multi-objective DCOPs.

The following definition is based on that provided by Bistarelli et. al [3] (definition 7.1) for c-semiring. It formalizes the combination of idempotent semirings.

DEFINITION 2.3. *Given the* n *idempotent semirings* $S_i = \langle V_i, \oplus_i, \otimes_i, \rangle$ *for* $i = 1, \ldots, n$ *we define the structure* $Comb(S_1, \ldots, S_n) = \langle \langle V_1, \ldots, V_n \rangle, \oplus, \otimes \rangle$ *where* $\oplus$ *and* $\otimes$ *are defined as follows: Given* $\langle a_1, \ldots, a_n \rangle$ *and* $\langle b_1, \ldots, b_n \rangle$ *such that* $a_i, b_i \in V_i$ *for* $i = 1, \ldots, n, \langle a_1, \ldots, a_n \rangle \oplus \langle b_1, \ldots, b_n \rangle = \langle a_1 \oplus_1 b_1, \ldots, a_n \oplus_n b_n \rangle$ *and* $\langle a_1, \ldots, a_n \rangle \otimes \langle b_1, \ldots, b_n \rangle = \langle a_1 \otimes_1 b_1, \ldots, a_n \otimes_n b_n \rangle$.

THEOREM 2.4. *If* $S_i = \langle V_i, \oplus_i, \otimes_i \rangle$ *for* $i = 1, \ldots, n$ *are all idempotent semirings, then* $Comb(S_1, \ldots, S_n)$ *is an idempotent semiring.*

Proof. From definition 2.3, the combined semiring uses the $\oplus$ and $\otimes$ operators from the component semirings directly. Hence, the properties that hold for the component semirings also hold for the combined semiring.□

If $a \oplus b = a$ then for all components i, $a_i \oplus_i b_i = a_i$. This corresponds to the 'dominates' concept in pareto-optimality. Hence the $Comb()$ operator implements the commonly accepted pareto-optimal ordering over multiple objectives. If a different ordering is desired then the $\oplus$ operator can be redefined to implement the desired ordering.

2.2. Distributed Constraint Optimization Problems. In Constraint Satisfaction Problems (CSPs), the constraints within the problem and the criteria that the solution must satisfy can be viewed as one and the same. However Constraint Satisfaction/Optimisation Problems (CSOPs) allow both objectives and constraints as criteria that the solution must satisfy, so the three concepts must be addressed separately. Hence the core concept in CSOPs is to assign a value to each of the variables from the variables' domain such that not only the constraints are satisfied but also the set of optimization criteria are satisfied. Due to the structure of a CSOP, objectives cannot be realized directly, they must be realized via intermediate soft constraints. For example, in valued CSOPs, the criteria is to minimize (resp. maximize) the total value of the soft constraints. This requires soft constraints that return a real number, rather than true or false. Using optimization criteria such as minimize or maximize has other difficulties, as a naive solver must explore the entire solution space to determine if the criteria is satisfied. Furthermore, the relaxation of an over-constrained problem yields a CSOP. Over-constrained problems can be converted by either relaxing the constraints or the solutions. Relaxing the constraints yields an optimization criteria on the constraints (satisfy the maximum number of constraints). While relaxing the solutions means that some variables can be left without a value assigned, leading to the objective: maximize the number of variables assigned a value. If there exists more than one optimization criteria, then often there is no solution that satisfies all of them. So, the criteria is normally relaxed to find a pareto-optimal solution.

A unified formulation must also support the concept of agents. The concept of agents is required for distributed problems. An agent has knowledge of a subset of the entire problem and can only affect a subset of the sub-problem it knows. There are several common justifications (such as resource limits, privacy concerns and communication capacity) for limiting an agent's knowledge to only a subset of the entire problem. Furthermore, resource and communication limitations can lead to efficiencies, such as a reduction of memory requirements for each agent and a reduction in required messages to keep the knowledge of all agents synchronized. An agent may also not know of variables and constraints that represent privileged information held by another agent.

DEFINITION 2.5. *A Semiring-Based Distributed Constraint Satisfaction/Optimization Problem (SDCOP) is a tuple* $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{V}, \mathcal{C} \rangle$ *where*

- $\mathcal{A}$ *is a non-empty set of agents. An agent is a pair* $\alpha = \langle R_\alpha, W_\alpha \rangle$. R_a *is a set of variables for which the agent has* **read privileges**, *and* $W_\alpha \subseteq R_\alpha$ *is a set of variables for which the agent has* **write privileges**.
- $\mathcal{X}$ *is a set of variables.*
- $\mathcal{D}$ *is a set* $\{D_1, \ldots, D_n\}$ *where* $n = |\mathcal{X}|$ *and each* D_i *is a set of values to be assigned to a variable (the domain).*
- $\mathcal{V} = \langle V, \oplus, \otimes \rangle$ *is an idempotent semiring utilized to evaluate variable assignments.*
- $\mathcal{C}$ *is a non-empty set of constraints, where each constraint* c_i *is a pair* $\langle def_i, con_i \rangle$ *where* def_i *is a function* $def_i : (\bigcup \mathcal{D})^k \longrightarrow V$ *(where* $k = \mid con_i \mid$*) and* $con_i \subseteq \mathcal{X}$ *is the* **signature** *of constraint* c_i *(i.e., the set of variables referred to in that constraint).*

If an agent α *has write privileges for a variable* v, α *must also have read privileges for all variables which share a constraint with* v *i.e.* $\forall v \in W_\alpha, \forall c_i \in \mathcal{C}$, *if* $v \in con_i$, *then* $con_i \subset R_\alpha$.

The set of agents refers to the agents that must co-operate to solve the problem. There is of course much more to an agent than just its privileges, such as the amount of resources available or the 'temperament' of the agent. We do not consider those attributes (and similar ones) of agents as they only apply to solving the problem, not to the description of the problem.

Read privileges serve a dual purpose. The primary purpose is to identify which agents must know the value assigned of a variable. The secondary purpose is to determine the required communication links between agents. Knowledge of variable assignments is only one aspect of privacy. Often it is also desirable to ensure that other agents can not discover the constraints on an agent's variables. The flow of such information is entirely dependent on the algorithm and so is outside the scope of this work.

As the required communication is already prescribed by read privileges, the write privileges are purely to determine which agents have permission to allocate a value to a variable. In most situations, it is acceptable for there to be variables in the problem for which no agent has write privileges. These can be viewed as hard constraints or environmental constants. This concept is particularly useful in a mixed-initiative setting whereby assignments made by the human operator should not and cannot be overruled by the machine solver.

The explicit assertion of read/write privileges is unique to this formulation. Traditional formulations implicitly prescribe global read privileges while maintaining only local write privileges. We feel that this approach over simplifies real world problems. Such an assumption creates additional communication overhead to maintain synchronization and impedes on the scalable and distributed nature of DCSOPs. Furthermore, to honestly reflect the privacy property in DCOPs, the read/write privileges prescribes access control machinery to control access to information. However, note that the traditional formulation (global read/local write) approach is a special case for our formulation where agents are given read privileges to all variables. It also allows for parameters (variables for which no agent has write access) to be naturally represented within the problem and allows the community to explore problems without the simplifying assumption that agents have exclusive control over variables. Designing algorithms which support allowing multiple agents to write to a variable presents new challenges. The algorithm must have some mechanisim for either managing access to the variable or resolving the resulting inconsistancies.

The definition of read and write privileges allows a graph representing the commiunication channels between agents to be definied. Each node in the graph represents an agent and there is an edge between two nodes if the respective agents must communicate. Two agents must communicate if they each control a variable which contributes to the same constraint or objective. To minimize communication bandwidth and privacy loss, the

communication channels between agents exist only in situations where the agents share variables. More formally,

Definition 2.6. *Given a SDCOP* $= \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{V}, \mathcal{C} \rangle$*, a* **neighbourhood graph** *is a directed graph* $\langle N, E \rangle$ *where* $N = \mathcal{A}$ *and* $E \subseteq \mathcal{A} \times \mathcal{A}$ *such that* $\forall \alpha, \alpha' \in \mathcal{A}, \langle \alpha, \alpha' \rangle \in E$ *iff* $\alpha \neq \alpha' \wedge \exists x \in \mathcal{X}$ *such that* $x \in W_\alpha \wedge x \in R_{\alpha'}$*.*

If an agent α has write privileges to a variable x and another agent α' has read privileges to x then they must be able to communicate, as α must inform α' of any changes it makes to x. If two agents must communicate then they are in the same neighbourhood.

Variables are identified by a unique name. All variables share the same domain, rather than having separate domains as per common practice. The common domain can be viewed as the union of all individual domains, with unary constraints on the individual variables restricting which subset of the domain values can be assigned to the variables. This approach generalizes existing practice. The idea of having the domain of each variable defined by unary constraints was discussed by Ross et. al. [18]

Our use of functions that return a value from an idempotent semiring allows for the representation of a range of different forms of constraints. In classic satisfaction problems, the semiring $\langle \{\text{True}, \text{False}\}, \vee, \wedge \rangle$ is sufficient to reflect the satisfaction of the constraints. For valued constraint optimization problems, the semiring $\langle \Re, \min, + \rangle$ is suitable for minimization problems and $\langle \Re, \max, + \rangle$ is suitable for maximization problems. In the cases where there are multiple objective functions, each objective is represented by its own idempotent semiring. The *Comb*() operator can then be utilized to combine the individual semirings. Using this approach to combine the different objectives naturally leads to a search for a pareto-optimal solution. By using this approach we do not impose an ordering on the objectives, however an ordering can be added if so desired. Furthermore, such an approach also allows for all the different types of constraints to be combined.

We also distinguish between two classes of constraints: local constraints and shared constraints. A local constraint is known by only one agent and can only be evaluated by that agent. For this to occur one agent must have write privileges to all of the variables in con_i (refer to definition 2.5). Shared constraints are known by all agents which have write privileges to a variable in the constraints signature and can be evaluated by any one of those agents. This occurs when more than one agent has write privileges to one of the variables in con_i (refer to definition 2.5).

Definition 2.7. *Given an SDCOP* $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{V}, \mathcal{C} \rangle$*,* $\mathcal{S}$ *is the set of all possible assignments to variables. Each assignment is a set of variable-value pairs where no variable appears more than once. We will refer to an assignment* $s \in \mathcal{S}$ *as a* **complete assignment** *if it assigns a value to every variable in* $\mathcal{X}$*. For some assignment* $s \in \mathcal{S}$*, we will use* $s \downarrow_X$ *to denote the* projection *of the assignment to a set of variables* X *(i.e., the subset of* s *that refers to variables in* X*). Given an assignment* $s = \langle \langle x_1, v_1 \rangle, \langle x_2, v_2 \rangle, \ldots, \langle x_k, v_k \rangle \rangle$*,* $val(s) = \langle v_1, v_2, \ldots, v_k \rangle$ *and* $var(s) = \langle x_1, x_2, \ldots, x_k \rangle$*.*

The different criteria are reflected in the definition of a solution to a SDCOP. The concept of a solution to a satisfaction problem is generalized in our definition of an acceptable solution.

Recall that our unified definition of a CSP is agnostic to the solving method, so while it has been shown [16] that combining semirings is not sufficient for solving multi-objective problems using inference based methods [1, 5, 6, 7, 15] (specifically the operators $\bigoplus$ and $\bigotimes$). However, combining semirings is sufficient for search based methods [2, 20], which are common in the DCOP literature. For this reason we define the solutions to a SDCOP in terms of the properties they must satisfy, which are independent of the solving method and the semiring used to evaluate assignments. Furthermore, we utilize a notion of acceptable threshold to define an acceptable solution to an optimization problem. The idea of an acceptable threshold on an optimization problem has been proposed by J. Larrosa [11].

Definition 2.8. *An* **acceptable solution** *to a SDCOP with* $\mathcal{C} = \{c_1, \ldots, c_n\}$ *is a complete assignment* s *such that* $def_1(val(s \downarrow_{con_1})) \otimes \ldots \otimes def_n(val(s \downarrow_{con_n})) \preceq_V v$*, where* $v \in V$ *is a minimum threshold on the value of the solution.*

The solution to a classic satisfaction problem can be represented as an acceptable solution with a threshold of True. An acceptable solution is also useful to represent problems where the optimal solution is not required or it is too expensive to search for the optimal solution.

Definition 2.9. *An* **optimal solution** *to a SDCOP with* $\mathcal{C} = \{c_1, \ldots, c_n\}$ *is a complete assignment* s *such that there does not exist another complete assignment* s' *where* $def_1(val(s' \downarrow_{con_1})) \otimes \ldots \otimes def_n(val(s' \downarrow_{con_n})) \prec_V def_1(val(s \downarrow_{con_1})) \otimes \ldots \otimes def_n(val(s \downarrow_{con_n}))$ *(note:* $a \prec_V b$ *iff* $a \preceq_V b$ *and* $b \not\preceq_V a$*).*

THEOREM 2.10. *At least one optimal solution exists for any SDCOP.*

Proof. The associative property of $\oplus$ means that the ordering $\preceq_V$ derived from $\oplus$ is transitive. Due to the ordering being transitive, cycles can not exist within the ordering. Because there are no cycles there must be at least one abstract value which is not dominated by another abstract value.

The constraints map each assignment to exactly one abstract value, which is used to order the assignments. As such, the ordering over assignments is also transitive. Therefore there must be at least one complete assignment which is not dominated by another complete assignment.□

Note that the definition of an optimal solution is equivalent to a non-dominated solution, as such the set of all optimal solutions is the pareto-frontier. If an acceptable solution (for a given threshold) does not exist, one way to get a solution which is acceptable is to relax the concept of a solution. This is done by not requiring a complete solution to the SDCOP.

DEFINITION 2.11. *Given a constraint $c_i = \langle def_i, con_i \rangle$ the projection of c_i onto a set of variables X, $c_i \downarrow_X = \langle def'_i, con_i \cap X \rangle$. If $con_i \subseteq X$ then c_i is returned unchanged. def'_i is defined such that for a given input V it returns one of the values def_i can return for the input V expanded to include values for the other variables in con_i.*

There are two different intuitions about which value to return in the projected version of the constraint. First, the projected constraint should return the best possible value using the assigned variables. Second, the projected constraint should return the worst possible value using the assigned variables. It is possible that there is more that one best or worst value, as the semiring allows a partial ordering. In this case the choice between the best or worst values is arbitrary.

DEFINITION 2.12. *Given a set of constraints $\mathcal{C} = \{c_1, \ldots, c_n\}$ and an assignment s, let $\mathcal{C}' = \{c_1 \downarrow_{var(s)}, \ldots, c_n \downarrow_{var(s)}\}$. s is a* **relaxed solution** *with reference to the constraints $\mathcal{C}'$ iff $def_1(val(s \downarrow_{con_1})) \otimes \ldots \otimes def_n(val(s \downarrow_{con_n})) \preceq_V v$, where v is a minimum threshold on the value of the solution and there does not exist another assignment s' where $|s'| > |s|$ and $def_1(val(s' \downarrow_{con_1})) \otimes \ldots \otimes def_n(val(s' \downarrow_{con_n})) \preceq_V v$.*

There are many other ways to define a solution to a SDCOP, such as a k-optimal solution [13]. These other solution concepts can be easily modified for the SDCOP structure.

2.3. Example Instantiations. In this section, we will illustrate the usefulness of our framework by the SDCOP instances that correspond to several common DCOPs. We will do so by presenting three instantiations: a CSP instance, a DCOP instance and a DCOP variation. Bistarelli's c-semiring framework [3] is one of the commonly accepted frameworks for generalizing CSP problems. Both Bistarelli's c-semiring framework and our SDCOP framework are based on using a variation of a semiring to order the solutions. As highlighted earlier, CSPs are special instances of DCOPs. Hence, it is befiting to illustrate the generality of SDCOP by demonstrating Bistarelli's c-semiring framework is a special instance of an SDCOP.

THEOREM 2.13. *Any problem represented as a c-semiring can be represented as an idempotent semiring. Any idempotent semiring which also satisfies the following properties can be represented as a c-semiring.*

- *$\bot_i \in V$ is the absorbing element of $\otimes$ and the unit element of $\oplus$.*
- *$\top_i \in V$ is the unit element of $\otimes$ and the absorbing element of $\oplus$.*

Proof. We consider two representations of a constraint optimisation problem to be equivalent iff the solutions of the representations are equal. The solution to a constraint optimisation problem is defined in terms of the $\oplus$ and $\otimes$ operators. Therefore it is sufficient to show that the $\oplus$ and $\otimes$ operators in c-semirings and idempotent semirings have the same properties.

Given a c-semiring $\langle V_c, \oplus_c, \otimes_c, \bot_c, \top_c \rangle$ and an idempotent semiring $\langle V_i, \oplus_i, \otimes_i, \bot_i, \top_i \rangle$ as described above, we show that the two are equivalent.

V_i and V_c are equivalent by definition, as are $\otimes_i$ and $\otimes_c$. $\bot_i, \bot_c$ are the absorbing elements of $\otimes_i, \otimes_c$ respectively, as per the definition. Similarly $\top_i, \top_c$ are the unit elements of $\otimes_i, \otimes_c$ respectively.

It remains to show that $\oplus_i$ is equivalent to $\oplus_c$. $\oplus_c$ is defined over a set of operands, while $\oplus_i$ is strictly a binary operator, as such the behaviour of $\oplus_c$ with zero or one operands is not relevant. $\oplus_i$ is commutative, associative and idempotent, so it follows that $\oplus_i(\{v_i, v_{i+1}, \ldots, v_j\}) = v_i \oplus v_{i+1} \oplus_i \cdots \oplus_i v_j$ for all sets of indices $i, i+1, \ldots, j$. It is shown in Bistarelli's paper that $\oplus_c$ is idempotent, commutative and associative. By specifying that $\top$ is the absorbing element of $\oplus_i, v_1 \oplus v_2 \oplus_i \cdots \oplus_i v_n = \top$ for $v_1, \ldots, v_n \in V$, which is equivalent to $\oplus_c(V) = \top_c$. $\oplus_i(\mathcal{V}\rangle) = \top$ will be the case iff $\top$ is the absorbing element of $\mathcal{V}\rangle$. □

We have just shown that the idempotent semiring used in our framework is more general than the c-semiring used in Bistarelli's framework [3]. Therefore, all the constraint schemes which can be represented in Bistarelli's framework can also be represented in our framework. Namely classical, fuzzy, probabilistic, weighted (minimization), set based, and multi-objective constraint optimization problems.

There exists several formulations of DCOPs [2, 7, 12, 14, 20]. All of these formulations capture the important aspects of DCOPs equally well. We will focus on the instantiation of Petcu's formulation [14] into SDCOP as it is the most formal definition.

Petcu [14] defines a COP as follows:

DEFINITION 2.14. *[14] A* discrete *constraint optimization problem (COP) is a tuple* $\langle \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$ *such that:*

- $\mathcal{X} = \{X_1, \ldots, X_n\}$ *is a set of* variables *(e.g. start times of meetings);*
- $\mathcal{D} = \{d_1, \ldots, d_n\}$ *is a set of discrete, finite variable* domains *(e.g. time slots);*
- $\mathcal{R} = \{r_1, \ldots, r_m\}$ *is a set of* utility functions*, where each* r_i *is a function with the scope* $(X_{i_1}, \ldots, X_{i_k})$, $r_i : d_{i_1} \times \cdots \times d_{i_k} \longrightarrow \Re$*. Such a function assigns a* utility (reward) *to each possible combination of values of the variables in the scope of the function. Negative amounts mean costs. Hard constraints (which forbid certain value combinations) are a special case of utility functions, which assign* 0 *to feasible tuples, and* $-\inf$ *to infeasible ones;*

DEFINITION 2.15. *[14] A discrete distributed constraint optimization problem (DCOP) is a tuple of the following form:* $\langle \mathcal{A}, \mathcal{COP}, \mathcal{R}^{ia} \rangle$ *such that:*

- $\mathcal{A} = \{A_1, \ldots, A_k\}$ *is a set of agents (e.g. people participating in meetings);*
- $\mathcal{COP} = \{COP_1, \ldots, COP\}$ *is a set of disjoint, centralized COPs; each* COP_i *is called the local sub-problem of agent* A_i*, and is owned and controlled by agent* A_i*;*
- $\mathcal{R}^{ia} = \{r_1, \ldots, r_n\}$ *is a set of inter-agent utility functions defined over variables from several different local sub-problems* COP_i*. Each* $r_i : d_{i_1} \times \cdots \times d_{i_k} \longrightarrow \Re$ *expresses the rewards obtained by the agents involved in* r_i *for some joint decision. The agents involved in* r_i *have full knowledge of* r_i *and are called 'responsible' for* r_i*. As in a COP, hard constraints are simulated by utility functions which assign* 0 *to feasible tuples, and* $-\infty$ *to infeasible ones;*

Further, Petcu defines a solution as:

DEFINITION 2.16. *[14] The goal is to find a complete instantiation* $\mathcal{X}$ *for the variables* X_i *that* maximizes *the sum of utilities of individual utility functions.* The definition of a solution provided by Petcu [14] corresponds to the definition of an optimal solution within SDCOP. This definition of a DCOP corresponds to the following SDCOP:

THEOREM 2.17. *Petcu's definition of a DCOP is a SDCOP* $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{V}, \mathcal{S}, \mathcal{C} \rangle$ *with the additional properties:*

- *Exactly one agent has write access for each variable.*
- $\bigcup \mathcal{D}$ *is finite.*
- $\mathcal{V} = \langle \Re, max, + \rangle$*.*

Proof. The two formulations are equivalent iff every problem that can be described by Petcu's definition has an equivalent definition within the SDCOP and every problem that can be described by the SDCOP has an equivalent definition within Petcu's definition.

We start by showing that every problem that can be described by Petcu's definition has an equivalent definition within the SDCOP. It is clear that both formulations support the concept of agents. Petcu's definition allows an agent to control a set of variables described as a sub-problem while SDCOP directly specifies the variables that an agent controls. Petcu's use of real numbers to measure utility can easily be represented by the idempotent semiring $\langle \Re, \max, + \rangle$

Now we show that every problem that can be described by the SDCOP has an equivalent definition within Petcu's definition. Both definitions support the concept of agents. The first additional restriction on a SDCOP results in each agent controlling a distinct set of variables. The variables which an agent controls form the agent's sub-problem in Petcu's definition. There is no formal concept of read privileges in Petcu's definition, instead it is simply assumed that if an agent knows a constraint, it has read privileges for all variables in the constraint, which is the minimal read privileges required for SDCOP. SDCOP uses a single domain for all variables, as opposed to separate finite domains for each variable. The second restriction limits SDCOP to a finite domain, which can be transformed into variable specific domains by first assigning each variable the

entire domain, then propagating all the unary constraints on that variable. The resulting domain is specific domain for that variable. After this the unary constraints can be removed. Restriction three limits SDCOP to using real numbers as the utility, which is equivalent to Petcu's use of real numbers. Finally the use of n-ary constraints in SDCOP is almost identical to the constraints in Petcu's definition.□

Recently, Grinshpoun et al. proposed the Asymmetrical DCOP (ADCOP) model [8]. In this model, the utility gained by each agent participating in a constraint is tallied separately, however the objective is to minimize (maximize) the sum of the utilities of all agents. The motivation is to preserve the privacy of each agent regarding its local utility. Grinshpoun et al. [8] define a DCOP as follows:

DEFINITION 2.18. *[8] A DCOP is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $A_1, A_2, \ldots, A_n$. $\mathcal{X}$ is a finite set of variables $X_1, X_2, \ldots, X_m$. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains $D_1, D_2, \ldots, D_m$. Each domain D_i contains a finite set of values which can be assigned to variable X_i . $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in R$ defines a non-negative cost for every possible value combination of a set of variables, and is of the form:*

$$C : D_{i1} \times D_{i2} \times \cdots \times D_{ik} \longrightarrow \Re^{+} \tag{2.1}$$

Grinshpoun et al. [8] define an ADCOP as follows:

DEFINITION 2.19. *[8] An ADCOP is defined by the following tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$, where $\mathcal{A}$, $\mathcal{X}$, and $\mathcal{D}$ are defined in exactly the same manner as in DCOPs. Each constraint $C \in R$ of an asymmetric DCOP defines a set of non-negative costs for every possible value combination of a set of variables, and takes the following form:*

$$C : D_{i1} \times D_{i2} \times \cdots \times D_{ik} \longrightarrow \Re^{+^k} \tag{2.2}$$

Grinshpoun's definition of a DCOP is comparable to Petcu's, which we have already shown to be an instance of of an SDCOP. It remains to show that the constraint definition in an ADCOP is an instance of an idempotent semiring.

THEOREM 2.20. *Asymmetrical constraints as defined in a ADCOP can be represented as the an idempotent semiring $\langle V, \otimes, \oplus \rangle$ where: V is the Cartesian product of the set of utilities for each agent. $\Re_{+}^{n}$, given agents $a_1, \ldots, a_n$, $\otimes$ is defined as $\otimes(v, v') = \langle v_1 + v'_1, v_2 + v'_2, \ldots, v_n + v'_n \rangle$ and $\oplus$ is defined as*

$$\oplus(v, v') = \begin{cases} v & \Sigma_n^1 v \leq \Sigma_n^1 v' \\ v' & otherwise \end{cases} \tag{2.3}$$

In addition, if there does not exist a variable which is owned by an agent α, in the signature of a constraint c, then the cost for α in the result of the constraint c must be zero.

Proof.

The ADCOP definition allows the cost for each agent to be recorded separately, by having one integer/real number for each agent, the idempotent semiring described also allows the cost for each agent to be recorded separately. The aggregation operator in ADCOP is sum, with the cost for each agent aggregated separately, this is reflected in the $\otimes$ operator. The comparison operator in ADCOP is a utilitarian minimization operator, this is reflected in the $\oplus$ operator.

The idempotent semiring always includes a cost for all agents, even if the agent is not involved in the constraint, thus it may represent constraints that can't be represented in an ADCOP. The restriction on the cost for an agent if the agent is not part of the constraint prevents this case. □

3. Semi-Ring Support Based Distributed Optimisation. The Support Based Distributed Optimisation (SBDO) algorithm [2], which this algorithm extends, is intended to be deployed in real life environments. We characterise real life environments as having the following properties:

- Dynamic
- Anarchic
- Unreliable agents/communication

- Hetrogeneous agents
- Selfish agents
- Malicious agents

These are in addition to the properties of the problem, such as different constraint types and multiple objectives.

In order to operate in this environment, we identified three ideals that the algorithm should meet. These are autonomy, equity, and fault tolerance. Autonomy means that the agent retains control of its local knowledge and has the power to act or not act as its situation dictates. Equity means that each agent has the same influence over the final solution and requires a similar amount of resources to the other agents. Fault tolerance means that the overall system should not fail when any of its components fails.

In order to attempt to meet these ideals, SBDO is based on negotiation/argumentation between the agents, rather than a power structure as commonly used. Because of this, the agents have equal power over the other agents and the final solution to the problem. In the process of solving the problem, the agents dynamically form coalitions and use their combined power to influence the other agents. These coalitions are determined based on the structure of the problem and random chance, rather than any bias in the algorithm. The lack of an enforced structure means that the algorithm is not unduly affected should an agent fail, and can continue attempting to solve the problem without the failed agent. The redundancy built into the message passing means that an agent which fails can be recovered quickly. The communication protocol used is also not sensitive to messages arriving in the wrong order. Together these make the algorithm tolerant of a variety of different faults.

The version of SBDO described in this chapter can solve SDCOP problems which meet the following conditions:

- The domain of variables ($\bigcup \mathcal{D}$) must be finite.
- The quality of a solution must be monotonically non-decreasing as the solution is extended, i.e. $\forall a, b \in V, a \otimes b \preceq a$.

How SBDO and SBDOsr support dynamic problems is discused in earlier publications [2], so is not discused here.

We start our discussion of the SBDOsr algorithm by defining the messages sent between agents.

Definition 3.1. *An* **assignment** *is a triple* $\langle a, v, u \rangle$ *where* a *is an agent in the SDCOP,* v *is a set of variable-value pairs, and* u *is the utility of this assignment returned by the agents local constraints.* v *must contain a variable-value pair for every variable in* W_a *for which another agent has read privileges.*

Definition 3.2. *Given an SDCOP* $= \langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{V}, \mathcal{C} \rangle$*, a* **proposal** *is a pair* $\langle VA, SCE \rangle$*, where VA (variable assignments) is a sequence* $\langle ass_1, \ldots, ass_n \rangle$ *of assignments such that the sequence of agents forms a simple path through the neighbourhood graph and there are no conflicting assignments. SCE (shared constraint evaluations) is a set of evaluations of shared constraints. A shared constraint can only be evaluated if an assignment to every variable involved in the constraint is included in VA. Each evaluation is a tuple* $\langle o, u \rangle$ *where* o *is a shared objective and* u *is the utility returned by the objective function* o *given the assignments in* VA*.*

As an example, consider the two agents A and B, who share a constraint O, as well as having their own local constraints. When A first creates a proposal it simply contains an assignment to A, $\langle \langle \langle A, \{\langle a, 1\rangle\}, 3\rangle\rangle, \{\}\rangle$. Later when B extends the proposal, B then has enough information to evaluate the shared objective, producing $\langle \langle \langle A, \{\langle a, 1\rangle\}, 3\rangle, \langle B, \{\langle b, 2\rangle\}, 2\rangle\rangle, \{\langle O, 10\rangle\}\rangle$.

The sequence of variable assignments indicates the order in which this proposal has been constructed and is required both to store the utility of the solution and for the generation of nogoods. The set of shared constraint evaluations is required to record the utility of constraints shared by more than one agent. If the utility of the shared constraints is combined with the local constraints they might be double counted when cycles form. Note that in situations where privacy is important, the assignment to a variable only needs to be disclosed if another agent has a constraint involving this variable.

The utility of an assignment can be determined by evaluating the applicable functions in $\mathcal{C}$ and aggregating them using $\otimes$. The total utility of an proposal is determined by applying the aggregation ($\otimes$) operator over the utility of each assignment and evaluation. As such a proposal encodes a partial solution to the problem as well as the relative utility of the partial solution. When a proposal is considered as an argument the first $n-1$ assignments form the justification and the last assignment is the conclusion.

The utility value provides a partial order over the the proposals (partial solutions). Comparison between proposals can be performed by first applying the $\otimes$ operator over the utility of each assignment and evaluation to determine its utility value and then the $\oplus$ operator to determine which proposal is better. Whenever we refer to one proposal being better than another in this paper it is with respect to this induced ordering.

The counterpart of a proposal is a nogood. A nogood represents a partial solution that violates at least one constraint and should never be reconsidered or included in the final solution. This inconsistency is discovered when the utility of an isgood is $\perp$. Nogoods with justifications [19] are used as these allow us to guarantee that all the hard constraints are satisfied (as shown in [10]) as well as allowing obsolete nogoods to be identified after the constraints that the nogood violated are removed from the problem. For our purposes, a nogood with justification (originally defined in [19]) is treated as follows:

DEFINITION 3.3. *Given an SDCOP $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{V}, \mathcal{C} \rangle$, a* **nogood** *is a pair $\langle P, C \rangle$ where P is a set of variable-value pairs representing a partial solution and $C \subseteq \mathcal{C}$ is the set of constraints that provides the* **justification** *for the nogood, such that the combination of s and C is inconsistent (results in $\perp$).*

A **minimal nogood** *is a nogood $n = \langle P, C \rangle$ such that there does not exist a nogood $n' = \langle P', C \rangle$ where $P' \subset P$ or a nogood $n'' = \langle P, C' \rangle$ where $C' \subset C$.*

In static environments, detecting that the network has reached a quiescent state is sufficient to detect termination. This can be achieved by taking a consistent global snapshot [4]. The algorithm will also terminate if it detects that there is no solution to the problem, by generating the empty nogood. Otherwise, due to the dynamic nature of the input problem, the algorithm will only terminate when instructed to by an outside entity. Detecting that the network of agents has reached a quiescent state, or detecting that the problem is over-constrained are in themselves insufficient as terminating criteria, since new inputs from the environment, in the form of added or deleted variables/constraints might invalidate them.

Algorithm 1 send_nogood(I)

```
Let N be a nogood derived from I
Send N to A
Delete I
```

Algorithm 2 process remove-nogood message

```
Let C be the constraint referenced
for Each received nogood N do
    if N is in the remove-nogood message then
        Delete N from nogoods
        delete N from the remove-nogood message
if counter ≠ 0 then
    Add the remove-nogood message to removed-constraints
receive remove constraint(C)
```

3.1. Algorithm. The core of SSBDO is very simple. First the agent reads any messages it has received from other agents and updates its knowledge. If it has received a proposed solution from another agent that is inconstant, it responds with a nogood message. Second it chooses an assignment for all variables for which it has write privileges. Third the agent sends a message to each of its neighbours, informing them of any change to its proposals. Finally the agent waits until it receives new messages.

All agents continue in this fashion until all their proposals are consistent. When this happens all agents will no longer send any new messages, as their proposed solutions don't change. If deployed in a static environment, termination can be detected by taking a consistent global snapshot [4]. Otherwise they continue to wait until they are informed that the environment has changed, or they are requested to terminate.

Because there is no ordering defined over the agents, this algorithm can very easily adapt to changes in the problem, such as adding or removing constraints. It also degrades gracefully when agents fail, making the

Algorithm 3 process remove-constraint message

```
Let C be the removed constraint
for Each neighbour A do
    for Each nogood N sent to A do
        Let obsolete = {}
        if N contains C as part of its justification then
            Add N to obsolete
            Delete N from sent-nogoods
    if |obsolete| > 0 then
        Let M be a new remove-constraint message with C and nogoods
        Send M to A
for Each received nogood N do
    if N contains C as part of its justification then
        Mark N as obsolete
```

overall system fault tolerant.

In order to generalize SBDO to support SDCOP, agents have been adapted to maintain more than one proposed solution at a time. This allows the algorithm to find many solutions in one execution. Changing the agents **view** from a single proposal to a set of proposals requires changes to the way the agents **view** is created as well as how proposals are sent to other agents. After these changes the information an agent γ stores is:

- **support** The agent that γ is using as the basis for almost all decisions it makes. The **support** 's beliefs about the world (its **view**) are considered to be facts.
- **view** This is a set of proposals consisting of the proposals received from **support** with an assignment to γ's variables appended. This represents the γ's current beliefs about the world, or its world view.
- **recv(A)** This is a mapping from an agent α to the last set of proposals received from α. This stores the other agents most recent arguments.
- **nogoods** This is an unbounded multi-set of all current nogoods received. It contains pairs ⟨sender, nogood⟩.
- **sent(A)** This is a mapping from an agent α to the last set of proposals sent to α. This stores the arguments most recently sent to the agents neighbours.
- **sent-nogoods** This is an unbounded set of all nogoods sent by γ. It contains pairs ⟨destination, nogood⟩.
- **removed-constraints** An unbounded set of known obsolete nogoods. This stores references to all the nogoods that are known to be obsolete, but have not yet been deleted.
- **constraints** A set of all constraints γ knows. It must include all of an agent's local constraints and all constraints this agent shares with other agents.

As with SBDO each agent must first update its **view** based on its current **support** , as new information may have made its current **view** obsolete. The approach used in SBDO is to choose the neighbour that has sent the best proposal as this agents **support** , which clearly does not apply to SSBDO. Instead the agent that has sent the largest number of non-dominated proposals is chosen as this agents **support** . Specifically, all proposals this agent knows of, i.e. those it has received from its neighbours and those it has generated as its current **view** , are considered. Out of those proposals the set of non-dominated proposals is computed and they are partitioned based on their source. If the source of the largest partition is this agents **view** , then this agents **support** does not change. Otherwise this agent changes its **support** to the agent which is the source of the largest partition. If there is a tie for the largest partition, it is broken by considering the following criteria, in lexicographical order:

1. Largest number of proposals received from each source.
2. Largest total length of received proposals from each source.
3. Consistent random choice. i.e. if the set of proposals A is preferred over the set of proposals B, A will

Algorithm 4 main()

```
while Not Terminated do
    for all received nogoods N do
        if this nogood is obsolete then
            decrement counter on the removed-constraint message
            if counter = zero then
                delete constraint-removed message
        else
            Add N to nogoods
            for all neighbours A do
                if There is no valid assignment to myself wrt recv(A) then
                    send_nogood(A)
    for all received environment messages do
        Process message
    for All received sets of proposals S_i do
        Let A be the agent who sent I
        Set recv(A) to I
        for Each proposal I in S_i do
            if There is no valid assignment to myself wrt I then
                send_nogood(A)
    Set view to the non-dominated sub-set of all consistent extensions to all received proposals
    for All neighbours A do
        Set proposed_proposals to an empty set
        for Each proposal I in view do
            if I is part of a cycle then
                if I is dominated by an proposal in recv(A) then
                    Postpone this proposal
                else
                    Add I to proposed_proposals
            else
                Set preferred such that it meets the criteria
                Set I' to a tail of I, such that the length of I is min(max_length, preferred)
                add I' to proposed_proposals
        if proposed_proposals ≠ sent(A) then
            Set sent(A) to proposed_proposals
            Send proposed_proposals to A
    Wait until at least one message has been received
```

always be preferred over B[1].

To illustrate this procedure, consider an agent α which has received the following proposals from it's neighbour β:

- $\langle\langle\epsilon, \{\langle e, 0\rangle\}, (3, 0)\rangle, \langle\beta, \{\langle b, 1\rangle\}, (0, 3)\rangle\rangle, \{\}\rangle$ with a total utility of (3, 3).
- $\langle\langle\gamma, \{\langle c, 0\rangle\}, (2, 1)\rangle, \langle\epsilon, \{\langle e, 1\rangle\}, (0, 3)\rangle, \langle\beta, \{\langle b, 1\rangle\}, (0, 3)\rangle\rangle, \{\}\rangle$ with a total utility of (2,7).

α has also received the following proposals from it's neighbour γ:

- $\langle\langle\epsilon, \{\langle e, 0\rangle\}, (3, 0)\rangle, \langle\gamma, \{\langle c, 0\rangle\}, (2, 1)\rangle\rangle, \{\}\rangle$ with a total utility of (5,1).

Finally, α's **view** consists of the following proposals:

- $\langle\langle\gamma, \{\langle c, 1\rangle\}, (1, 2)\rangle, \langle\alpha, \{\langle a, 1\rangle\}, (0, 3)\rangle\rangle, \{\}\rangle$ with a total utility of (1,5).

[1] Hash functions can provide a suitable comparison.

- $\langle\langle\langle\gamma, \{\langle c, 1\rangle\}, (1, 2)\rangle, \langle\alpha, \{\langle a, 0\rangle\}, (3, 0)\rangle\rangle, \{\}\rangle$ with a total utility of (4,2).

Given this information, the set of non-dominated proposals are:

- $\langle\langle\langle\epsilon, \{\langle e, 0\rangle\}, (3, 0)\rangle, \langle\beta, \{\langle b, 1\rangle\}, (0, 3)\rangle\rangle, \{\}\rangle$ with a total utility of (3, 3).
- $\langle\langle\langle\gamma, \{\langle c, 0\rangle\}, (2, 1)\rangle, \langle\epsilon, \{\langle e, 1\rangle\}, (0, 3)\rangle, \langle\beta, \{\langle b, 1\rangle\}, (0, 3)\rangle\rangle, \{\}\rangle$ with a total utility of (2,7).
- $\langle\langle\langle\epsilon, \{\langle e, 0\rangle\}, (3, 0)\rangle, \langle\gamma, \{\langle c, 0\rangle\}, (2, 1)\rangle\rangle, \{\}\rangle$ with a total utility of (5,1).
- $\langle\langle\langle\gamma, \{\langle c, 1\rangle\}, (1, 2)\rangle, \langle\alpha, \{\langle a, 0\rangle\}, (3, 0)\rangle\rangle, \{\}\rangle$ with a total utility of (4,2).

One of the non-dominated proposals originate from α, two originate from β and one from γ. This will cause α to change its **support** to β. If there was a tie between α and β, then α would win, as it has three total proposals compared to β's two. In the case of a tie between β and γ, β would win, as the total length of its proposals is five compared to γ's four.

In the case of SBDO, where there is at most one proposal from each source, there is normally only one proposal in the non-dominated set. It is possible for two (or more) proposals with equal utility to form the non-dominated set. When this happens the tie breaking procedure is equivalent in SSBDO and in SBDO.

If this agents **support** has changed, then it must re-compute its **view** . To generate its **view** , this agent extends the proposals it has received from its **support** . For each proposal it has received from its support, this agent computes the set of non-dominated proposals which can be generated by extending the received proposal with an assignment to this agent. If the agent changes the value of a variable which already has a value assigned to it in this proposal (i.e. more than one agent has write privileges), it must trim the proposal such that there are not two different values assigned to the same variable. The resulting reduction in the total utility of the proposal ensures that variables which have already been assigned by another agent will only be changed if there is significant benefit in doing so.

The procedure in SBDO for updating an agent's neighbours assumes that one proposal has been sent to and received from each neighbour. It must be generalized for sending sets of proposals, taking care to ensure the properties required for the proof of termination and completeness still hold. These are: postponing of proposals that are involved in a cycle, sending a new proposal if this agent is in conflict with the destination agent and not sending a new message if the content has not changed since the last message.

As with SBDO, each proposal is treated individually, then the proposals that will be sent to this neighbour are grouped and sent in one message. Cycle elimination is the same as in SBDO. If there are two consecutive assignments in the received proposal which were generated by this agent and the destination agent respectively, then this proposal is part of a cycle. If the proposal is dominated by one of the proposals previously sent to the destination agent then it must not be sent at this time. Instead the proposal which dominates it should be resent. If the proposal dominates all of the previously sent proposals, then the entire proposal should be sent. Otherwise, when neither proposal dominates the other, both should be sent. If the proposal is not part of a cycle the next consideration is how much of the proposal to send to the destination agent. The proposal must be long enough to meet the following criteria:

1. If one of the proposals previously sent to the destination agent is a sub-proposal of this proposal, then this proposal is an update. In which case the length of the newly sent proposal should be the length of the previously sent proposal +1.
2. It must contain enough assignments to evaluate shared objectives/constraints. Specifically, if there exists a constraint/objective involving at least the destination agent B and another agent C in this agents' view, then the proposal must contain the assignment to C.
3. If the assignment to A is not consistent with any proposal received from B, then A should send a counter-proposal that is more preferred than the conflicting proposal.
4. The proposal should be equal to or longer than the shortest proposal previously sent to B.

It is not always possible to send a proposal of the desired length, as the length of the sent proposal is limited by the length of the proposal in **view** . Once the correct length of the proposal has been decided a new proposal is created. Once all proposals in **view** have been considered, the new proposals are checked against the proposals previously sent to this agent. If any of them have changed a proposal message is sent to the destination agent containing all of the proposals.

To illustrate this procedure, again consider an agent α. α's **view** is currently:

- $$\langle\langle\langle\beta,\{\langle b,0\rangle\},(2,0)\rangle,\langle\epsilon,\{\langle e,1\rangle\},(0,3)\rangle,\langle\gamma,\{\langle c,0\rangle\},(2,1)\rangle,\langle\alpha,\{\langle a,1\rangle\},(0,3)\rangle\rangle,\{\}\rangle$$
  with a total utility of (4,7).
- $$\langle\langle\langle\epsilon,\{\langle e,1\rangle\},(0,3)\rangle,\langle\gamma,\{\langle c,1\rangle\},(1,2)\rangle,\langle\alpha,\{\langle a,1\rangle\},(0,3)\rangle\rangle,\{\}\rangle$$
  with a total utility of (1,8).
- $$\langle\langle\langle\gamma,\{\langle c,0\rangle\},(2,1)\rangle,\langle\alpha,\{\langle a,0\rangle\},(3,0)\rangle\rangle,\{\}\rangle$$
  with a total utility of (5, 1).

Further, α has previously sent the following proposals to β:

- $$\langle\langle\langle\beta,\{\langle b,1\rangle\},(0,3)\rangle,\langle\epsilon,\{\langle e,0\rangle\},(3,0)\rangle,\langle\gamma,\{\langle c,1\rangle\},(1,2)\rangle,\langle\alpha,\{\langle a,0\rangle\},(3,0)\rangle\rangle,\{\}\rangle$$
  with a total utility of (7,5).
- $$\langle\langle\langle\gamma,\{\langle c,0\rangle\},(2,1)\rangle,\langle\alpha,\{\langle a,0\rangle\},(3,0)\rangle\rangle,\{\}\rangle$$
  with a total utility of (5,1).

Finally, α has received (along with other proposals) the following proposal from β:

$$\langle\langle\langle\epsilon,\{\langle e,0\rangle\},(3,0)\rangle,\langle\gamma,\{\langle c,1\rangle\},(1,2)\rangle,\langle\alpha,\{\langle a,0\rangle\},(3,0)\rangle,\langle\beta,\{\langle b,1\rangle\},(0,3)\rangle,\rangle,\{\}\rangle$$

with a total utility of (7, 5).

We can see that the proposal received from β contains an assignment to α then an assignment to β, as such they are part of a cycle. The first proposal in α's view has assignments to the same variables in the same order as the proposal received from β, so it might need to be postponed. Neither proposal dominates the other, so both proposals are sent to β. The second proposal in α's view is an update to the second proposal α sent previously, so a slightly longer subset of that proposal is sent to β. Finally, the third proposal is new, so α attempts to send a length three version of it, but as it is only length two, the entire proposal is sent. Therefore the new proposal message α sends to β is:

- $$\langle\langle\langle\beta,\{\langle b,1\rangle\},(0,3)\rangle,\langle\epsilon,\{\langle e,0\rangle\},(3,0)\rangle,\langle\gamma,\{\langle c,1\rangle\},(1,2)\rangle,\langle\alpha,\{\langle a,0\rangle\},(3,0)\rangle\rangle,\{\}\rangle$$
  with a total utility of (7,5).
- $$\langle\langle\langle\beta,\{\langle b,0\rangle\},(2,0)\rangle,\langle\epsilon,\{\langle e,1\rangle\},(0,3)\rangle,\langle\gamma,\{\langle c,0\rangle\},(2,1)\rangle,\langle\alpha,\{\langle a,1\rangle\},(0,3)\rangle\rangle,\{\}\rangle$$
  with a total utility of (4,7).
- $$\langle\langle\langle\epsilon,\{\langle e,1\rangle\},(0,3)\rangle,\langle\gamma,\{\langle c,1\rangle\},(1,2)\rangle,\langle\alpha,\{\langle a,1\rangle\},(0,3)\rangle\rangle,\{\}\rangle$$
  with a total utility of (1,8).
- $$\langle\langle\langle\gamma,\{\langle c,0\rangle\},(2,1)\rangle,\langle\alpha,\{\langle a,0\rangle\},(3,0)\rangle\rangle,\{\}\rangle$$
  with a total utility of (5, 1).

The procedure for determining the length of each proposal to send is the same as in SBDO. However there are some changes due to it acting on two sets of proposals, rather than two proposals. First, for each new proposal, it is not clear which old proposal it should be compared with. In this case it is compared with all of them and the longest required proposal is sent. Further, when a proposal is postponed, the old version of the proposal must be sent. Otherwise the proposal will be lost when the destination agent updates its received proposals. Finally, an updated proposal message must be sent when any of the individual proposals change.

The changes made to the procedure for sending updates to an agents neighbours invalidate the proof of termination for SBDO. Here we present a generalization of the proof of termination for SBODsr. This is based on the proof of termination for SBDS [9].

LEMMA 3.4. *If no new nogoods are generated, then eventually the utility of view will become stable for each agent.*

Proof. Let $W_i \subseteq \mathcal{X}$ be the set of agents whose view dominates $i \in \mathcal{Z}$ of the possible solutions to the problem. An agent's view v dominates a solution s iff there exists a proposal $p \in v$ such that the utility of p is greater than or equal to the utility of s (not less than or incomparable). We will prove that any decrease in $|W_i|$ must be preceded by an increase in $|W_j|$, where $j < i$.

First, we note that an agent will never willingly reduce the number of solutions its view dominates, as per the proposal ordering and the requirements of **update_view()** . So, in the usual case, $|W_i|$ will be monotonically increasing, for all i. However, in limited circumstances an agent may receive a weaker proposal from its support, and so the number of solutions its view dominates could be forced to decrease. Such events are rare, but they can occur whenever a cycle of supporting agents is formed. Let us assume that some agent v receives a worse proposal I from its current support, and so v is forced to choose a view which dominates less solutions. Let i number of solutions v's old view dominates, and j be the number of solutions v's new view dominates, respectively. The new, worse view for v will obviously decrease each $|W_k|$, where $j < k \leq i$.

However, for v to have received the worse proposal I, some agent w must have formed a cycle by changing its support. Note that w will only have selected a new support if it could increase the number of solutions its view dominates, as per the proposal ordering and the requirements of **update_view()** . Also note that the newly-formed cycle cannot have a total utility of more than j, else there would have been no reason to reduce the number of solutions v's view dominates. Therefore, the number of solutions w's new view dominates must then be less than or equal to j, but is certainly more than it's old view.

So, if an agent v is forced to reduce the number of solutions its view dominates, then there must be some preceding agent w which increased the number of solutions its view dominates. Further, w's new view is guaranteed to dominate no more solutions than v's new view. Therefore, the term $|W_1|.|W_2|.|W_3|\ldots$ must increase lexicographically over time. As the term is bounded above, we can conclude that the utility of view must eventually become stable for each agent. □

THEOREM 3.5. *SSBDO is an instance of SDCOP with the following limitations:*

- *The domain of variables ($\bigcup \mathcal{D}$) must be finite.*
- *The quality of a solution must be monotonically non-decreasing as the solution is extended, i.e. $\forall a, b \in V, a \otimes b \preceq a$.*

Proof. The elements $\mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{V}$ and $\mathcal{C}$ are defined the same in SSBDO and SDCOP.

SSBDO does not support problems where the domain of a variable may be finite. This case is excluded by the first restriction on an SDCOP.

Further, the SSBDO algorithm is only sound when aggregating two semiring values does not produce a less preferred semiring value. This case is excluded by the second restriction on an SDCOP. □

3.2. Example. As an example of how SBDOsr works consider the following simple graph colouring problem. Note that while SDCOP requires the constraints to be encoded as functions returning semiring values, for ease of understanding we will discuss the problem in terms of hard and valued constraints. There are three agents, Γ, Δ and Θ, each of which have write privileges for one variable, γ, δ and θ respectively. Each variable can take one of three 'colours', 0, 1 and 2. Neighbouring variables share a valued constraint of colour difference, maximize the difference between the values assigned to each agent. Each variable also has a unary valued constraint of colour affinity, minimize the distance between its value and an ideal value. The ideal value is 0, 1 and 1 for γ, δ and θ respectively.

When the algorithm starts each agent has received no other proposals to build upon. So all of them choose an assignment based on the colour affinity constraint. Γ adopts the proposal $\langle\langle\langle\Gamma, \{\langle\gamma, 0\rangle\}, (0, 2)\rangle\rangle, \{\}\rangle$, Δ adopts the proposal $\langle\langle\langle\Delta, \{\langle\delta, 1\rangle\}, (0, 2)\rangle\rangle, \{\}\rangle$ and Θ adopts the proposal $\langle\langle\langle\Theta, \{\langle\theta, 1\rangle\}, (0, 2)\rangle\rangle, \{\}\rangle$. Each agent then sends their choice to each of the other agents.

Now we concentrate only on Θ. The reasoning for the other agents is similar. None of the proposals Θ has dominate any of the others, and each of itself, Δ and Γ have supplied one non-dominated proposal, and all proposals are of length 1. As such Θ randomly chooses Γ as its **support** and extends each of Γ's proposals to find the following non-dominated proposals:

- $\langle\langle\langle\Gamma, \{\langle\gamma, 0\rangle\}, (0, 2)\rangle, \langle\Theta, \{\langle\theta, 2\rangle\}, (0, 0)\rangle\rangle, \{\langle(\gamma, \theta), (2, 0)\rangle\}\rangle$ with a utility of $(2, 2)$
- $\langle\langle\langle\Gamma, \{\langle\gamma, 0\rangle\}, (0, 2)\rangle, \langle\Theta, \{\langle\theta, 1\rangle\}, (0, 1)\rangle\rangle, \{\langle(\gamma, \theta), (1, 0)\rangle\}\rangle$ with a utility of $(1, 3)$

As the first proposal found is new, only the front of it, $\langle\langle\langle\Theta, \{\langle\theta, 2\rangle\}, (0, 0)\rangle\rangle, \{\}\rangle$, is sent to the other agents, while the entirety of the second proposal is sent.

In Θ's next cycle it receives the following proposals from Γ and Δ:

- $\langle\langle\langle\Delta, \{\langle\delta, 1\rangle\}, (0, 1)\rangle, \langle\Gamma, \{\langle\gamma, 0\rangle\}, (0, 2)\rangle\rangle, \{\langle(\gamma, \delta), (1, 0)\rangle\}\rangle$ with a utility of $(1, 3)$
- $\langle\langle\langle\Theta, \{\langle\theta, 1\rangle\}, (0, 2)\rangle, \langle\Delta, \{\langle\delta, 1\rangle\}, (0, 2)\rangle\rangle, \{\langle(\delta, \theta), (0, 0)\rangle\}\rangle$ with a utility of $(0, 4)$
- $\langle\langle\langle\Delta, \{\langle\delta, 0\rangle\}, (0, 0)\rangle\rangle, \{\}\rangle$ with a utility of $(0, 0)$, (because it is a new assignment it starts at length one)
- $\langle\langle\langle\Delta, \{\langle\delta, 2\rangle\}, (0, 0)\rangle\rangle, \{\}\rangle$ with a utility of $(0, 0)$

Θ then decides to retain Γ as its support because the largest number of non-dominated proposals are from Θ's **view** (Δ and Θ supply one each). Next Θ extends the proposal received from Γ to get:

- $\langle\langle\langle\Delta, \{\langle\delta, 1\rangle\}, (0, 1)\rangle, \langle\Gamma, \{\langle\gamma, 0\rangle\}, (0, 2)\rangle, \langle\Theta, \{\langle\theta, 1\rangle\}, (0, 1)\rangle\rangle,$
  $\{\langle(\gamma, \delta), (1, 0)\rangle, \langle(\gamma, \theta), (1, 0)\rangle, \langle(\delta, \theta), (0, 0)\rangle\}\rangle$ with a utility of $(2, 4)$
- $\langle\langle\langle\Delta, \{\langle\delta, 1\rangle\}, (0, 1)\rangle, \langle\Gamma, \{\langle\gamma, 0\rangle\}, (0, 2)\rangle, \langle\Theta, \{\langle\theta, 2\rangle\}, (0, 0)\rangle\rangle,$
  $\{\langle(\gamma, \delta), (1, 0)\rangle, \langle(\gamma, \theta), (2, 0)\rangle, \langle(\delta, \theta), (1, 0)\rangle\}\rangle$ with a utility of $(4, 3)$

Again Θ then informs Δ and Γ of the solutions it has chosen.

As these two proposals represent the optimal solutions for this problem execution continues for one more cycle, as Δ and Γ accept them as the optimal solutions.

4. Results. We ran a set of experiments to evaluate SBDOsr. To do so, we implemented SBDOsr in C++[2] and ran tests on a set of graph colouring problems. The tests were run on an Intel Xeon X3450 CPU with 8GB of RAM.

At this time, there is only one other published algorithm that is capable of solving problems with many objective functions, B-MUMS [7]. We do not compare SBDOsr with B-MUMS as B-MUMS does not support hard constraints, as are used in our experiments, and B-MUMS only returns one solution[3].

In our test problem, there is a one to one mapping from agents to variables and each variable can take a value from the domain $\{0, 1, 2, 3, 4\}$. Each variable is identified by a unique integer. There are three constraints between each pair of neighbouring variables. The first is a hard constraint that neighbouring variables must not have the same value. The second is a valued constraint to maximize the distance between the two values, given that the values wrap around i.e. the distance between 0 and 4 is 1. The third is a valued constraint where the variable with the higher identifier should be assigned a larger value. Finally, every variable has a unary valued constraint to minimize the distance between the variables value and an ideal value, being the variables identifier modulo 5.

For our tests, we varied the number of variables in each problem and the number of constraints. The parameter for the graph connectedness varies linearly between 0, where the constraints form a spanning tree over the variables, and 1, which is a fully connected graph. We used a number of variables from the set {4, 5, 6, 7, 8, 9, 10, 15, 20, 25}, a number of constraints from the set {0.0, 0.1, 0.2, 0.3. 0.4}, and randomly generated five problems for each pair of parameters. Each problem was solved five times and the performance averaged to give the results presented here. Individual runs were terminated after half an hour of wall clock time.

We present the performance of SBDOsr based on five metrics, and the standard deviation for each metric:

1. (Terminate) Time for the algorithm to terminate.

[2]Source code available from http://www.geeksinthegong.net/svn/sbdo/trunk/.

[3]We acknowledge that B-MUMS could easily be modified to return many solutions, as it finds them during processing.

Table 4.1: Performance of SBDOsr. See text for description of metrics.

variables	Terminate (s)	Aggregate (s)	number of solutions	solution quality	Proportion
4	0.15 (0.13)	0.01 (0.00)	9.66 (5.20)	0.13 (0.13)	0.46 (0.26)
5	0.20 (0.12)	0.01 (0.01)	11.38 (6.33)	0.22 (0.32)	0.49 (0.26)
6	0.35 (0.33)	0.02 (0.01)	13.58 (6.11)	0.25 (0.29)	0.52 (0.24)
7	0.76 (0.83)	0.04 (0.04)	17.54 (10.18)	0.23 (0.27)	0.47 (0.24)
8	1.87 (3.24)	0.05 (0.04)	21.87 (14.65)	0.35 (0.34)	0.38 (0.24)
9	3.68 (5.54)	0.15 (0.24)	31.63 (24.70)	0.38 (0.33)	0.35 (0.24)
10	6.02 (8.33)	0.16 (0.20)	30.27 (17.72)	0.50 (0.33)	0.25 (0.21)
15	125.10 (233.45)	5.43 (20.17)	52.47 (66.61)	-	-
20	287.72 (259.97)	64.67 (184.73)	76.44 (189.77)	-	-
25	433.88 (548.73)	269.87 (546.00)	50.00 (158.81)	-	-

2. (Aggregate) Time to aggregate the partial solutions.
3. (number of solutions) Total number of solutions found.
4. (solution quality) The average of the minimum euclidean distance from the utility of each non-optimal solution to the utility of an optimal solution. Formally:

$$\text{quality} = \frac{\sum_{n \in N} \min\left(f\left(n, o_1\right), \ldots, f\left(n, o_x\right)\right)}{|N|}$$

   where $f(x, y)$ is the euclidean distance between x and y, S is the set of the utilities of the solutions found by SBDOsr, O is the set of utilities of the optimal solutions and $N = S/O$.
5. (Proportion) The proportion of optimal solutions found. Formally:

$$\text{proportion} = \frac{|O \cap S|}{|O|}$$

Note that the set of optimal solutions must be known to compute metrics four and five. We used exhaustive search to find all the optimal solutions for problems with up to ten variables, it proved to be infeasable to solve bigger problems using exhaustive search.

Because each agent only has a local view of the problem a post-processing step is required to combine all the partial solutions into complete solutions. Whether this step is required depends on the problem being solved, decision support applications will require complete solutions, but some things like autonomous robots may be able to function using only local knowledge. It also depends on how many solutions are desired, for these experiments we extracted all pareto-optimal solutions found by SBDOsr. Because of this, we have presented the time required for the algorithm to terminate and the time to aggregate the partial solutions into global solutions separately.

The performance of SBDOsr is shown in Table 1. The number of constraints in the problem had very little effect on the performance, so we have not reported those results here. As expected, the number of variables in the problem has a large impact on the performance. Both the time required for the algorithm to terminate and the number of solutions found when the algorithm does terminate increases exponentially as the number of variables increases. Furthermore, the time required to aggregate all solutions is dependant on the number of solutions, so aggregation time also rises exponentially as the number of solutions increases.

The standard deviations show that the performance of SBDOsr is highly unstable, often the standard deviation is greater than the mean. This is due to the highly non-deterministic nature of the SBDOsr algorithm. The order in which agents preform actions introduces a search bias. This bias determines which solutions are found and how much effort is required to terminate.

The proportion of optimal solutions that SBDOsr finds drops off as the number of variables increases. While the average distance from each non-optimal, found solution to an optimal solution remains constant, representing a change to the assignment to one variable by about one unit. This shows that while the number of

points discovered on the actual pareto-front drops off as the number of variables increases, the solutions found remain close to the actual pareto-front.

In the process of solving these problems, SBDOsr generates a large number of nogoods. As our implementation of SBDOsr only has relatively simple code for searching and checking nogoods, this represents a significant performance bottleneck and contributes to the time required for the larger problems.

5. Conclusion. We have presented a modification of SBDO to support problems with multiple objectives, or in general, any DCOP problem where there does not exist a total pre-order over the set of solutions. In order to represent these problems, we propose a new definition of a DCOP using an idempotent semiring to measure the cost/utility of a solution. The SBDO algorithm was then modified to use this new definition. To solve problems of this form, each agent maintains multiple candidate solutions simultaneously. The partial solutions maintained by each agent can then be combined into a set of complete solutions.

Empirical evaluation shows that the algorithm finds a good approximation of the pareto-frontier, however the post-processing step to combine each agent's partial solutions into complete solutions requires significant computational effort, and is currently done centrally. While this is a weakness of this approach, in situations such as autonomous robot control where complete solutions are not required for the agents to act, such a weakness is acceptable.

REFERENCES

[1] U. Bertele and F. Brioschi, *Nonserial Dynamic Programming*, Academic Press, 1972.
[2] G. Billiau, C. F. Chang, and A. Ghose, *SBDO: A New Robust Approach to Dynamic Distributed Constraint Optimisation*, in Principles and Practice of Multi-Agent Systems, 2010.
[3] S. Bistarelli, U. Montanari, and F. Rossi, *Semiring-based constraint satisfaction and optimization*, J. ACM, 44 (1997), pp. 201–236.
[4] K. M. Chandy and L. Lamport, *Distributed Snapshots: Determining global states of distributed systems*, ACM Transactions on Computer Systems, 3 (1985), pp. 63–75.
[5] R. Dechter, *Bucket elimination: A unifying framework for reasoning*, in Articial Intelligence, vol. 113, 1999, pp. 41–85.
[6] R. Dechter and I. Rish, *Mini-buckets: A general scheme for bounded inference*, in Journal of the ACM, vol. 50, Mar 2003, pp. 107–153.
[7] F. M. D. Fave, R. Stranders, A. Rogers, and N. R. Jennings, *Bounded Decentralised Coordination over Multiple Objectives*, in AAMAS, 2011, pp. 371–378.
[8] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer, and A. Meisels, *Asymmetric distributed constraint optimization problems*, J. Artif. Intell. Res. (JAIR), 47 (2013), pp. 613–647.
[9] P. Harvey, *Solving Very Large Distributed Constraint Satisfaction Problems*, PhD thesis, University of Wollongong, 2010.
[10] P. Harvey, C. F. Chang, and A. Ghose, *Support-based distributed search: a new approach for multiagent constraint processing*, in AAMAS '06, ACM, 2006, pp. 377–383.
[11] J. Larrosa, *On arc and node consistency in weighted CSP*, in proc. of AAAI'02, 2002, pp. 48–53.
[12] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, *ADOPT: asynchronous distributed constraint optimization with quality guarantees*, Artificial Intelligence, 161 (2005), pp. 149–180.
[13] J. P. Pearce and M. Tambe, *Quality Guarantees on k-Optimal Solutions for Distributed Constraint Optimization Problems*, in IJCAI, M. M. Veloso, ed., 2007, pp. 1446–1451.
[14] A. Petcu, *A class of algorithms for Distributed Constraint Optimistation*, PhD thesis, École Polytechnique Fédérale de Lausanne, Oct 2007.
[15] A. Petcu and B. Faltings, *DPOP: A Scalable Method for Multiagent Constraint Optimization*, in IJCAI 05, Aug 2005, pp. 266–271.
[16] E. Rollon, *Multi-objective Optimization in Graphical Models*, PhD thesis, Universitat Politècnica de Catalunya, 2008.
[17] A. Rosenfeld, *An Introduction to Algebraic Structures*, Holden-Day, 1968.
[18] F. Rossi, P. van Beek, and T. Walsh, eds., *Handbook of Constraint Programming*, Elsevier, 2006.
[19] T. Schiex and G. Verfaillie, *Nogood Recording for Static and Dynamic Constraint Satisfaction Problem*, International Journal of Artifical Intelligence Tools, 3 (1994), pp. 187–207.
[20] M. C. Silaghi and M. Yokoo, *Nogood based asynchronous distributed optimization (ADOPT-ng)*, in AAMAS '06, ACM, 2006, pp. 1389–1396.

Edited by: Yang Xu
Received: April 15, 2015
Accepted: January 14, 2016