Testing Computational Dialectic

Simon Wells and Chris Reed

May 22, 2005

Abstract

Computational dialectics are a powerful way to structure the communicative acts that are expressed during the process of argumentation. Many such systems have been proposed but there has been no consensus over what constitutes a good system, or on what basis such a consensus could be built. This paper introduces a testbed for computational dialectics that enables disparate systems to be easily implemented and investigated, and for further comparisons between those systems to be made.

1 Introduction

This paper introduces a scenario and knowledge domain for computational dialectic testing, named GC_0, and an implementation of that domain named Sweetwater. The purpose of GC_0 and Sweetwater are to provide tools for automating the comparison and evaluation of computational dialectics.

Formal dialectics are two-player, turn-taking games. The players take turns to make moves in accordance with a set of rules. Moves are locutionary acts for which the rules prescribe, prohibit or permit the legality at any given point. Formal dialectics were introduced in [8] as a practical means to formally structure the interactions between the participants of a dialogue. The aim was to examine and clarify the conditions and situations in which logical fallacies occur. Computational implementations of dialectics have found application in many areas of computing research, including as interface components in computer-based learning systems [16], in AI and law, and as a means to structure the communications between software agents [17]. Many systems of dialectics have been proposed including, H [8], DC [10], DL3 [7], PPD [26], R [21] as well as related systems such as the Toulmin Dialogue Game [5], the case study games [11], the eightfold model [12] and variations on existing dialogue games [1]. Dialogue games and formal dialectic systems have also been proposed as means to structure argumentative dialogue between agents in a MAS [18].

The abundance of dialectics mean it is not straightforward to select an individual formulation of rules to implement, nor is it simple to identify which systems are best applied to a given situation. It is necessary that this situation is clarified. For dialectics to be accepted and implemented by developers it is necessary to be able to show how these systems differ from each other, how they perform, and to which communicative situations a given system is best applied. Approaches to this problem involve comparison of the rules which constitute a system of dialectic. Various aspects of this approach have been examined in [13] which looks at measurable attributes of dialogue systems and [27] which examines the unified specification of such systems. Another approach uses comparison and analysis of the dialogues produced by a system and is the basis for the work presented here. Together these approaches enable a comprehensive overview of computational dialectics.

2 Finding a *drosophila*

Drosophila Melanogaster is a type of fruit fly used in the biological sciences as a testbed against which to measure progress [3]. Herbert Simon [22] and later John McCarthy [14] refer to the use of Chess as a *'drosophila'* for AI. In a similar vein McCarthy proposes the missionaries and cannibals puzzle as a drosophila for problems in logical AI [15]. The notion is that certain classes of problem can be used as a basis for measuring, testing and comparing the progress made in a given field. This concept, the use of puzzles, problems and games, can be used as the basis for a testbed for computational dialectics. Precedent for this approach is found in [23] and [24] which posits the application of benchmark problems to research in defeasible reasoning and the interpolation of said results as a pragmatic research method.

Our drosophila is the four-colour problem[6], a subproblem in the graph-colouring domain. This asks whether any map can be coloured using only four colours such that no two neighbouring regions share the same colour. In graph-theoretic terms each region can be considered to be a vertice in a graph. It may then be asked whether, given a connected planar graph, only four colours are required to assign each vertice a colour such that no neighbouring vertices share the same colour. For non-planar graphs it may be asked whether the graph can be coloured using n colours with the same stipulations.

The selection of this problem relies on the need to satisfy several criteria for a testbed for dialectics:

1. The problem should provide a social context for initiating argumentative dialogue incorporating goal-directed behavior, problem-solving and reasoning abilities.
2. The knowledge domain extracted from the problem must provide a basis for argumentative discourse between agents. Argument artifacts should be rooted in the state of the mas so that the basis for an argument can be examined and verified.
3. There must be scope for rich agent interactions and behavior, but arguments and dialogues must be sufficiently simple to comprehend and analyse. Human comprehension is aided by requiring arguments to be reminiscent of real-world argument structures.
4. The scenario should facilitate structured expansion. The benefits of this are twofold, firstly it enables the basic assumptions of the initial scenario to be developed, and secondly it enables the scope of the initial scenario to be widened. The intuition is that no single problem will enable a full analysis of all computational dialectics but that the interpolation of results from a number of problems will give a comprehensive picture of the abilities of a system.

The aim is not to produce a solution to the four-colour problem, but to provide a basis for the automatic production of a body of dialogues according to the rules of computational dialectic systems. The dialogues can then be analysed and used to identify the effects of differing formulations of rules on the resultant dialogues. Essentially this is an experimental apparatus in which every functional aspect of the agents is kept static other than the dialectic and some carefully defined aspects of the agents knowledge which facilitates dynamic behaviour. Differences between the resultant dialogues can then be traced back to the differences in the formulation of rules for the dialectic.

3 Scenario: GC_0

The initial graph-colouring scenario, named GC_0, specifies an initial scenario for testing computational dialectic. GC_0 is presented as a starting point for examining computational dialectic. The elements of the core scenario are presented as follows:

Colours Each agent possesses a colour state attribute. Colour states are selected from a fixed pool. The number of colours is fixed at four; red, yellow, green and blue.

Relationships Each agent maintains relationships with a known set of other agents. Graph-theoretically, if agents are vertices then a relationship is defined as an edge joining two vertices. Where two vertices are joined by exactly one edge those vertices are called *neighbours*. Neighbours are only ever connected directly by one edge although there may be higher order relationships connecting two vertices through other vertices. Relationships are initiated during system startup and are fixed throughout the duration of the MAS.

Agent Knowledge At start-up an agent knows only its own colour and a set of neighbouring agents. An agent must therefore research other agents in the MAS in order to increase its knowledge base. To achieve this the agent must engage in information-seeking dialogue with its neighbours in order to find out the colour properties and relationships of the other agents. Matters are further complicated because knowledge is uncertain. Whilst an agent can only ever be in a single colour state at any given time that colour state is mutable, it changes with time as the agent determines that another colour state is more suitable. The other agents in the system do not necessarily know that a particular agent has changed colour and might attempt to use information that is no longer accurate. As a result the agents must reason with uncertain and dynamic information in order to achieve their goals.

Conflicts Conflicts occur when neighbouring agents are in the same colour state. Conflict is defined in a graph as the state where two vertices connected by one edge are the same colour.

Goals The initial goal of all agents in GC_0 is to resolve all conflicts with their neighbours.

Actions An agent can elect to change its own colour state at will.

Conflict Resolution When a conflict occurs it is necessary to resolve the situation between the *conflicting agents* in accordance with the agents goals. If the agents who are parties in a conflict have available colours states to which they can change and which will not bring them into further conflict with other agents then either agent may change to one of those colour states. These colour states are referred to as *free-colours*. If there are no free-colours the conflicting agents must determine which agent should change colour to resolve the current conflict even though further conflicts result.

Communication Agents can engage in dialogue with their neighbours. The protocols which regulate the communicative acts made during dialogue are formal dialectic systems. Communication is the only means which agents can use to influence the actions of other agents and thereby bring about conflict resolution or to increase their knowledge about other agents relationships and colour states. During a conflict agents use computational dialectics to present arguments to each other with the aim of resolving the conflict. The formal dialectic systems specify the moves that can be made at any given time and the effects of making those moves but not the arguments themselves. Individual formal dialectic systems are expressed using a unified specification format [27].

Arguments Agents use their knowledge of relationships and conflicts to produce arguments. Arguments are expressions of support relations between concepts in the agents knowledge stores. There is a preference ordering over the arguments that an agent can produce which enables an agent to determine which arguments are acceptable [2]. Provided that an agent has sufficient knowledge and that the current state of the system is such that an argument can be produced, then whenever a move requires production of some argument the agent should be able to furnish such an argument from its knowledge store.

4 Implementation: Sweetwater

Sweetwater is a multi-agent implementation of GC_0. It is written in Java and uses the Jackdaw University Development Environment (JUDE). Jackdaw is a lightweight, flexible, industrial-strength agent platform that utilises a modular approach to agent development [9]. The domain specific functionality of sweetwater agents is encapsulated into modules which can be dynamically loaded into Jackdaw agents at runtime. Individual components are implemented that correspond to particular aspects of functionality within a sweetwater agent. These include reasoning, dialectic, argument and knowledge components. Communication between these components is mediated by public interfaces. This approach enables individual agent components to be replaced with new functionality as required.

4.1 Overview

The dialogue manager module does the reasoning for the sweetwater agent. The reasoning algorithm requires the dialogue manager to access the dialectic manager to determine what moves can be made. Formal dialectic moves specify various types of information that can stand as content for the move. This information is cached in the knowledge store which can be accessed directly or mediated by the argument manager. For each move, the dialogue manager accesses the knowledge store or argument manager as required. The knowledge store is used to retrieve concepts that the agent knows. The argument manager is used to construct and retrieve arguments as required by the dialectic system.

4.2 Dialectic Manager

The dialectic manager implements computational dialectics using a unified formal dialectic specification[27] enabling dialectics to be stored externally to the implementation in XML files. This treats individual systems of dialectic as a set of moves. Each move has a set of requirements, expressed in terms of earlier dialogue moves, commitment store contents, dialogue events, and the state of game pieces, and a set of effects, expressed in terms of commitment store updates and dialogue events.This approach enables dialectic systems to be specified externally and separately to the implementation of GC_0. Dialectics which enables a wide range of 'dialogue-types' can thus be implemented. Where a particular form of dialogue proves to be out-with the scope of GC_0 then an enhanced scenario can be implemented.

The unified specification format enables a wide range of formal dialectic systems to be represented efficiently and manipulated in a single software implementation. Further, many variations of existing dialectics can be implemented and executed simply by varying the set of rules held in the specification file. This enables a systematic exploration of computational dialectics to be made and the resulting dialogues compared. The effects of small differences in the rules of a system of dialectic on the resultant dialogue can thus be established. This process will lead to the identification of optimum systems of computational dialectic for a wide range of situations.

4.3 Argument Manager

The argument manager performs two core functions, it constructs arguments using concepts from the knowledge store, and it enables received arguments to be analysed and verified. The core purpose of this component is to supply the argument theoretic content for dialectic moves as required during a dialogue.

The argument manager stores *argument templates*. Argument templates can be characterised as semi-instantiated argumentation schemes [25]. Schemes are traditionally used to capture stereotypical patterns of reasoning and have found application in argument analysis [19] and argument generation [20, 4]. Schemes are concerned with the abstract form of arguments regardless of knowledge domain. Templates are less abstract and are used to specify the form of typical arguments within a given knowledge domain. Templates are used to relate knowledge store concepts to each other to construct arguments. This is achieved by specifying parameters for the range and type of propositions that can stand in each position in the scheme when instantiating an argument. Arguments are conceived in this case as a structure incorporating a set of premises and a conclusion.

The external interface to the argument module enables the reasoning component to retrieve argument-theoretic concepts such as premises, conclusions, warrants as required. Given an argument object the argument module can also be used to retrieve rebutting, refuting and undercutting arguments as required to complete moves suggested by the dialectic system. This functionality is implemented through the

Individual templates are specified externally to the implementation in an xml file to enable new templates to be added rapidly with a minimum of new code. The following xml elements are allowed;

conclusion: ⟨conc type="element_type" name="concept_name"⟩

premise: ⟨prem type="element_type" name="concept_name"⟩

parameter: ⟨param name="concept_name"⟩

where *type* determines how the content is to be retrieved, a *dynamic* type retrieves a concept directly from the knowledge store whilst the *conditional* type checks for a condition to be met in the knowledge store before retrieving the associated concept. The *name* parameter simply specifies the concept that is to be retrieved. The *param* element specifies concepts that the conditional-type elements should check before retrieving a concept.

Example Argument Template

```
⟨arg_template⟩
    ⟨conc type="dynamic" name="conflict_comparison"⟩
    ⟨prem type="dynamic" name="num_conflicts"⟩ my_agent_name ⟨/prem⟩
    ⟨prem type="dynamic" name="num_conflicts"⟩ other_agent_name ⟨/prem⟩
    ⟨prem type="conditional" name="lesser"⟩
        ⟨param name="num_conflicts"⟩my_agent_name⟨/param⟩
        ⟨param name="num_conflicts"⟩other_agent_name⟨/param⟩
    ⟨/prem⟩
⟨/arg_template⟩
```

Commentary This template specifies parameters for an argument of the general form, "you have more conflicts than me because I have n conflicts and you have m conflicts and n is less than m." The exact values for the variables n and m are retrieved from the knowledge store and depend upon the agents knowledge of the current state of the mas. The conditional determines that n is indeed less than m which determines that there is a valid inference.

4.4 Reasoning

The reasoning process is implemented in the dialogue manager module. This module coordinates the functionality of the dialectic manager and the argument manager. These managers produce sets of legal moves and valid arguments. These must be integrated to produce a set of fully-instantiated legal moves, those moves which are legal and have content such that they satisfy the requirements for the move, and consequently may be uttered. Fully-instantiated move selection, selecting the move which should be uttered from the moves that can be uttered is the core of the reasoning process. This is currently achieved through random selection from the set of fully-instantiated moves. A more mature *strategic dialogue manager* is included in the future development path for this module, so that a more structured selection of moves can be made. For the moment random-selection is sufficient.

5 Towards Metrics for Computational Dialectics

Aspects of computational dialectics and their resultant dialogues which might be measured fall into two broad categories, *inspection metrics*, those metrics which can be measured through external examination of the rules of the system, and *process metrics*, those metrics which are most easily measured through application of the system and subsequent examination of the results. Many inspection metrics are identified in [13] which sets out thirteen desirable qualities of argumentative dialogue systems. A range of process metrics have been identified which can be investigated using sweetwater. Three initial metrics for investigation include:

Efficiency There are two measures of efficiency that are pertinent. Firstly there is the efficiency of computational dialectics versus other forms of reasoning and communication, this is not addressed here. The second is the performance of individual systems of dialectic against each other. This can be examined by isolating as many variables external to the dialectic system as possible, then comparing between systems, the average number and size of messages required to satisfy a goal.

Flexibility Through varying the number of argument templates available to a system, which effectively varies the number of different arguments available, examine the ability of a system to satisfy goals whilst reasoning with less complete knowledge.

Stability Given the same general inputs, does the system converge to the same set of results?

6 Conclusions and Future Work

GC_0 and Sweetwater are a good way to approach the comparative evaluation of computational dialectics. It provides a social context for initiating argumentative dialogue, a simple, structured body of knowledge, and a means to extract argument-theoretic structures from that knowledge.

Future work involves a systematic exploration of the space of formal dialectic systems, the production and analysis of a body of example dialogue for each system, identification of further process metrics, and the exploration of enhanced scenarios that build upon GC_0.

References

[1] L. Amgoud, N. Maudet, and S. Parsons. Modelling dialogues using argumentation. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, 2000.

[2] L. Amgoud and S. Parsons. Agent dialogues with conflicting preferences. In *ATAL 2001*, pages 190–205, 2004.

[3] M. Ashburner. *Drosophila: A Laboratory Handbook*. Cold Spring Harbor Laboratory Press, 1989.

[4] K. Atkinson, T. Bench-Capon, and P. McBurney. Justifying practical reasoning. In I. Rahwan, P. Moraitis, and C. Reed, editors, *First International Workshop on Argumentation in Multi-Agent Systems*, 2004.

[5] T. J. M. Bench-Capon. Specification and implementation of toulmin dialogue game. In *Proceedings of JURIX 98*, pages 5–20, 2001.

[6] A. Cayley. Open problem. *Proceedings of the London Mathematical Society*, 9:148, 1878.

[7] R. A. Girle. Commands in dialogue logic. *Practical Reasoning: International Conference on Formal and Applied Practical Reasoning, Springer Lecture Notes in AI*, 1996.

[8] C. L. Hamblin. *Fallacies*. Methuen and Co. Ltd., 1970.

[9] Calico Jack Ltd. http://www.calicojack.co.uk, 2005.

[10] J. D. Mackenzie. Question begging in non-cumulative systems. *Journal Of Philosophical Logic*, 8:117–133, 1979.

[11] N Maudet and F. Evrard. A generic framework for dialogue game implementation. In *Proceedings of the Second Workshop on Formal Semantics and Pragmatics of Dialog*, 1998.

[12] P. McBurney, D. Hitchcock, and S. Parsons. The eightfold way of deliberation dialogue. *International Journal of Intelligent Systems*, 2002.

[13] P. McBurney, S. Parsons, and M. Wooldridge. Desiderata for agent argumentation protocols. *Proceedings of the First AAMAS*, pages 402–409, 2002.

[14] J. McCarthy. Ai as sport. *Science*, 276(5318):1518–1519, 1997.

[15] J. McCarthy. Elaboration tolerance, 1998.

[16] D. Moore and D. Hobbes. Computational uses of philosophical dialogue theories. *Informal Logic*, 18(2 and 3):131–163, 1996.

[17] T. J. Norman, D. V. Carbogim, E. C. W. Krabbe, and D. N. Walton. Argument and multi-agent systems. In *Argumentation Machines*, chapter 2, pages 15–54. Kluwer Academic Publishers, 2004.

[18] S. Parsons and N. R. Jennings. Negotiation through argumentation. In *Proceedings of ICMAS'96*, pages 267–274, 1996.

[19] C. Reed and G. Rowe. Araucaria: "software for puzzles in argument diagramming and xml. Technical report, University Of Dundee, 2001.

[20] C. Reed and D. Walton. Towards a formal and implemented model of argumentation schemes in agent communication. In I. Rahwan, P. Moraitis, and C. Reed, editors, *First International Workshop on Argumentation in Multi-Agent Systems*, 2004.

[21] N. Rescher. *Dialectics.* State University of New York Press, Albany., 1977.

[22] H. Simon and J. Schaeffer. The game of chess, 1992.

[23] G. A. W. Vreeswijk. Interpolation of benchmark problems in defeasible reasoning. In *WOCFAI*, pages 453–468, 1995.

[24] G. A. W. Vreeswijk and F. P. M. Dignum. Towards a testbed for multi-party dialogues. In *Workshop On Agent Communication Languages*, pages 212–230, 2003.

[25] D. Walton. *Argumentation Schemes for Presumptive Reasoning.* Lawrence Erlbaum Associates, 1996.

[26] D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue.* SUNY series in Logic and Language. State University of New York Press, 1995.

[27] S. Wells and C. Reed. Formal dialectic specification. In I. Rahwan, P. Moraitis, and C. Reed, editors, *First International Workshop on Argumentation in Multi-Agent Systems*, 2004.