Multiple Agent Systems for Configuration Design

Stuart Watt and **Zdenek Zdrahal**
Knowledge Media Institute
Open University, Milton Keynes, UK.

Mike Brayshaw
School of Computer Science
University of Birmingham, Birmingham, UK.

Abstract

This paper investigates how the task of configuration design can be carried out using concepts of multiple agency. Configuration design is the task of selecting components from a predefined set to complete a system which meets a given functional specification and other design constraints. It is a class of task which is conventionally solved using a single agent reflecting an arbitrary balance of the design criteria chosen by the system designer. To study the efficacy of the multiple agent approach, we show how more of the original domain knowledge can be applied in an alternative form where each agent in a multiple agent design system has individual design goals and acts in negotiation with others in order to achieve those goals. Each agent's goal represents different domain axes upon which design decisions are based. At any point where different design decisions can be made, negotiation between these agents enables a balancing between the different agent's goals, and therefore these axes, to be achieved. Using the Sisyphus-2 benchmark configuration design problem, we will compare and contrast these methods to identify their relative merits.

1 Introduction

This paper compares and contrasts traditional methods and multiple agent architectures as techniques for performing the task of configuration design. Configuration design is a common engineering problem, involving component selection and arrangement, and thus presents an interesting area to apply AI technology. One particular example, the VT elevator design system (Marcus et al., 1988), has been proposed as the Sisyphus-2 benchmark for testing and comparing various AI techniques and methodologies. Taking this as a starting point we will demonstrate how a multiple agent architecture gives extra flexibility in use and transparency of design choice over a conventional system. Our argument will be structured as follows. First we shall describe the engineering task and domain under discussion. Then we shall discuss how traditional approaches would go about solving the problem.

We note some of the features and shortcoming of such procedures. Then we shall outline an alternative using a multiple agent paradigm. To put some flesh on this model we will then say in detail how this works for various parts of our chosen problem. Finally we discuss some of the advantages and disadvantages of the approach and draw some lessons for the future. We begin with a discussion of the problem domain.

2 Domain description of VT

Sisyphus-2 is a configuration design (CD) problem. The task can be understood as the selection of components from a predefined set in order to provide the functionality specified and meet additional criteria. If the artifact to be designed is complex the task needs a large amount of domain specific knowledge.

The configuration design task for VT (Marcus et al., 1988) is that of designing an elevator; the complete problem specification is described in (Yost, 1992). The domain is described in terms of a predefined set of *components* and *constraints* imposed on their properties and interconnections. Components are characterised by their *parameters.* The components are the modules from which the elevator can be configured and eventually built (e.g. various types of motors, cabs, cables, gear boxes, sheaves etc.) Some components and their parameters are active, i.e. the component participates in the final design. Initially, values for a subset of parameters are given as a requirement specifications for the design. There are, for example, parameters of the building, capacity and speed of the elevator, or required additional equipment, such as a telephone. The initial parameter set distinguishes individual cases. From these initial values new parameters are calculated and components are selected for the solution. The formulae for calculating new parameters are based on known principles of physics and engineering. If proposed values for parameters violate a constraint a correction must be made. CD can be viewed as a problem of achieving global consistency of the constraint network defined in terms of parameters and constraints.

For the purpose of this presentation we will significantly simplify the problem description. We assume that the interconnection of components is predefined and that parameters are calculated step by step in a fixed order known in advance. Under these assumptions the design can be described in terms of a set of parameters and their values. Values of parameters are either defined in advance (initial parameter set) or they are calculated from other parameters. Functions which calculate the values of parameters are called *dependencies.* Constraints are relations over the parameter set. Some parameters can be adjusted to fix violated constraints. The function for calculating the new value of the parameter is called a *fix.* Fixes are associated with constraints, that is, if a constraint is violated the associated fixes can be used as a remedy. A fix can be associated with several constraints. There are constraints without fixes, and if such a constraint is violated no solution for those values exists. Some fixes attempt to correct the constraint violation by using a domain-specific function to calculate a new value for a parameter from the old ones, the others simply increment or decrement the value and thus effectively search for a value satisfying the constraint. These fixes are called *incremental*.

In practice the goal is usually not to find any conflict-free design but the optimal one, or one close to it. In the VT domain the quality is measured by a generalised cost called the *desirability factor* associated with the fixes. The idea is that the initial selection of values for parameters is made as "cheap" (in terms of the generalised cost) as possible and thus the cost of the solution can be measured only in terms of the costs of fixes.

The following examples from (Yost, 1992) show a dependency and a constraint and its associated fixes:

> The dependency for parameter "Counterweight runby": The counterweight runby is equal to $R - (D - (F + B + H))$, where R is the counterweight bottom reference, D is the pit depth, F is the counterweight footing channel height, B is the counterweight buffer blocking height, and H is the buffer height.
>
> The constraints and fixes for parameter "Counterweight runby": the counterweight runby must be between 6 and 36 inches, inclusive. There are no fixes for the minimum. For the maximum, two fixes are possible: decrease the counterweight bottom reference by the amount the runby exceeds its maximum (desirability D1), or increase the counterweight buffer blocking height by steps of one inch (desirability D4).

The VT domain consists of 230 parameters, 31 of which have already initial values, so there are 199 dependencies for calculating the remaining parameters. There are 62 constraints, 38 of them having recommended fixes. One constraint has 5 fixes associated, one has 4 fixes, three have 3 fixes, six have 2 fixes and 27 constraints have just 1 fix. 24 constraints have no fixes associated, i.e. if these constraints are violated, they cannot be fixed. The total number of fixes is 45, 24 of them being incremental. Each fix has an associated desirability factor from D1 (most desirable) to D10 (least desirable) (Yost, 1992).

3 Traditional solution to VT

The original solution to the VT problem (Marcus et al., 1988; Yost, 1992) is based on proposing extensions to the current incomplete solution and revising it if some constraint is violated. This technique is called "Extend-Model-and-Revise" (EMR) (Motta et al., 1994) and is analogous to the backtracking algorithms for constraint satisfaction. In (Balkany et al., 1993) EMR is shown to be a very common technique for various CD problems. Alternatively, it is possible to complete the solution without paying too much attention to its consistency and then gradually fix all violated constraints. This approach (Motta et al., 1994) is called "Complete-Model-and-Revise" (CMR). In the mechanism for selecting fixes there is no significant difference between EMR and CMR.

The algorithm for fixing a violated constraint selects a constraint and attempts to fix it by applying the associated fixes. By applying a fix the value of a parameter is changed by the fix to its tentative value and this value is propagated by means of all possible dependencies thus calculating a set of tentative values. If these values do not remove the original constraint violation or if they produce a new one these tentative values are discarded and another fix is selected. Fixes are tested in order against their decreasing desirability factor. The strategy is as follows: first all fixes at the highest desirability level are tested individually. If the constraint violation persists fix combinations at that highest level are applied. Then individual fixes at lower levels are taken into consideration, and then these too are combined with those with gradually decreasing desirability values, and so on. If a fix combination removes the constraint violation the new values of the parameters are made permanent and next violated constraint is selected.

EMR can be understood as optimisation of the overall design cost by optimising each design step. The alternative strategy in CMR; if the criterion function is complex, the global optimisation, i.e. the optimal design, is not necessarily composed of locally optimal steps. One has to move away from the optimal solution path in order to find a globally cheaper path. We have implemented such solution by means of the A* optimisation algorithm (Zdrahal and Motta, 1994). The algorithm explores all possible paths and the solution accumulates costs of individual design steps to get the cheapest, i.e. the optimal one. However it is not easy to find a suitable heuristic function for the A* algorithm and one is restricted to the rigid search strategy define by this function.

These types of solution are very good when the area is well specified and the solution route known, but they compiles within them many of the design choices made in determining the eventual solution. As a result they has the following identifiable drawbacks:

1. the implicit nature of the system means that it is sometimes not obvious why a particular decision should be made. It tells you want to do, but not the justification,

2. it is rigid, the system hasn't much flexibility in the solutions that it can offer,

3. it only tells you a small part of the story, removing a critical part of the account, the actual conceptual design of the system by the designers,

4. because this design layer is missing you don't know how generally you can apply these solutions, nor can you easily extend the system into a new domain without working things out from the beginning every time.

Under some circumstances such predetermined action plans may be a feature. In general, though, we feel that the above four points reduce the desirability of the approach. In the next section we will show how multiple agents can be used to augment this solution strategy so as to keep and use to best effect original domain knowledge that that would otherwise be ignored in the design process. We begin this discussion with a general introduction to multiple agency.

4 Multiple agent systems

Defining an 'agent' is not trivial, because the word can have some very different meanings. Within artificial intelligence, it can sometimes just be used to refer to a computational process or task. In our sense, though, something deeper is implied. An agent is something that acts for someone, having been delegated a task by them.

The task that has been delegated will usually appear in the form of a 'goal,' something that the agent will persistently be trying to achieve. This goal may, and often does, involve maximising or minimising a value, cost or time for example.

The other key feature of agents is that they do not act in isolation. They are embedded in a larger social system, which sometimes only comprises a human, but more often than not includes a set of other agents. This is where the problem lies, because agents, as they represent different goals, will inevitably get into situations where they conflict. One agent will want to take one action and a different one another, but the actions are incompatible. The agents then need to negotiate in order to act, and this negotiation makes explicit the decisions behind resolving the conflict and allows the different perspectives to be balanced to achieve a joint optimality.

It is through this property that we see multiple agent systems making a useful contribution to configuration design, which usually has an implicit set of rules to resolve conflicts. By using agents instead, the negotiation makes the conflicts explicit and a visible part of the problem-solving process. The negotiations themselves can be based on simple voting or on more complex interactions including argumentation, 'speech act' statements, challenge of warrant and counter warrant, and confederacy. Good existing examples include for argumentation (Stutt, 1988), and for inter-agent dialogue (Galliers, 1988).

This also opens the negotiation process to the addition of other, new agents, without breaking the basic structure of the system, but still allowing new goals to be combined with the old ones. The first issues, though, are the methodological considerations involved in identifying the perspectives of the different agents.

5 Decomposition to agents

Within the Sisyphus-2 problem description there are several alternative ways that a possible solution can be structured in terms of multiple agents. There is, in general, no single way of structuring a solution which is 'better' than any other, but the advantage of using a multiple agent architecture is that it forces the designers to make explicit the conflict resolutions that would be implicit in a conventional design. The actual structure that is chosen should normally reflect those conflicts which are most important to the solution.

Each of the different structures will normally imply a very different negotiation strategy, because the different agents will have different categories of goal, and will therefore be negotiating about very different things.

In Sisyphus-2, as a configuration design problem, there are three primary alternative decompositions, representing the structure of the design, the design criteria, and the structure of the organisations involved in the design process. Darr and Birmingham's (Darr and Birmingham, 1994) approach to VT mixes the design criteria decomposition with agents in a fourth decomposition—that of the VT software designer—but we feel that this is an unwarranted deviation from the original problem specification (Yost, 1992) and, further, that this means that many of their agents are more like computational tasks than representations of valuable domain information.

Structural decomposition to agents

In an elevator system, there are a number of modules which can be designed relatively independently. In Sisyphus-2, for instance, there are four key components: the hoistway, the car assembly, the counterweight assembly, and the suspension system. A structural decomposition to agents would mean assigning each of these different subsystems to a separate agent.

When a conflict arises between two agents, whatever the class of conflict, it will need to be resolved between the agents responsible for the relevant subsystems. The agents must therefore be able to negotiate about many different kinds of conflict.

The structural decomposition is perhaps closest to traditional human design groups, where individuals are typically responsible for the different subsystems.

Design criteria decomposition to agents

When a conflict arises, whatever its origin, the kind of conflict gives some hints as to how it can be dealt with. The VT solution uses 'fixes' to deal with these conflicts, and each fix has an associated code representing its desirability. Some fixes, for instance, require special part design and some increase maintenance requirements. In Sisyphus-2, these fix desirability codes are rated on a linear scale, so that some fixes are always preferred over others.

This leads to the second possible decomposition to agents, based on design criteria which are an explicit representation of these desirabilities. One agent, therefore, will monitor maintenance requirements, preferring low ones, while another will try to avoid special part design. Instead of an implicit ordering, agents will deal with conflicts by negotiation according to the design criteria they represent. Thus the constraints are dealt with in an explicit negotiation at the knowledge level before they are mapped into specific mathematical calculations.

Since the VT solution implicitly follows the design criteria decomposition by its use of a scale of desirabilities, we will expand on this breakdown in more detail later.

Organisational decomposition to agents

A third possible decomposition may be found in the organisational structure of the elevator suppliers. Since the supply of an elevator system requires interaction between several different organisations, another possible structuring would be to allocate agents for each of the participating organisations. Negotiation between these agents would therefore mirror the negotiation between the real organisations.

While this is a perfectly valid decomposition to agents, it is difficult to implement because negotiation between real organisations is so complex. It is, therefore, less likely to be used in practice than the structural or design criteria decompositions.

6 Detailed analysis of the design criteria decomposition

Since the VT solution and the Sisyphus-2 description both implicitly apply an ordering of desirability codes for conflict resolution, we will analyse the decomposition closest to the spirit of Sisyphus-2 in more detail; that is, the design criteria decomposition.

Sisyphus-2 rates all fixes with desirability codes on a ten-point scale, rating from D1, meaning "no problem", to D10, meaning "changes major contract specifications." Any number of fixes at one level are better than a single fix at the level above, so potentially fourteen fixes at D6, "changes minor contract specifications" will be preferred to one at D7, "requires special part design."

This is very rigid, and while it may be valid in standard design situations it has a number of shortcomings. First, the classes are too crisp and too rigid for the details of some conflicts. By representing the design criteria as agents, say D6 and D10 as an agent which avoids contract changes and has quite a lot of authority, and D7 as an agent which avoids making special parts, an individual conflict can be investigated in more detail than would be permitted in the VT solution. Clearly, when contract specifications change as in D6 and D10, there will be a spectrum of possible scales of impact, not just two. Secondly, because these categories are broad and rigid, the system is opaque to explanation and justification in detail about why one particular fix was chosen over another. Negotiation between the agents can provide that justification.

Adopting a single desirability code for conflict resolution has other problems. Financial cost, for instance, can escalate when some parts cost significantly more than others, and the difference is not taken into account in the fix conflict resolution. Adding a cost agent can help overcome this, representing a more abstract design criteria, which can then be combined with the existing ones without breaking the existing architecture. We can imagine a whole range of these more abstract design criteria agents, others representing, perhaps, quality, installation time, and so on. There is clearly a balance to be reached in the selection of the appropriate level for an agent's interest, but, in a sense, by allowing the choice of levels, the system becomes far more flexible than its conventional cousin. Any true design criteria decomposition, therefore, would require analysing costs not as simple desirability codes, but as multidimensional values corresponding to the different design criteria. When, as in Sisyphus-2, these values are folded into one desirability code there is a loss of information which forces the resulting system into ruts which a human expert—and a multiple agent system—could be flexible enough to avoid.

Finally, there is, in Sisyphus-2, a complete separation between the control strategy used to deal with conflicts (propose, then revise) and the systems for resolving conflicts themselves. A multiple agent architecture would carry the additional benefit of being able to add agents which represent design process criteria, as well as the design criteria available for conflict resolution. That is, additional agents can be aware of the computational cost of certain paths of investigation and can negotiate with others to follow some paths over others.

Some solutions to Sisyphus-2 are built as constraint satisfaction systems, using the constraints and those dependencies which can be formulated as hard constraints to bind the parameters into a declarative solution. Fixes, and those dependencies which are best formulated as soft constraints, are sometimes just ignored in these solutions.

In (Motta and Zdrahal, 1995) it has been shown that soft constraints and fixes can be combined into additional disjunctive hard constraints. The terms in the disjunction correspond to mutually inconsistent legal ways of satisfying the constraint. Each term has associated a preference (desirability, cost) factor. The preference of a satisfied disjunctive constraint is defined as the preference of the satisfied term. For example, the default strategy (i.e. soft constraint) is to locate the counterweight at the midpoint between the rear side of the cab and the rear side of the hoistway. If this strategy violates a safety constraint an alternative strategy is to locate it a predefined distance from the rear side of the cab. The design task is to assign values to all parameters so that all constraints (i.e. the original hard constraints and the disjuncive hard constrains) are satisfied and the cumulative cost over all disjunctive constraints is minimised. These disjunctive hard constraints are the beginnings of agents in the Sisyphus-2 domain. By associating an agent with each disjunctive hard constraint, that is, with each degree of freedom of the system, the optimisation strategy can be defined by controlling the negotiation between the agents (Khedro and Genesereth, 1994).

This solution corresponds to a multiple agent system continuously interacting with an underlying constraint system. Agents are formed as groups of dependencies and fixes bound together by reference to shared design criteria, with the constraint system representing the hard physical constraints of the underlying configuration design and its dependencies. These agents explore the space of the constraint system rather as they would a traditional physical environment (Pollack and Ringuette, 1990), and many of the same principles apply .

The design criteria decomposition in the Sisyphus-2 case in particular shows a set of agents can be identified trivially which neatly matches that implicit in the problem specification. This initial set of agents is as follows:

A1 Minimise changes to the building

A2 Minimise changes to the contract

A3 Avoid making special parts where possible

A4 Minimise maintenance requirements

A5 Keep changes as small as possible

These agents are generalisations of those in the initial VT description (Yost, 1992). It is also clear that this can be easily extended by adding more agents implementing more abstract design criteria, including

B1 Minimise financial cost

B2 Minimise installation time

B3 Minimise installation ease

B4 Minimise number of revision steps

The next stage in our design allows us to form teams from agents dealing with related issues, almost like subcommittees. In this case, a natural team would bundle together A4, B2, and B3 to form an agent whose interest was the minimisation of the human resources cost to the supplier.

We show one such grouping in table 1. Here, four main agents are shown as combinations of the agents described above. Each is assigned a simple vote within its grouping, calculated so that decisions will be as close as possible to those of Sisyphus-2. This shows how by allowing for variation in more factors a system can explore the space of possible solutions around Sisyphus-2 while still remaining broadly within its parameters.

Agents need not be restricted to the domain level. The problem solving strategy that is used to get to the solution can also affect the solution with respect to the agent's interests. It seems natural, therefore, to add agents with an interest in the actual problem solving strategy. For instance, the space between the EMR strategy and the CMR strategy can be filled by a 'boldness' (Pollack and Ringuette, 1990) which can be implemented either as a property of each interested agent or as a 'facilitator' agent (Khedro and Genesereth, 1994) responsible for the collaboration. Also, in the traditional solutions no domain knowledge is applied in making the decision, but by providing agents with control over the problem solving strategy and

Agent			Votes	
G1		Minimise installation impact	10	
	A1	Minimise changes to the building		9
	A2	Minimise changes to the contract		10
G2		Minimise human resources	2	
	A4	Minimise maintenance requirements		2
	B2	Minimise installation time		0
	B3	Minimise installation difficulty		3
G3		Minimise complexity	6	
	A3	Avoid making special parts where possible		7
	A5	Minimise the size of changes		8
G4		Minimise financial cost	0	

Table 1: One possible agent grouping equivalent to Sisyphus-2

over the tactics used to achieve a good solution, a coherent strategy can naturally emerge.

Multiple agent architectures have the advantage of forcing the designer to keep explicit the different perspectives on the design process. That is, they are as much a methodological tool as an implementational one. As we have shown in this one instance of the design criteria decomposition, the advantage of this methodology is that key domain knowledge is not lost during the system development, but it can be retained to allow exploration of a space of different possible solutions.

7 An example: agents in the revision procedure

As an example, we will show an example analysis of agent-based problem solving in the revision part of the CMR solution. Since there is only one design extension procedure for each parameter the initial completed model has the same parameter values regardless of the order of their calculation. For the test case in (Yost, 1992) these values violate the constraints in table 2.

A static analysis of the dependencies between constraints shows that the fixes for the first constraint, MIN-PHL, do not affect any other constraints in the table. The ‘revise’ task can therefore be divided into two completely independent tasks: (a) revise the design in order to fix MIN-PHL, and (b) revise the design to fix constraints MIN-MBSM, LV-MM, MAX-TR, MAX-VF, MIN-HCSF, and MAX-MGP. Task (a) always has the same solution. The solution to task (b) depends on the order in which the constraint violations are processed: there are 6!, or 720, possible sequences of the 6 violated constraints, but this is halved by a domain rule which prescribes that MAX-MGP must always be resolved before MAXTR (Yost, 1992, p34).

Each of the remaining 360 possible sequences either produce a solution with a

Number	Constraint	Mnemonics
1	`platform-to-hoistway-left`	MIN-PHL
2	`machine-beam-section-modulus`	MIN-MBSM
3	`motor-model`	LV-MM
4	`hoist-cable-traction-ratio`	MAX-TR
5	`car-guiderail-vertical-force`	MAX-VF
6	`hoist-cable-safety-factor`	MIN-HCSF
7	`machine-groove-pressure`	MAX-MGP

Table 2: Constraints violated in CMR after the 'propose' task

cost A1-A5 or B1-B4, or fail to find a solution because all fixes are rejected as they generate new constraint violations. The actual sequence chosen decides the cost of the solution, or whether there is a solution at all. We evaluated all the possible sequences, and found that 210 failed to find a solution, and 150 found solutions that were classified into four groups according to cost: 'cheap,' 'medium cost,' 'expensive,' and 'very expensive.' We also classified the sequences depending on the number of revision steps they took to reach a solution, this time into three groups depending on whether they required 5, 6, or 7 revision steps.

To compare the sequences we introduced a binary relation '*before*' whose arguments are constraints: *before(x,y)* means that constraint *x* is resolved before constraint *y* in the sequence. For the six constraints, bearing in mind that MAX-MGP must always be resolved before MAX-TR, this means that each sequence can be uniquely identified by a 15 dimensional binary vector of the possible *before* values. This set of 360 examples, together with their results with respect to solution cost was then processed by ID3 to generate the decision table in figure 1. Similarly, the examples together with their required number of revision steps was processed by ID3 to generate the decision table in figure 2.

Each of these two decision trees represents the space of possible strategies for an agent representing that criterion. When the two agents are both involved in the problem solving process, the differences in the decision trees means that there will be conflicts between the agents as each competes to maximise its criterion. The agents can then use all the negotiation strategies at their disposal to navigate the way through their respective trees while still aiming to maximise their criterion.

The two trees show the solution step choices that can be made by agents representing the two criteria of solution cost and number of revision steps. Because the decision trees are not the same, there will be conflicts between the agents as each competes to maximise its criterion. Straightforward minimaxing can be used to achieve this.

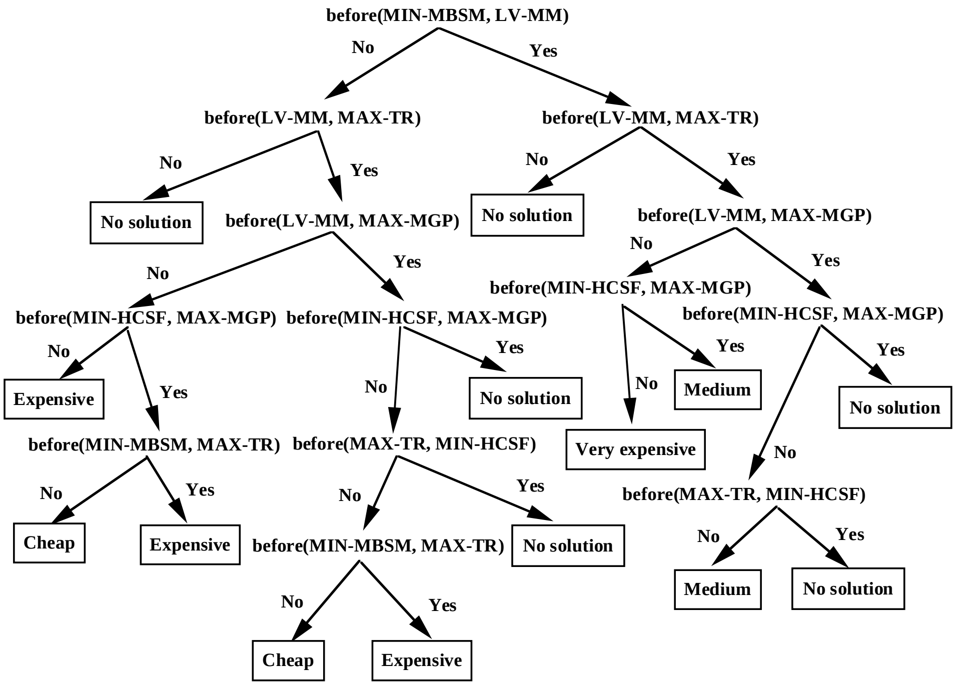

Figure 1: Decision tree for constraint ordering by solution cost

8 Conclusions

The paper has attempted to demonstrate how a multiple agent architecture is able to model explicitly many traditionally implicit design choices and trade-offs. There may be a cost trade-off here, to develop and implement a multiple agent system that truly models all the design choice points is a non-trivial task and a (potentially) expensive labour. Further, some of the higher level (and more aesthetic in judgment) design agents are a research issue in their own right. Once you have such a system it is more flexible to use, applicable in many different contexts, and is transparent in function. In addition, this transparency aids verification and validation of solutions paths and conclusions, where a traditional system with implicit hidden choice points, may be limited to a particular class of problems and presents users with no mechanism to see *why* something is the case. Further, given that all the implicit design choices were at some stage explicit in the traditional approach it is unclear that any extra expense incurred by writing down these design choices is significant. In conclusion, we believe that a clear and fuller model of what you are doing and why is an inherent outcome of adopting a multiple agent approach.

We have demonstrated how a multiple agent system can be used to provide a practical engineering solution for a particular problem. In general, whilst traditional methods work well for problems which don't change, have a rigid solution route, and where justifications for actions are of no concern to the final application

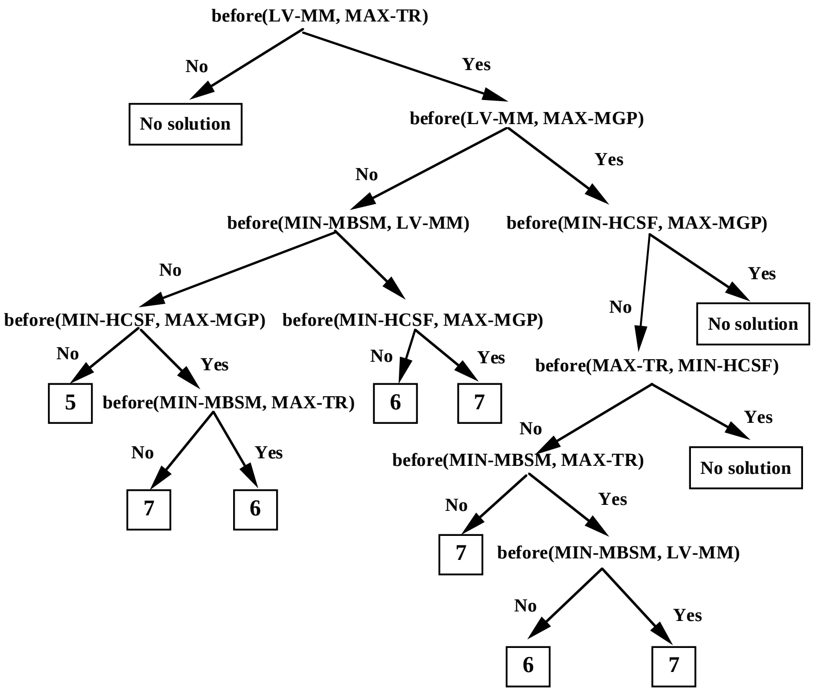

Figure 2: Decision tree for constraint ordering by number of revision steps

designer, we can identify four types of domain where we believe multiple agent solutions have the potential to provide improved solutions, namely:

- where the system needs to be extra flexible and applicable to many domains.
- where the domain itself changes or is in flux.
- where not just the solution but the solution path and justification are important to see
- where a designer wishes to experiment or investigate alternative scenarios.

In this last context, multiple agent design systems can also make interactive and dynamic design tools, opening up a whole new set of possibilities for conceptual innovation.

References

Balkany, A., Birmingham, W. P., and Tommelein, I. D. (1993). An analysis of several configuration design systems. *AI in Engineering, Design, and Manufacturing*, 7:1–17.

Darr, T. P. and Birmingham, W. P. (1994). Automated design for concurrent engineering. *IEEE Expert*, pages 35–42.

Galliers, J. R. (1988). A strategic framework for multi-agent dialogue. In *8th European Conference on Artificial Intelligence*, pages 415–420, Munich.

Khedro, T. and Genesereth, M. R. (1994). Modeling multiagent cooperation as distributed constraint satisfaction problem solving. In *Proceedings of the 11th European Conference on Artificial Intelligence, ECAI'94*, Amsterdam.

Marcus, S., Stout, J., and McDermott, J. (1988). VT: An expert elevator designer that uses knowledge-based backtracking. *AI Magazine*, 9(1):95–112.

Motta, E., O'Hara, K., Shadbolt, N., Stutt, A., and Zdrahal, Z. (1994). A VITAL solution to the Sisyphus II elevator design problem. In *8th Knowledge Acquisition Workshop*, volume 3, Banff, Alberta.

Motta, E. and Zdrahal, Z. (1995). The trouble with what: Issues in method-independent task specifications. In *9th Knowledge Acquisition Workshop*, Banff, Alberta.

Pollack, M. E. and Ringuette, M. (1990). Introducing the tileworld: Experimentally evaluating agent architectures. In *Proceedings of the 8th National Conference on Artificial Intelligence, AAAI'90*, pages 183–189, Boston, Massachusetts.

Stutt, A. W. (1988). *Argument and the Humanities*. PhD thesis, Human Cognition Research Laboratory, Open University.

Yost, G. R. (1992). Configuring elevator systems. Technical report, AI Research Group, Digital Equipment Corporation.

Zdrahal, Z. and Motta, E. (1994). An in-depth study of propose and revise problem solving. In *Proceedings of the Japanese Knowledge Acquisition Workshop, JKAW'94*, Hatoyama, Japan.