Modeling and Generating Tradeoffs for Constraint-Based Configuration

Eugene C. Freuder* and Barry O'Sullivan

Cork Constraint Computation Centre
University College Cork, Ireland
{e.freuder, b.osullivan}@4c.ucc.ie

Abstract. During an interactive configuration session we may reach a point where our desires can not be met. At this point we can consider "tradeoffs". Ideally, we would like the configurator to suggest appropriate tradeoffs to us. In this paper we have modeled tradeoffs in constraint-based configuration as additional constraints, and begun to study the issues involved in generating and evaluating such tradeoffs. We describe our basic approach in the context of a "toy" configuration problem based on the classic N-Queens problem. Initial experiments compare "proposal strategies" for generating tradeoffs. We lay out a program for further study, outlining the parameters of the problem space to be explored.

1 Introduction

Configuration is becoming a well studied design activity [8]. While there has been a growing interest in issues such as diagnosis of knowledge-bases for configuration [2], advice generation for design [1] and explanation generation [3], there is still a need for work on techniques which learn user preferences and use these to assist users achieve satisfactory configurations. This paper presents initial steps towards the development of such techniques.

During an interactive configuration session we may reach a point where our desires can not be met. At this point we can consider "tradeoffs". For example, in configuring a camera, we find that it is impossible to get one weighing less that 10 ounces with a zoom lens of 10X or more, so we consider a tradeoff: "I will increase my weight limit to 14 ounces if I can have a zoom lens of 10X or more." Ideally, we would like the configurator to suggest appropriate tradeoffs to us.

We have modeled tradeoffs in constraint-based configuration as additional constraints, and begun to study the issues involved in generating and evaluating such tradeoffs [4]. In Section 2, we describe our basic approach in the context of a "toy" configuration problem. We utilize the classic N-Queens problem, with the addition of user-generated "preference constraints" and system-generated "tradeoff constraints". Initial experiments compare "proposal strategies" for generating tradeoffs. Our results can be summarized as follows. Firstly, users who

* This work was performed while this author was at the University of New Hampshire.

have strong preferences can be more successfully assisted in finding acceptable solutions to a configuration problem. Secondly, when users have strong preferences, arc-consistency is a sufficient trigger for proposing tradeoffs to the user to successfully overcome inconsistency. This is a particularly interesting result since techniques based on arc-consistency are extremely useful in industry-based configurators. In Section 3, we lay out a program for further study, outlining the parameters of the problem space to be explored. Our approach enables us to address important but ill-defined questions like "what is a good tradeoff?" in a formal, experimental manner. Finally, a number of concluding remarks are made in Section 4.

2 A Case Study

In this section we describe our basic approach in the context of a toy configuration problem. We utilize the classic N-Queens problem, with the addition of user-generated preference constraints and system-generated tradeoff constraints. In Section 2.1 we present an approach which models preferences as unary constraints and tradeoffs as binary constraints. In Section 2.2 we present a number of proposal strategies for generating tradeoffs. In Section 2.3 we present some initial experimental results comparing these proposal strategies.

2.1 Modeling Tradeoffs for the N-Queens Problem

The configuration problem that will be studied here is based on the N-Queens problem. The user attempts to solve the configuration problem by interactively specifying a series of preference constraints to a constraint-based configurator. We assume that the user prefers higher column values. Thus, when the user proposes a preference constraint, we will assume it to be of the form $row \geq column$, where *row* corresponds to a row number in the N-Queens problem and *column* corresponds to the column value for that row. For example, the constraint $4 \geq 6$ means that the queen on row 4 should be placed in a column whose value is at least 6.

During an interactive session with the configurator, the user may specify a constraint which causes the set of preference constraints to become "over-constrained", identified using some measure of consistency (e.g the problem becomes arc-inconsistent). At this point our configurator attempts to recommend a set of appropriate "tradeoff" constraints to the user which she can accept before continuing to develop a solution for the configuration problem.

Thus, the interactive solving process can be summarized as follows:

- Repeat until user satisfied:
  - Repeat until "over-constrained":
    - User proposes a constraint: *"queen in row x must be $\geq$ column y"*
  - Repeat until user satisfied or there are no tradeoffs which can be used to overcome the inconsistency in the problem:

* System proposes a tradeoff: *"queen in row x must be $\geq$ column i and queen in row y must be $\geq$ column j"*

User specified preference constraints are modeled as unary constraints. For example, for the 8-queens problem a constraint which states that *"the queen in row x must be $\geq$ 6"* is modeled as:

$$C_x = \{6, 7, 8\}$$

On the other hand, tradeoffs are modeled as binary constraints. For example, a tradeoff constraint which states that *"the queen in row x must be $\geq$ column* 6 *and the queen in row y must be $\geq$ column* 7*"* is modeled as:

$$C_{(x,y)} = \{(6, 7), (6, 8), (7, 7), (7, 8), (8, 7), (8, 8)\}$$

During an interactive session with our configurator, when a tradeoff constraint, involving row x and y, is accepted, it replaces the unary preference constraints involving these variables.

Using binary constraints as the basis for modeling configuration problems has long be an accepted approach [7]. Modeling tradeoffs as binary constraints is useful, particularly since standard constraint processing algorithms can be used to reason over them.

2.2 Tradeoff Proposal Strategies

We have considered a number of tradeoff proposal strategies. Since we have assumed in our case-study that higher column values are preferred by the user, we have considered the following two strategies:

- Maximum Sum of Arc-Consistent Column Values;
- Maximum Sum of Viable Column Values.

We will discuss both of these strategies together below. In addition, we considered the following more general purpose tradeoff strategies:

- Maximum Viability;
- Minimum Viability;
- Pareto optimality.

Each of these strategies will be discussed here in turn.

Maximum Sum of Column Values Earlier we stated that in our case-study higher column values are considered better that lower ones. Therefore, a tradeoff may have a higher likelihood of being accepted if it is consistent with the preferences of the user. We have studied two strategies which attempt to maximize the sum of the column values of user-specified preference constraints: "maximum sum of arc-consistent column values" and "maximum sum of viable column values" values.

The maximum sum of arc-consistent column values strategy proposes a set of tradeoff constraints, each of whose sum of column values is maximal and is *arc-consistent* with respect to the remaining preference constraints specified by the user.

The maximum sum of viable column values strategy generates a set of tradeoff constraints, each of whose sum of column values is maximal and could yield a *solution* to the configuration problem given the remaining preference constraints specified by the user.

Maximum Viability This strategy generates a set of tradeoff constraints which have the potential to yield the greatest number of solutions (maximally viable) to the configuration problem, given the other preference constraints which have been specified by the user.

Minimum Viability This strategy generates a set of tradeoff constraints which could yield at least one solution (minimally viable) to the configuration problem, given the other preference constraints which have been specified by the user.

Pareto optimality Pareto optimality is an economics principle which is frequently used for comparing solutions to multiple objective optimization problems. Pareto optimality has also been used in the constraints paradigm for comparing the quality of solutions of constraint satisfaction problems which involve preferences [5].

In contrast to a single objective optimization problem which produces a single optimum (or a set of equivalent optima), multiple objective optimization produces a set of non-dominated (Pareto optimal) solutions. A set of non-dominated solutions is characterized by the property that every solution in the set is either better than every other solution to the problem, with respect to at least one of the objectives, or is at least as good as every other solution on all objectives [6].

Formally, given the set, S, of candidate solutions to a multi-objective optimization problem the Pareto optimal set, P_s, can be defined as

$$P_s = \{s_i | s_i \in S, not\_dominated(s_i, S)\}.$$

The predicate $not\_dominated(s_i, S)$ means that a candidate solution, s_i, is not dominated by any other candidate solution in S. Thus,

$$not\_dominated(s_i, S) \Leftrightarrow \forall s_j \in S, \neg dominates(s_j, s_i)$$

A solution, s_1, *dominates* another solution, s_2, if s_1 *improves* on s_2 with respect to any objective and s_2 does not improve on s_1 with respect to any objective. Thus,

$$dominates(s_1, s_2) \Leftrightarrow improves(s_1, s_2) \wedge \neg improves(s_2, s_1).$$

The predicate $improves(s_1, s_2)$ can be defined as follows,

$$improves(s_1, s_2) \Leftrightarrow \exists \langle F, I \rangle \in O, better(F(s_1), F(s_2), I).$$

where O is the set of objective functions, each objective function in O being a pair, $\langle F, I \rangle$, where F is a function and $I \in \{minimal, maximal\}$; the predicate *better* is defined as

$$better(x, y, maximal) \Leftrightarrow x > y$$

$$better(x, y, minimal) \Leftrightarrow x < y.$$

In the case-study being presented here, there is a preference on each variable (row) in our configuration problem – this preference is that each row take on a column whose value is maximal. Thus, our toy configuration problem can be regarded as a multi-objective optimization problem. The Pareto optimality strategy generates a set of tradeoff constraints which could yield a Pareto optimal solution to the configuration problem given the other preference constraint specified by the user.

2.3 Evaluation of Proposal Strategies

Experiments were made on the 8-Queens problem. Experiments involved simulating the interaction between a user and a configurator.

We simulated the human user on a combination of two axes. Firstly, we modeled the set of solutions to the problem which the user would find acceptable. This reflected the set of configurations that the user tried to achieve. Therefore, we considered the number of acceptable solutions as an experimental axis. Different points on this axis were chosen and solutions were generated at random for each point. In our experiments we chose points along the axis of acceptable solutions from the set $\{2, 4, \ldots, 92\}$, where 92 is the number of solutions to the 8-Queens problem. Our simulated user would accept a tradeoff if it subsumed a solution in the "set of acceptable solutions". In this way we could evaluate how the ability of our tradeoff proposal strategies to find acceptable solutions varied with the number of solutions which the user would find acceptable.

Secondly, we modeled the "strength" (greediness) of the user's preference constraints. Therefore, we considered the strength (m) of the simulated user's constraints as an experimental axis. Based on this axis our simulated user proposes new preference constraint bounds randomly chosen between m and N (for N-Queens) for different values of m. In our experiments we chose points along this axis from the set $\{2, 4, 6\}$. Thus, if we were simulating a user whose preference constraints had strength 4, a new constraint on row x would be of the form $x \geq y$, where y would be randomly chosen from the set of integers between 4 and 8 (for 8-Queens).

We simulated an interactive configurator. The configurator accepted user-specified preference constraints and incorporated these into the "built-in" model of the Queens problem. As the user specified constraints they were checked for consistency. If an inconsistency was detected, tradeoffs were proposed to the user based on a particular tradeoff proposal strategy. We used two different measures of consistency to trigger the proposal of tradeoffs: arc-consistency and full viability checking.

In order to evaluate our tradeoff proposal strategies we simulated an interaction between a user and a configurator as follows. Preference constraint were accepted from the simulated user while they were consistent. If the simulated user managed to solve the problem without encountering an inconsistency, tradeoffs were never proposed. When the user proposed a preference constraint, $x \geq i$, which had the effect of introducing an inconsistency into the model, the configurator proposed a set of tradeoffs to the user. The user would accept a tradeoff if it could yield a solution in the pre-determined "acceptable set" of solutions. If the user accepted one of the proposed tradeoff constraints, $C_{(x,y)}$, the unary constraints on x and y in the user's set of preference constraints were replaced with a single binary constraint representing the accepted tradeoff. If the set of tradeoff constraints proposed to the user was empty, or none was acceptable to the user, this was regarded as a failure. In the experiments presented here the user could not "backup" by retracting a previous preference constraint or tradeoff.

We understand that achieving success in less than all cases is not practical – we are just using this situation, where we measure the number of successes achieved "without backup", as an initial way of comparing, and getting a measure of, effectiveness. In future we will want to allow, and hope to have our system provide advice for, further user interaction to work towards a (good) solution in all solvable situations.

Before proceeding with a presentation of results, an example interaction between the user and configurator will be presented briefly here. In Figure 1 an example set of acceptable solutions, which will be used in the explanatory example here, is presented. Each solution is represented as a set of pairs, each of which denoting a row-column pair. Thus, the first solution can be read as *"the queen in row 1 must be in column 3, the queen in row 2 must be in column 6, etc."*.

```
{ {(1,3),(2,6),(3,8),(4,2),(5,4),(6,1),(7,7),(8,5)},
  {(1,4),(2,7),(3,3),(4,8),(5,2),(6,5),(7,1),(8,6)} }
```

Fig. 1. An example set of acceptable solutions.

An example of a simulated interaction between the user and configurator used in our experiments is presented in Figure 2. This is an example interaction for which the consistency metric used to trigger tradeoff proposals is a full viability check, the strength of the user's constraints is 2 ($m = 2$) and the tradeoff strategy used is "max sum of viable column values".

In the scenario presented in Figure 2 the user interactively proposes preference constraints until the problem is over-constrained. This occurs after the user has entered the fourth preference constraint, $4 \geq 5$. A single tradeoff is proposed, namely that the constraint on row 1 be "increased" so that its value is at least 4 while the constraint on row 3 is "decreased" so that its value is at least 3. Since

```
user proposes: 1 >= 2
user proposes: 2 >= 6
user proposes: 3 >= 6
user proposes: 4 >= 5
    over-constrained, so
        system proposes: { 1 >= 4 and 3 >= 3 }
    user accepts: 1 >= 4 and 3 >= 3

user proposes: 5 >= 4
    over-constrained, so
        system proposes: { 4 >= 8 and 5 >= 2 or
                           3 >= 3 and 5 >= 2 or
                           2 >= 7 and 5 >= 2 or
                           1 >= 4 and 5 >= 2 }
    user accepts: 4 >= 8 and 5 >= 2

user proposes: 6 >= 5
user proposes: 7 >= 2
    over-constrained, so
        system proposes: { 6 >= 5 and 7 >= 1 or
                           5 >= 2 and 7 >= 1 or
                           4 >= 8 and 7 >= 1 or
                           3 >= 3 and 7 >= 1 or
                           2 >= 7 and 7 >= 1 or
                           1 >= 4 and 7 >= 1 }
    user accepts: 5 >= 2 and 7 >= 1

user proposes: 8 >= 6

Solved: { 1 >= 4, 2 >= 6, 3 >= 3, 4 >= 8,
          5 >= 2, 6 >= 5, 7 >= 1, 8 >= 6 }
```

Fig. 2. An example trace of an interaction between the simulated configurator and the simulated user used in our experiments.

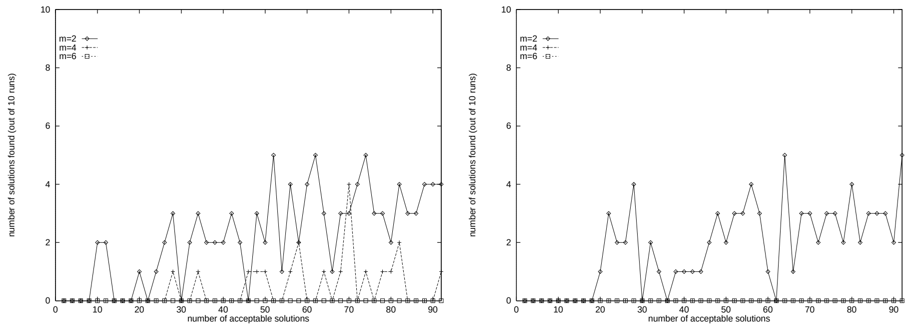
this tradeoff would allow the user to achieve a solution in her acceptable set, namely the solution

$$\{(1,4),(2,7),(3,3),(4,8),(5,2),(6,5),(7,1),(8,6)\}$$

the tradeoff is accepted by the user, who proceeds by proposing constraints on the remaining variables. From Figure 2 it can be seen that the user eventually completes a set of preference constraints which permits the above acceptable solution.

Using the experimental design discussed above, results from our initial experiments using the tradeoff proposal strategies presented in Section 2.2 are now presented. Note that in the graphs presented the x-axes are labeled "number of acceptable solutions" while the y-axes are labeled "number of solutions found (out of 10 runs)". The x-axes should be interpreted as the number of solutions of the 8-Queens problem that the user finds acceptable. The y-axes should be interpreted as the number of times that the user found a set of preference constraints which completely described an acceptable solution.

Maximum Sum of Column Values As discussed earlier, we have studied two strategies which attempt to maximize the sum of the column values described in the set of user-specified preference constraints: "maximum sum of arc-consistent column values" and "maximum sum of viable column values". Figure 3 presents the results obtained for the "maximum sum of arc-consistent column values" strategy.



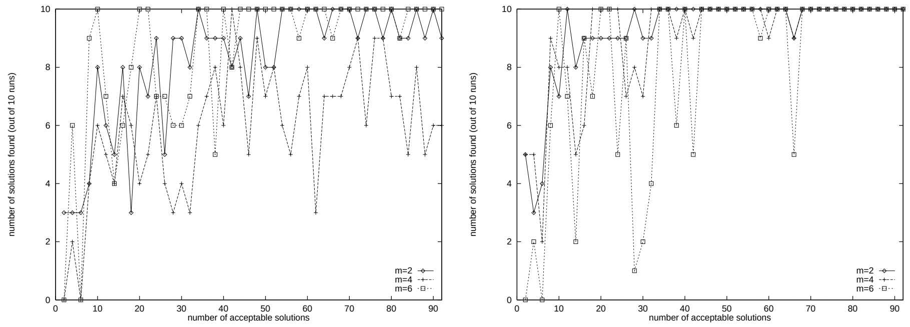
(a) Generating tradeoffs when arc-inconsistent

(b) Generating tradeoffs when viability check fails

Fig. 3. The performance of the "maximum sum of column values with arc-consistency" strategy.

From Figure 3 it can be seen that the "maximum sum of arc-consistent column values" strategy performed quite poorly, effectively only finding solutions in approximately 20% of cases above a certain number of acceptable solutions. The reason for this behavior is that this strategy is extremely greedy. When an inconsistency is detected, either through arc-consistency (Figure 3(a)) or full viability checking (Figure 3(b)), this strategy is locally very greedy. In particular, it only suggests tradeoffs which are arc-consistent with respect to the set of user-specified preference constraints and inherent N-Queens constraints, and which would maximize the sum of column values. Thus, this strategy does badly because it works against the distributions found in the solutions to the N-Queens problem. As a consequence, the user is rarely presented with tradeoffs which do not offer the possibility of finding an acceptable solution.

Also notice that it only for very weak preference constraints ($m = 2$) that the strategy find solutions. This also supports the argument that the strategy works against the distributions in the N-Queens problem. The strategy only finds solutions when the user is not very greedy, in terms of preference constraints, and is willing to accept a large number of solutions.



(a) Generating tradeoffs when arc-inconsistent

(b) Generating tradeoffs when viability check fails

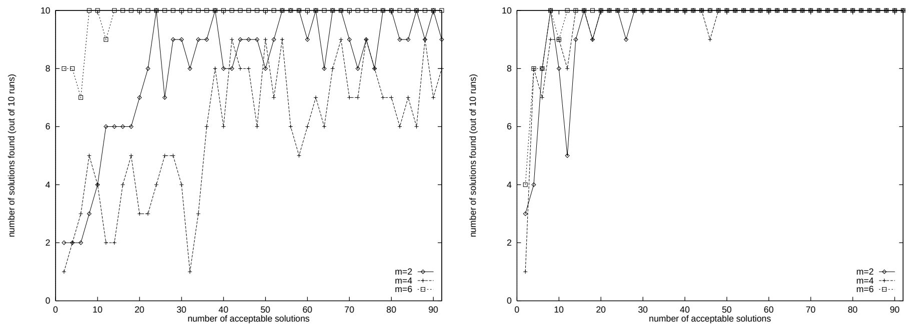
Fig. 4. The performance of the "maximum sum of column values with full viability checking" strategy.

By modifying the level of viability checking done by the previous strategy we achieved dramatically better results with the "maximum sum of viable column values" strategy. This strategy only recommends tradeoffs which it knows could lead to a full solution to the problem. Thus, while greedy, it attempts to work with the distributions found in the solutions to the N-Queens problem. The performance of this strategy is presented in Figure 4.

One of the most interesting aspects of the performance of the "maximum sum of column values with full viability checking" strategy is that its performance is quite volatile for smaller sets of acceptable solutions. This can be seen quite clearly in Figure 4(b), but is also evident in Figure 4(a) for user preference strengths of 2 (m=2) and 6 (m=6). The fact that either high strength (m=6) and low strength (m=2) preferences are more successful at finding solutions (Figure 4(a)) is a consequence of the tradeoffs working with both the distribution of solutions in the Queens problem and their symmetries. This trend will appear far more clearly in the results of our remaining experiments.

It should also be noted that for high strength preferences (m=6), solutions are found consistently regardless of the type of consistency check used to trigger tradeoff proposals. This is significant since it implies that arc-consistency can be used to generate good tradeoff proposals to the user. Arc-consistency is of much more practical use than full viability checking. This trend will also appear far more clearly in the results of our remaining experiments.

Maximum Viability This strategy proposes tradeoff constraints which could yield the maximal number of solutions to the configuration problem given the other preference constraints that have been specified by the user. From Figure 5, and in particular in Figure 5(b), and for high strength and low strength preferences in Figure 5(a), it can be seen that this strategy performed particularly well in terms of its ability to consistently find acceptable solutions.



(a) Generating tradeoffs when arc-inconsistent

(b) Generating tradeoffs when viability check fails

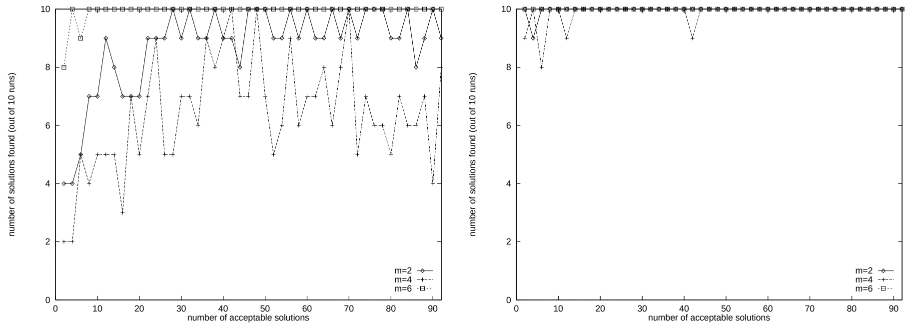
Fig. 5. The performance of the "maximum viability" strategy.

One of the most interesting features of these graphs is the fact that stronger preference constraints ($m = 6$) appear to behave best in the sense that the user successfully finds acceptable solutions on a consistent basis. This is quite obvious in Figure 5(a). This behavior might at first appear to be counter-intuitive; one would normally expect a min-commitment approach to be more successful. However, the reason for this phenomenon is that preference constraints are stated in the form $row \geq column$. Therefore, higher column values are tighter constraints. An inconsistency is more likely to be encountered earlier using stronger preference constraints, with the effect that we can rely on tradeoff proposals to steer the user clear of dead-ends.

It is also interesting to see how successful a tradeoff proposal trigger arc-consistency is when the user has strong preferences (m=6). As in the results for the previous strategy, this is of significant practical value.

However, one of the most significant disadvantages of the "maximum viability" strategy is that is needs to solve the enumeration CSP in other to determine which tradeoffs are maximally viable, i.e. discover which tradeoffs describe the largest number of solutions to the problem. This is a serious limitation for using this strategy in practice.

Minimum Viability and Pareto Optimality Both the "minimum viability" and "Pareto optimality" strategies found acceptable solutions for almost every point along the axis of number of acceptable solution when using a high preference strength. Both of these strategies propose tradeoffs which are have the capability to yield solutions to the configuration problem. Therefore, they will almost always generate tradeoffs which are acceptable to the user.

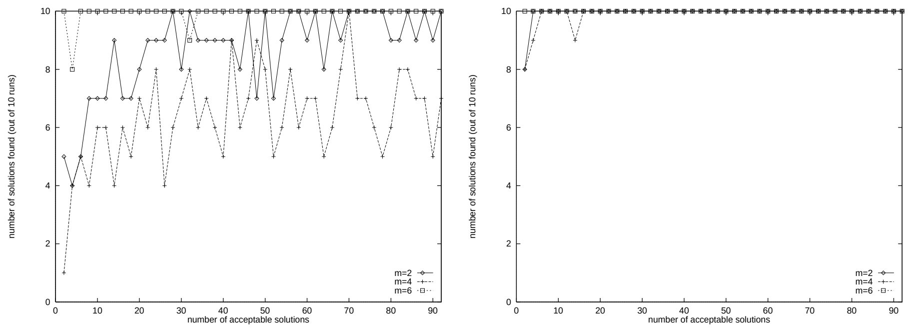


(a) Generating tradeoffs when arc-inconsistent

(b) Generating tradeoffs when viability check fails

Fig. 6. The performance of the "minimum viability" strategy.

Figure 6 presents the performance of the "minimum viability" strategy. It is worth noting that the performance of this strategy is similar to that of the "maximum viability" strategy. However, since this strategy generates tradeoffs which are minimally viable (i.e. describe at least one solution to the problem), there is a disadvantage associated with it, namely, that a large number of tradeoffs are proposed. In an interactive environment this could cause a certain amount of information overload from the perspective of the user. However, it may be possible to alleviate the problems associated with this by using an anytime approach to generating tradeoffs based on "minimum viability".



(a) Generating tradeoffs when arc-inconsistent

(b) Generating tradeoffs when viability check fails

Fig. 7. The performance of the "pareto optimality" strategy.

At the outset we assumed that the user preferred higher column values for each of the rows in our toy configuration problem based on N-Queens. From a Pareto optimality perspective this means that all solutions to the problem are equally good – no solution is dominated by another. Thus, this is why, in this case, this strategy behaved similarly to "minimum viability" (Figure 7).

There may be more effort required to generate tradeoffs based on Pareto optimality than on "minimum viability". Pareto optimality defines an ordering over the solutions to a problem, so we cannot start to eliminate solutions until we have at least two. In contract, the minimum viability strategy proposes tradeoffs as long as there is at least one solution possible involving the tradeoff. However, if we were to select a subset of the columns in our toy problem on which we defined our maximal preference, Pareto optimality should generate more relevant tradeoffs to the user.

Finally, once again in both cases, high strength preferences coupled with arc-consistency as a tradeoff proposal trigger performed extremely well.

3 A Framework for Further Study

In Section 2 we described our initial experiments on tradeoff generation in the context of a toy configuration problem based on the well-known N-Queens problem. However, there is considerable scope for study which extends this basic approach. In this section we lay out our program for further study, outlining the parameters of the problem space to be explored. Our approach enables us to address important, but ill-defined, questions like "what is a good tradeoff?" in a formal, experimental manner.

Essentially, we are interested in further exploring the general notion of tradeoff generation for constraint-based configuration as a technique for assisting users achieve their objectives by learning from their preferences. Our basic model is that tradeoffs can be represented as binary constraints. We are starting from an assumption that tradeoffs can be handled with independent, conjunctive tradeoffs. Once the first tradeoff involving some subset of the variables in a problem is accepted, the binary constraint representing the tradeoff is incorporated into the model of the problem to avoid the need for a separate, disjunctive unary constraint. Our agenda for studying tradeoffs is defined on a number of levels: *models of interaction* with the user; tradeoff *suggestion strategies*; *metrics* for measuring the effectiveness of these strategies; *experimental models*; and *practical application*. Each of these issues will be discussed below except for, due to space reasons, the issue of practical application. The issue of practical application has already been discussed in detail [4].

Models of Interaction One of our primary objectives is to assist the user to achieve a viable solution which the user will find acceptable in as few tradeoff interactions as possible. As we have seen in this paper, this may required the consideration of alternative approaches to measuring viability which may incur a higher cost that performing an AC check, for example, full solvability checking.

However, we have seen in this paper how well triggering the proposal of tradeoffs constraints using AC is when coupled with strong preferences from the user. The challenge here is trying to strike a balance between the effort expended on checking the viability of the user's preference constraints and our reliance on tradeoff suggestion strategy's ability to put the user back on the right track. Our initial experiments give us some insight on how to proceed here. In the approach adopted in this paper, we have seen how greedy tradeoff models can be more successful at finding acceptable solutions. This is a different behavior to that which we reported earlier [4], due largely to a reinterpretation of the notion of tradeoff constraint". Therefore, we need to identify some taxonomy which associates appropriate interpretations for the meaning of the phrase "tradeoff constraint" and tradeoff proposal strategies with particular classes of problem and particular classes of user.

Tradeoff Suggestion Strategies Another issue that needs exploration is the effect of the tightness and density of random problems on the likelihood of success for various tradeoffs strategies. In particular, an investigation on how to match greedier tradeoff suggestion strategies with different classes of random problems classified according to density and tightness would be of considerable interest. We have already seen how constraint tightness can affect the success rate associated with a particular tradeoff strategy.

There is also considerable scope for developing more intelligent tradeoff proposal strategies which attempt to ensure that a solution is found if one exists. One approach to developing more intelligent tradeoff proposal strategies may be by incorporating some form of backtracking, either user- or system-driven. These strategies would enable users to backtrack from situations where no tradeoff was available to a point where a tradeoff was accepted which, in hindsight, was a bad choice. Further extensions to the generic concepts of tradeoff strategies could be made by incorporating costs, allowing disjunctive tradeoffs etc.

Metrics for Tradeoff Strategies In order to judge the suitability and quality of various tradeoff proposal strategies, some metrics are required. Essentially, we see several dimensions to the question of metrics for tradeoff strategy evaluation. We discuss some of these briefly below.

Firstly, tradeoff strategies can be compared on the basis of their ability to find a satisfactory solution. In our initial experiments we defined a satisfactory solution as one which was in a predetermined set of acceptable solutions.

Secondly, tradeoff strategies can be compared on the basis of the number of interactions with the user they require before finding a satisfactory solution. Obviously, fewer interactions are better, since this would mean that we are minimizing overhead on the human user.

Thirdly, tradeoff proposal strategies can be compared on the basis of the amount of information we can learn about the user. Using our general model of tradeoffs, where a tradeoff is handled with independent, conjunctive constraints. Through an interactive process with the user we can learn about the complexities of the user's preferences from the tradeoffs that are accepted. In our initial experiments we can regard the pre-existing acceptable set of solutions as a very crude model of this acquired knowledge.

Finally, we can compare strategies based on "effort". For example, choosing tradeoffs that satisfy AC, will generally, take less effort than, but may not be as successful as, using viability. Closely, linked to the effort metric is execution time, which is an obvious measure of the performance of a tradeoff proposal strategy. Here, execution time refers to the speed at which a configurator, based on a particular tradeoff strategy, can respond to the user. Obviously, the faster a tradeoff proposal strategy can present the user with a set of alternative tradeoffs the better.

Experimental Models In Section 2.3 we presented the experimental design for our initial experiments. These experiments involved the evaluation of a number of tradeoff proposal strategies against two experimental axes: (a) the number of acceptable solutions and (b) the strength of the simulated user's constraints, and using two different triggers for proposing tradeoffs: (a) arc-consistency and (b) full viability checking. While we have already begun to consider differing combinations of proposal methods and viability checking, we need to investigate whether solutions fall into classes based on which tradeoff proposal mechanism which (a) can find them, and (b) most efficiently finds them.

A number of other important experimental models are also possible by considering relationships between classes of solutions and classes of tradeoffs, where tradeoff proposals are based on domain or user models, either given or acquired. Finally, incorporating costs into the framework is an interesting issue. More intelligent tradeoff strategies and experimental models can be developed here based on Pareto optimality and tradeoff domination.

Currently, our experiments capture an aspect of problems with ordered domains. For example, in configuration and design, people often have a preference defined over non-ordered domains such as colors. There are a number of ways in which we can extend our approach to remove this restriction. Some obvious approaches to overcoming this restriction are either to learn a user's preference over unordered domains through tradeoff constraints, or to *a priori* interactively define an ordering over such domains with the user [5].

4 Conclusion

The ability of configurators to generate tradeoffs to users during interactive configuration is valuable. In this paper we have modeled tradeoffs in constraint-based configuration as additional constraints, and begun to study the issues involved in generating and evaluating such tradeoffs. We described our basic approach in the context of a "toy" configuration problem and reported the results of some initial experiments to compare different "proposal strategies" for generating tradeoffs. We have found that users who have strong preferences can be more successfully assisted in finding acceptable solutions to a configuration problem. In addition, when users have strong preferences, arc-consistency is a sufficient trigger for proposing tradeoffs to the user to successfully overcome inconsistency. This is a particularly interesting result since techniques based on arc-consistency are extremely useful in industry-based configurators. Finally, we have laid out a program for further study, outlining the parameters of the problem space to be explored.

Acknowledgments

This work was supported in part by Trilogy. Professor Freuder is supported by a Principal Investigator Award from Science Foundation Ireland.

References

1. James Bowen. Using dependency records to generate design coordination advice in a constraint-based approach to Concurrent Engineering. *Computers in Industry*, 33(2–3):191–199, 1997.
2. Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Stumpter. Consistency-based diagnosis of configuration knowledge-bases. In *Proceedings of the 14h European Conference on Artificial Intelligence (ECAI'2000)*, pages 146–150, 2000.
3. Eugene C. Freuder, Chavalit Likitvivatanavong, and Richard J. Wallace. A case study in explanation and implication. In *In CP2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers*, 2000.
4. Eugene C. Freuder and Barry O'Sullivan. Modeling and generating tradeoffs for constraint-based configuration. In *Working Notes of the IJCAI-2001 Workshop on Configuration*, August 2001.
5. Barry O'Sullivan. *Constraint-Aided Conceptual Design*. PhD thesis, Department of Computer Science, University College Cork, Ireland, July 1999.
6. Sudhakar Y. Reddy. Hider: A methodology for early-stage exploration of the design space. In *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, August 1996. Irvine, California.
7. Daniel Sabin and Eugene C. Freuder. Configuration as composite constraint satisfaction. In *AAAI-96 Fall Symposium on Configuration*, pages 28–36, 1996. also in: Proceedings, Artificial Intelligence and Manufacturing Research Planning Workshop 1996.
8. Daniel Sabin and Rainer Weigel. Product configuration frameworks – a survey. *IEEE Intelligent Systems and their applications*, 13(4):42–49, July–August 1998. Special Issue on Configuration – Getting it Right.