Modeling and Generating Tradeoffs for Constraint-Based Configuration

Eugene C. Freuder
Constraint Computation Center
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
ecf@cs.unh.edu

Barry O'Sullivan
Constraint Processing Group
Department of Computer Science
University College Cork
Cork, Ireland
b.osullivan@cs.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. Finally, we discuss how our ideas might be applied to a real-world configuration problem.

1 Introduction

Configuration is becoming a well studied design activity [Sabin and Weigel, 1998]. While there has been a growing interest into issues such as diagnosis of knowledge-bases for configuration [Felfernig *et al.*, 2000], advice generation for design [Bowen, 1997] and explanation generation [Freuder *et al.*, 2000], 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. 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. 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. In Section 3.2, we discuss how our ideas might be applied to a real-world configuration problem in circuit board design. 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. This is the problem of placing N queens on a chess board of a size $N \times N$ so that no queen can capture another queen [Marriott and Stuckey, 1998]. In Section 2.1 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. This problem has been chosen since it is a well-known vehicle for facilitating explanation in the field of constraint processing. 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 generates 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 the interactive session with the configurator, the user may specify a constraint which causes the set of preference constraints to become inconsistent, identified using some measure of consistency (e.g arc-consistency, preference constraints are consistent with what the user would accept). At this point our configurator attempts to recommend a set of appropriate "tradeoff" constraints to the user which the user 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:
    - System proposes a tradeoff: *"queen in row x must be $\geq$ column i and queen in row y must be $\geq$ column j"*

Thus, 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, in this case-study, we model tradeoffs as a single binary constraint or as a conjunction of unary constraints. For example, a tradeoff constraint which states that *"if the queen in row x must be $\geq$ column 6 then the queen in row y must be $\geq$ column 7"* is modeled as:

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

or as

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

$$C_y = \{7, 8\}$$

During an interactive session with our configurator, when a tradeoff involving row x is accepted, it is incorporated into the set of user specified preference constraints.

Modeling configuration problems using binary constraints has long be an accepted approach in the literature [Sabin and Freuder, 1996]. Modeling tradeoffs as binary constraints is useful. In particular, standard constraint satisfaction techniques can be employed to assist a user to solve configuration problems modeled using binary constraints.

2.2 Tradeoff Proposal Strategies

Since we assume in our case-study that higher column values are preferred, the approaches we have taken in this early case-study were based on the following strategies: (a) Maximum Sum of Column Values, (b) Maximum Viability, (c) Minimum Viability and (d) Pareto optimality. Each of these strategies will be discussed here in turn.

Maximum Sum of Column Values

Earlier we stated our assumption that higher column values are considered better that lower ones. Therefore, when a tradeoff is to be presented to a user it has 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 described in the set 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 generates a set of tradeoff constraints each of whose column values are maximal and are arc-consistent with respect to the other user-specified preference constraints.

The maximum sum of viable column values strategy generates a set of tradeoff constraints each of whose column values are maximal and could yield a consistent solution to the configuration problem given the other preference constraints which have been specified by the user.

Maximum Viability

This strategy generates a set of tradeoff constraints which could yield the greatest number of solutions 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 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 a set of constraints which involve preferences [O'Sullivan, 1999].

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.

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 column have a value which is maximal. Thus, our toy configuration problem can be regarded as a multi-objective optimization problem. Thus, 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. 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. An acceptable solution to the configuration problem must satisfy the requirements of the user of the configurator application and must not violate any constraints which define the validity of a candidate configuration. 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 was contained in a solution in a randomly predetermined "acceptable set". In this way we could evaluate how the ability of the various tradeoff proposal strategies we presented above varies with the number of satisfactory solutions available.

We also considered the "strength" (m) of the simulated user's constraints as an experimental axis. Based on this axis our simulated user proposes new preference constraints 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 $\{2, 4, 6\}$.

In order to evaluate our tradeoff proposal strategies we simulated an interaction between the user and the configurator as follows. Preference constraint were accepted from the simulated user while they were consistent with the user's predetermined "acceptable set" of solutions. 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 when incorporated into the set of user-specified preference constraints made this set inconsistent, the configurator offered 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 a tradeoff, $y \geq j$, the constraint on y in the user's set of preference constraints was replaced with the accepted tradeoff. If the set of tradeoffs presented to the user was empty, or there did not exist an acceptable tradeoff, this was regarded as a failure.

Results from our initial experiments using the tradeoff proposal strategies presented in Section 2.2 will be presented below.

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".

The "maximum sum of arc-consistent column values" strategy performed extremely poorly, effectively never finding tradeoffs which were acceptable to the user. The reason for this behavior is that this strategy is extremely greedy. When an inconsistency is detected in the user-specified set of preference constraints, this strategy is locally very greedy since it only suggests tradeoffs which are arc-consistent with respect to the set of user-specified preference constraints and would maximize the column values. Thus, this strategy does badly because it works against the distributions found in solutions to the N-Queens problem. As a consequence, the user is presented with tradeoffs which do not offer the possibility of finding an acceptable solution.

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 1.

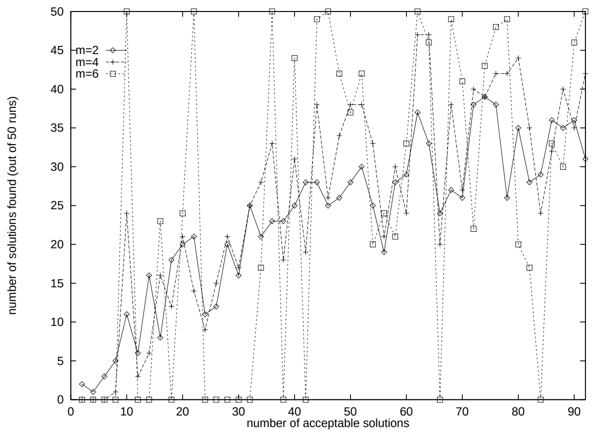

Figure 1: The performance of the "maximum sum of column values with full viability checking" strategy.

One of the most interesting aspects of the performance of the "maximum sum of column values with full viability checking" strategy is that as the strength, m, of the user's preference increases the ability of the user to find an acceptable solution by accepting tradeoffs becomes increasingly volatile. Using this strategy, to be able to consistently find acceptable solutions to a configuration problem, a user should not be too greedy in terms of preference constraint. Thus, it appears that combination of *user* who is content to adopt a least-commitment approach and a *configurator* which proposes tradeoffs which are known to be consistent with at least one solution to the configuration problem is best. However, as the user commits less, the quality of the solution is compromised since it will effectively describe a larger family of solutions. Therefore, it may be the case that the best strategy to adopt is application domain specific. For example, there may be no net benefit associated with a non-committal solution over having no solution at all. This is an interesting dilemma for those who must build configurators for general purpose applications.

Maximum Viability

This strategy generates a set of 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 2 it can be seen that this strategy performed particularly well in terms of its ability to consistently find acceptable solutions.

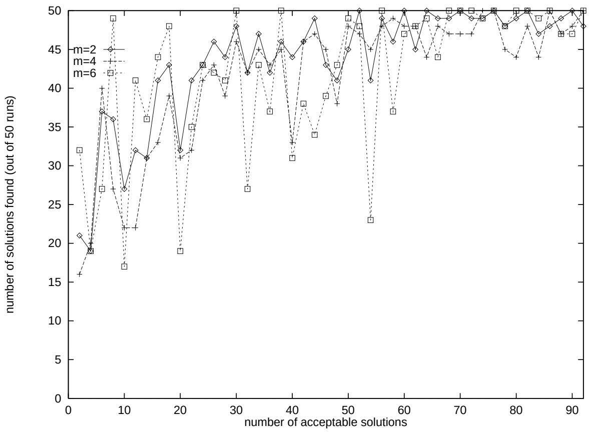

Figure 2: The performance of the “maximum viability” strategy.

Once again, as was the case for the “maximum sum of column values with full viability checking” strategy, it can be be seen that as the strength of the user’s preferences increases, the ability of the strategy to generate acceptable tradeoffs to the user which could yield solutions becomes more volatile. However, the “maximum viability” strategy is far more reliable at finding solutions. In addition, the most significant disadvantage of this strategy is that is need to perform a full solvability check in order to generate tradeoffs. However, performance was better than “maximum sum of column values with full viability checking”. Thus, it appears that a least-commitment approach is better.

Minimum Viability and Pareto Optimality

Both the “minimum viability” and “Pareto optimality” strategies found acceptable solutions for every point on both the axis of number of acceptable solutions and preference constraint strength. Both of these strategies propose tradeoffs which are have the capability to yield solutions to the configuration problem. Therefore, they will always generate tradeoffs which are acceptable to the user.

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. This is why, in this case, both of these strategies behave similarly.

There is more effort required to generate tradeoffs based on Pareto optimality. 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 would generate more relevant tradeoffs to the user.

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 Section 3.1 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. We will further demonstrate the utility of our approach on a real-world problem taken from the field of printed-circuit board design in Section 3.2.

3.1 Studying Tradeoffs

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 variable in a problem, the unary constraint for the variable is incorporated into it 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 the issue of practical application. A real-world example showing how our work on tradeoffs is of value is presented in Section 3.2.

Models of Interaction

In the initial experiments we presented here the consistency of the user’s preference constraints was interpreted as a set of constraints which were acceptable by the user, defined by the pre-determined set of acceptable solutions. It is also of value to check for consistency using just arc-consistency, since configurators tend to avoid doing full ”look for solution” viability checking, worrying about the cost. One of our primary concerns is to assist the user to achieve a viable solution which the user will find acceptable in as few tradeoff interactions as possible. This may result in the consideration of other approaches to measuring viability which may incur a higher cost that performing an AC check. For example, if we were to perform viability checking as full solvability checking we could still fail, if a solution can not be found through a tradeoff. However, it is expected that such an approach would help.

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. We have seen how greedy tradeoff models are less successful and how the strength of a user's preferences increases volatility in our ability to find acceptable solutions. Therefore, we need to identify some taxonomy which associates appropriate 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.

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 a solution 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.

A third metric for measuring the performance of a tradeoff proposal strategy is the amount of information we can learn about the user. Using our more 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 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. This basic model for evaluating alternative strategies can be extended in a number of ways.

For example, we have begun to consider differing combinations of proposal methods and viability checking. In addition, we are currently investigating 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 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 [O'Sullivan, 1999].

3.2 An Example Real World Application

In this section we present a real-world application of our work on tradeoffs from the field of printed-circuit board (PCB) design. A PCB can be regarded as a configuration of electronic components, connected by nets, placed on a multi-layer circuit board. Many of the common characteristics of configuration problems are present in particular classes of PCB design such as, specifying the detail of sub-systems within a pre-existing product structure and specifying the nature of the interface between the sub-systems. A typical example of PCB configuration is selecting options for PC motherboard assembly. However, in real world applications, there may be a need for design optimization which may, or may not, be considered part of the configuration process. An example of this from the field of PCB design is the placement of local fiducial marks for fine lead-pitch components – components whose pins are very close together. A fiducial is a marker placed on a PCB which facilitates precise optical alignment for placing fine lead-pitch components. In general, a PCB manufacturer would require that every component on a PCB would have two fiducial marks associated with it. Each of these fiducial marks would be one in two diagonally opposite corners of the component. An example arrangement of fiducial marks with respect to a fine lead-pitch component on a PCB is illustrated in Figure 3.

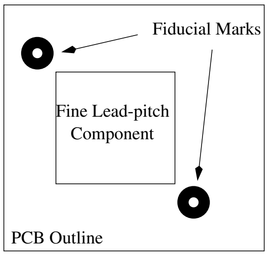

Figure 3: An example arrangement of fiducial marks with respect to a fine lead-pitch component on a PCB.

The placement of fiducial marks may appear to be a trivial issue. However, on an actual PCB, the problem is complicated by the existence of tracking (nets which connect components etc), obstructions caused by the placement of other components and areas which cannot, for one reason or another, be used for fiducial mark placement. A problematic scenario which frequently occurs in real-world designs is illustrated in Figure 4

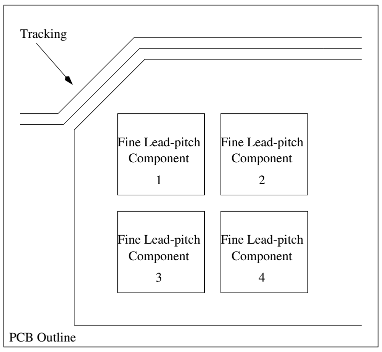

Figure 4: A typical, but problematic, PCB configuration for fiducial mark placement.

In general, the placement of fiducial marks must satisfy the following constraints

1. For each fine lead-pitch component two fiducial marks must be placed in two diagonally opposite corners of the component;
2. Fiducial marks should not be placed under the component;
3. Fiducial marks should be as close to the corner of the component as possible, typically within 5mm.
4. Fiducial marks should have a minimum separation of 15mm from each other.
5. A minimal number of fiducial marks should be used.

In addition, the placement of fine lead-pitch components must satisfy another set of constraints, the most relevant in this example being that fine lead-pitch components should share no more than one fiducial mark with another component. In practice the solution to the problem of fiducial placement reduces to acquiring what a user would regard as an acceptable meaning for "in the corner" of a component and "diagonally opposite" corners. Figure 5 illustrates the bounds of what a designer would accept as a tradeoff for "in the corner" of a component and "diagonally opposite" corners.

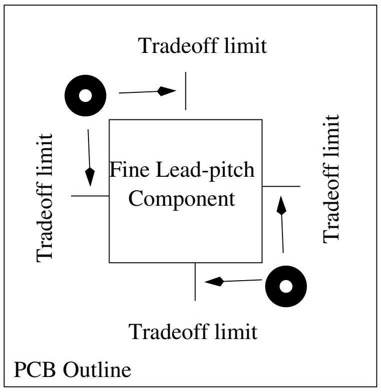

Figure 5: The limits of what would be regarded as in the corner of a component.

During the process of placing fiducial marks on the layout illustrated in Figure 4, the designer would attempt to locate a fiducial in each of the outside corners of each component with the consequence that the fiducial separation rule would be violated. Tradeoffs could be generated based on a user-model based on the scenario illustrated in Figure 5. Analogous to the interaction that a user would have solving the N-Queens problem, a designer could be presented with a series of tradeoffs which would assist the designer reach the acceptable solution illustrated in Figure 6.

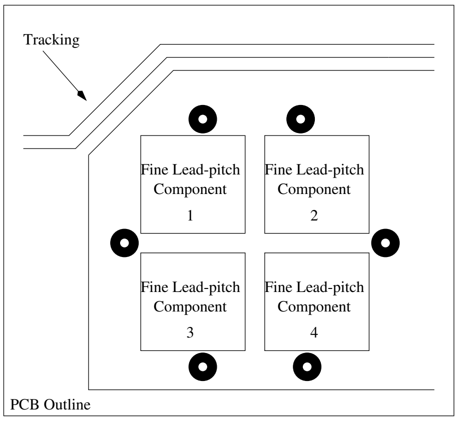

Figure 6: An acceptable placement of fiducial marks.

Although, at face value a very simple real-world application, this is a problem which is encountered by PCB designers on a daily basis. There is currently no systematic tool for assisting designers place fiducials.

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 laid out a program for further study, outlining the parameters of the problem space to be explored. Finally, we discussed how our ideas might be applied to real world configuration problems in circuit board design.

Acknowledgments

This work was supported in part by Trilogy.

References

[Bowen, 1997] 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.

[Felfernig *et al.*, 2000] 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.

[Freuder *et al.*, 2000] 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.

[Marriott and Stuckey, 1998] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. The MIT Press, Cambridge, Massachusetts, 1998.

[O'Sullivan, 1999] Barry Alan O'Sullivan. *Constraint-Aided Conceptual Design*. PhD thesis, Department of Computer Science, University College Cork, Ireland, July 1999.

[Sabin and Freuder, 1996] 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.

[Sabin and Weigel, 1998] 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.