CONTROL SEARCH USING RULES, TESTS AND MEASURES (its applications in the Harmonisation of Bach's Chorales)

Somnuk Phon-Amnuaisuk
somnuk.amnuaisuk@mmu.edu.my
Music Informatics Research Group,
Multimedia University, Malalaysia

Alan Smaill
smaill@inf.ed.ac.uk
Music Informatics Research Group,
The University of Edinburgh, Scotland.

August 30, 2004

ABSTRACT

Problem solvings may be viewed as search. In a search, knowledge plays a crucial role in guiding the search to an acceptable solution. We believe that an explicit separation between knowledge levels and the use of knowledge in problem solving allows knowledge-rich systems to be more flexible and powerful. With this aim, appropriate representation framework and inference mechanism are required.

This paper discusses the issues involved by viewing effective problem solving through the search control perspective. In this view, the data stream is defined as a stream of problem states returned after applying a control definition to a problem state ($C(S) \rightarrow \{s_1, s_2, ..., s_n\}$). The control definition is a control block constructed from three primitives, namely *rules, tests* and *measures*. A problem is said to be solved if a sequence of control definitions (control stream) applied to the start state yields the goal state e.g. $C_n(C_{n-1}(...(C_1(S))...)) \rightarrow G$. The stream of control procedure utilises knowledge to decide how to move from the start state to the goal state. We explain the search mechanism and how the knowledge can be applied to guide search in this view. We illustrate the approach with a case study in the harmonisation of Bach's chorales.

1 BACKGROUND

Search activities in AI are generally classified into two main classes: a brute-force search and a heuristics search. In the brute-force search approach, we are prepared to explore the whole search space. This is usually impractical in real life problem solvings since the search space is too large for exhaustive search. In practice, only part of the search space can be explored. Therefore, knowledge exploitation plays a crucial role in guiding the search to acceptable solutions.

In a state space search perspective, the state space can be represented as a graph, where each node represents different states of a given problem. The characteristics and shape of the state space are conditioned by the way a state (a node in the graph) is selected to be expanded and how the state is expanded. If plausible solutions are in the state space, then to solve a problem is to find a path to a desired solution in the state space. An informed search that makes use of the features of the space is a crucial ingredient for an effective traversing of this state space. Knowledge can be applied in two main activities:

- How is the node selected for expansion? For example, the depth first expands the node one by one from the stack.
- How is the node expanded? The transformation of one node to the next is important. For example, domain dependent knowledge helps by expanding the promising candidates first.

Search strategies which are less dependent on domain knowledge have an advantage of being general to many problem domains. General problem solver (GPS) and Genetic Algorithms (GAs) are examples of search strategies which employ general domain-independent knowledge to control and guide search. GPS employs means-ends analysis as a means to control search [7]. GAs' search is controlled by operators such as crossovers and mutations [4]. These operators are preferred to be domain independent to promote the generality of the problem solving mechanism.

However, the gain generality have to be traded with efficiency and (possibly) the quality of the solution. In problems where domain knowledge is well understood, a knowledge rich system seems to be a more appropriate approach.

2 AIMS

We propose a framework for making such control knowledge explicit in problem-solving and design situations. We want a language which allows us to express search regimes concisely, legibly and independently from object-level knowledge base. This should allow us to make use of object knowledge in many different ways by adopting different control strategies depending on the task at hand. Here, we make use of the distinction between meta-level and object-level knowledge as explored in e.g.[6], where we are interested in expressing control within a separate meta-theory.

2.1 A knowledge-rich controller

Amarel [2] describes how knowledge is used in the problem solving process as follows:

> "In order to decide what solution candidate the activity of the generator should focus attention on next, and what generation action (grammar rule) to select in continuing the construction of a solution candidate, the system must have an appropriate controller. Such a controller would operate under a guidance of control knowledge."

In this paper, we argue in favour of a knowledge-rich controller. Domain dependent and domain independent knowledge may be used to guide search. The system with more knowledge tends to out-perform systems with poorer knowledge provided that knowledge is used effectively [9].

In other words, how the knowledge is applied is also important to the success of the search. Here, we propose a framework which looks at a stream of states generated from a control definition. Knowledge is applied to manipulate the stream (e.g. filtering, reordering states in a stream) and to influence the execution of control definitions.

3 MAIN CONTRIBUTIONS

In this section, we describe our view in detail. We describe a problem domain D using a logical language $\mathcal{L}$. The problem is described with a set of clauses P and a set of constraints. The constraints are described over a set of free variables X. Let us define some terminologies used in this paper.

Definition 1 State property: *A state represents a set of problem states. A state property could be described declaratively as $\exists X_1, X_2, ..., X_n P_1 \wedge P_2 \wedge ... \wedge P_m$ where X_i are free variables in clause $P_1 ... P_m$.*

Definition 2 State space: *A state space can be represented using a directed graph. The problem is a start node in the graph while partial solutions and acceptable solutions are other nodes in the graph. The direction of an arrow indicates the transitional direction between any two states.*

Definition 3 Control Definition: *Control definitions are built up from domain-dependent atomic rules, atomic tests and atomic measures using domain independent control connectives.*

We do not give the full list of the connectives here. They include sequencing (**then**), alternatives (**or**), conditional tests, truncation of solution stream, filtering solution stream with a test, reordering part of the solution stream wirh respect to a measure, repeated successive use of given control definition, etc.

Definition 4 Control stream: *Let control stream be a stream of control definitions in the sense just introduced.*

Definition 5 Data stream: *Let data stream be a stream of states returned when a control rule is applied to a state. A control may generate a data stream of any length, possibly infinite. Each state in the data stream may generate further data streams. States in a data stream may be reordered using a measure. States in a data stream may be rejected using a test.*

3.1 Problem solving

We describe a problem using concepts in the domain D. The problem is described using a set of clauses P and a set of variables X. The problem solving task is to find appropriate values for $X_i, i = 1..n$ such that $P_k, k = 1...$ following from the knowledge base ($D \models P$). Solving for X_i is not a trivial task. Usually, the problem is too complex to be solved in one step and therefore must be decomposed to many small problems. The decomposition usually introduces some complications since the values of X_i is usually context sensitive. Therefore solving problems at a local level may not produce satisfactory solution at the global level. There is no panacea for this. An intuitive response would be to solve the problem hierarchically whereby less constrained variables are solved before higher constrained ones.

Generate and Test

Generate and test is a common problem solving template. A typical template may be described as follows: for a given problem, first construct candidate solutions (states), then select a state and determine if the solution is acceptable. Stop when an acceptable solution is found. Otherwise, repeat the generate and test steps. The selection of node for expansion is usually by the means of cost function.

Data streams and Control streams

Let us think of states as a data stream which is generated from a control stream. In the above generate and test example, the control stream is a simple loop of two control definitions. One is a rule which generates next states and the other is a test which either rejects or accepts a given state.

Let s_{ij} be a stream produced by applying a control definition c to a state s_i e.g. $c(s_i) \rightarrow \{s_{i1}, s_{i2}, ..., s_{in}\}$.

A simple depth first, left to right control would pick s_{i1}

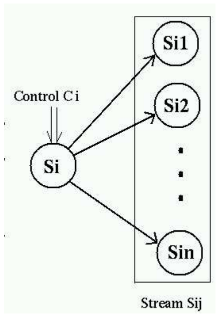


Figure 1: Control, state and stream

for a further investigation. This is an example of the application of domain independent knowledge in controlling search (i.e. the node is expanded without bias from domain knowledge).

Domain dependent knowledge may also be used to control and guide search (e.g. to influence the way the s_{ij} is generated by means of preconditions). Flexibility in data stream s_{ij} manipulation is dependent to the level of control allowed in the language..

Examples of stream manipulations are reordering states in a stream, filtering unwanted states from the stream, truncating a stream, appending two streams, etc. Applying a rule to state produces a stream. States in the stream may be accepted or rejected by tests. States in the stream could be ranked according to degrees of measurement. Tests can be seen as a set of hard constraints and measures can be seen as a set of soft constraints.

3.2 Control Language

A Mata-level and an Object level language

At this point, we want to introduce a distinction between object level and meta-level. The separation between an object level and a meta-level in a program is flexible (but not arbitrary). The two control steps (in the generate and test) mentioned earlier may be classified at the meta-level. Then, the control at the object level would be the control embedded in the selection of states and the application of operators in the object level program. These two levels may be written using different languages or the same language. A clear separation of the two levels usually brings more clarity and modularity to the program. Control search at the meta-level is usually more effective since the search space at meta-level is usually less complex than the search space at the object-level.

Control language

To be able to express control over the data stream, we need a suitable language. Here, we discuss a control language which is friendly to the concept of hierarchical task decomposition and stream manipulation. The control language has three functional primitives: the rules, the tests and the measures. These primitives are combined to form different compound (rule, test and measure) control definitions.

Definition 6 Rules ($\mathcal{R}$): *A rule transforms one state to the next state. Usually the value of state variables are non-deterministic (e.g. unknown or many acceptable values). Therefore a rule usually generates a stream of plausible next states: $R(s) \rightarrow \{s_1, s_2, ..., s_n\}$.*

Definition 7 Tests ($\mathcal{T}$): *A test returns `true` or `false`: $T(s) \rightarrow \{true, false\}$. It is used to test a state for required specifications.*

Definition 8 Measures ($\mathcal{M}$): *A measure returns a scale measurement of a state: $M(s) \rightarrow \{n | n \in \Re\}$. The measure does not reject the state, but labels each state with scaled measurement.*

Definition 9 Connectives ($\mathcal{C}$): *Connectives are symbols in the language that carry an unambiguous semantics. For examples, P then Q implies a sequence of order while P and Q does not imply a sequence.*

Each control statement is also called a control definition. The control definition performs one of the three functions of:

- Generate stream of plausible next states for a given state (by a rule definition).
- Reject or accept any given state (by a test definition).
- Label given states with scale (by a measure definition).

Transformation from one state to other states

The transformation from one state to the next state is by means of a rule control definition. Here we use capital S to mean a single state $\{s\}$ or a set of states $\{s_1, s_2, ..., s_n\}$ produced from a control C.

Let S be a finite set of states in the state space, let S_0 be a set of problem states and $S_0 \subset S$, let S_G be a set of acceptable solutions ($S_G \subset S$ and $S_0 \cap S_G$ is $\emptyset$), let $S_1, S_2, ..., S_m$ be any subsets of $\mathcal{P}(\mathcal{S})$; ($m > 1$), then there is a set of ordered $m + 2$ tuples of data stream $S_0 \times S_1 \times S_2 \times ... \times S_m \times S_G$, where $S_0 = \{s_{01}, s_{02}, ..., s_{0n}\}$; $S_1 = \{s_{11}, s_{12}, ..., s_{1n}\}$; where $S_1 \in \{s_{01}, s_{02}, ..., s_{0n}\}$ and so on.

Let C be an ordered n-tuple of a control stream $\{C_1, C_2, ..., C_n\}$ that traverses a path from the start state (s_0) to the goal state (s_G) e.g. $C_n(C_{n-1}(...(C_1(s_0))...)) \rightarrow s_G$. In such a case, the system can find a suitable substitution θ, such that $D \vdash_C P\theta$. The transformation is valid under a set of specified constraints in the control definitions C (for a particular set of problem statement P) and contents in the domain knowledge D.

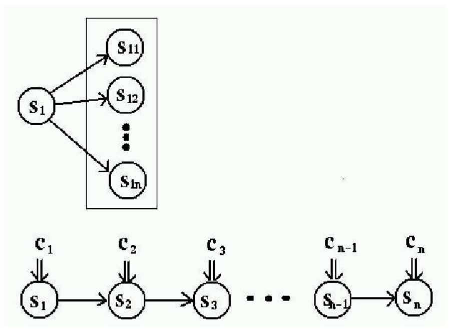


Figure 2: Apply a sequent of control to data stream

3.3 Control Search

Expressiveness of the representation language determines the coverage capacity of states in the state space. We would want the language to be abstract enough to ignore non-fruitful concepts and yet expressive enough to capture the desired properties in our representation.

Let $P(\mathcal{L}_o)$ be a problem statement described using an object level language $\mathcal{L}_o$. The problem statement is declaratively expressed using available data structures. The problem solving process is declaratively and procedurally expressed as a meta-interpreter (in the style of [11]) such that the domain theory D and control C together describe the search process.

The C control definition determines the transition between states and the way the search should progress from the start state to the goal state. This transition is usually non-deterministic since deterministic transition is only obtainable when constraints of a set of variables X (on the state) can be fully specified. Full specification of X is not trivial in most problems (i.e. due to incomplete information or dependency between concepts). The control usually makes a choice of X based on its current knowledge about the world. A strategy with the least commitment is the most common form of these variable instantiations.

There are usually many possible paths from the start to the end. However, sometimes the end state may not be reachable due to bad decisions which could lead to a loop or some remote area which makes it difficult to get back to the solution. This is a serious problem especially at the object level. This leads us to a discussion of the plausible remedies:

1. Guide search via hierarchical task decompositions
2. Reduce complexity of the search space with meta-level inference
3. Applying heuristics knowledge from an oracle to guide search
4. Applying feedback comments from a critique to guide search

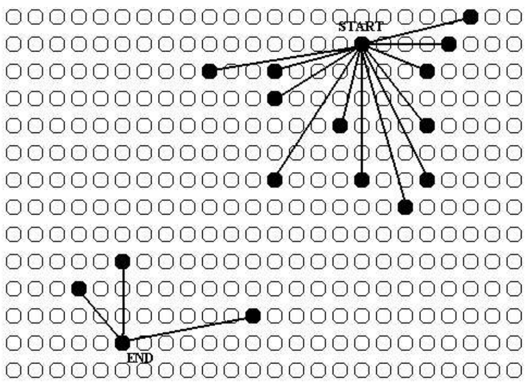


Figure 3: A start state and an end state in an object level state space

Guide search via hierarchical task decomposition

The hierarchical decomposition of tasks may be seen as the means to control search. For example, tasks c_1 and c_2 are two necessary tasks. If c_1 has 10 variables associated to it and task c_2 has 5 variables associated to it; then it is more effective to solve c_2 prior to c_1 since c_2 has a smaller number of variables associated with it. In this perspective the task with more details are left to be completed last.

Dividing the original problem into many smaller sub-problems proves to be useful in many cases especially when dependencies between sub-problems are minimum or the dependencies between sub-problems are in order (this means solving a sequence of small sub-problems resolves the dependency along the way).

Our control language fully supports this tactic. The rule, test and measure control definitions are hierarchically composed from the rule, test and measure primitives.

Reduce complexity of the search space

Usually, the search space tends to be very complex (more expressive and less tractable) at the object level. If we perform search at a low grain level (such as at the object-level), the control of the search seems to be harder to express in our program.

A meta-level language may be seen as a way to partition the object level state space into different partitions (see Figure 4) The search space at the meta-level is usually much smaller and therefore it is more effective to express the problem solving process at the meta-level.

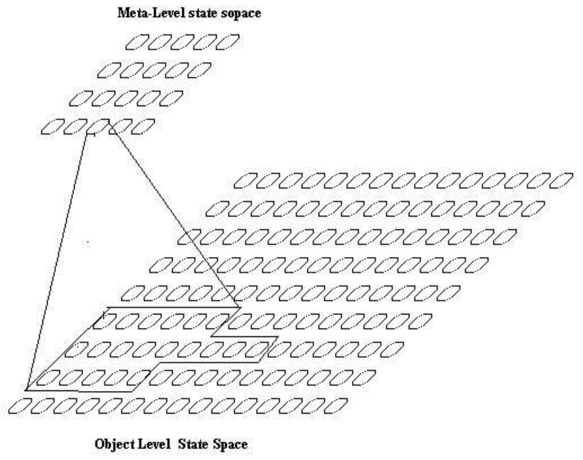


Figure 4: A meta-level state space is usually smaller than the object level state space

Our control language could be constructed in the object level and the meta-level. To fully benefit from the less complex meta-level search, it is required that any $O \vdash W$ at the object level must have a corresponding predicate at the meta-level; e.g. $M \vdash demo(O, W)$. This ensures that the search performed at the meta-level is the same as the search performed at the object level.

Consult oracles for Heuristics

This is a rich area where knowledge can be applied to guide search via rule, test and measure control definitions. Applying heuristics to control definitions could be in the forms of *preconditions* and *postconditions* of the control definitions.

Heuristics applied to rules influence the variables selection and instantiation. This affects the next states generation. Heuristics applied to tests and measures could be in the form of hard constraints which may reject a set of states in a stream (pruning part of the search space using Test); or in the form of soft constraints which can be used to order states into priority groups (preferences are expressed using measurements).

Consult critiques for comments

Applying heuristics as mentioned above in the forms of preconditions and post-conditions allows the program to make a better choice (bias from pre/post conditions). Another important search control information is from critiques. Critiques can provide useful feedbacks which has the tone of reflections/comments. This information could be used to dynamically determine the setting of parameters in the rules, tests or measure control definitions.

Backtracking

As long as we are able to re-examine any discarded portion of our search space, we are ensured of a solution (if it exists in our search space). Backtracking mechanism is crucial for this requirement.

4 EXAMPLES

The ideas presented in this paper has been implemented. We describe a test case that illustrates this approach. The problem is that of musical harmonisation in a given musical style (chorale) where a given melody line (a soprano) is accompanied with three other lines (e.g. alto, tenor and bass). The start state would be a given melody and the goal state would be a complete four-part harmonisation. This problem has been studied for example by [3, 5, 9, 8].

In the harmonisation problem, dependencies among local areas are common. For example, decisions on how to end the phrases of the Chorale is dependent on how the Chorale is started or how it is ended in the previous phrase. Therefore good sub-solutions from local areas may or may not form a good global solution. A successful control language must provide the means to allow flexibility in controlling dependencies among subproblems. We give examples of how to control search in the Chorale harmonisation task below:

4.1 Hierarchical task decompositions

To harmonise a given chorale melody, we may decompose the task into many major steps. We could write down the high level concept in our control language as follows:

```
definition(harmonise,
        rule:analyse(inputMelody)
        then outlineHarmonicProgression
        then outlineVoices
        then fillinVoices ).
```

The search space is hierarchically narrowed down with the decomposition of the main tasks (harmonise) into four main tasks. The task *rule:analyse(inputMelody)* is a primitive rule and the three other tasks are non-primitive and may be hierarchically constructed. Thus, the composition process can be conceived at a very abstract level.

In this manner, hierarchical structure and dependency of compositional processes can be explicitly controlled. In the skeleton plan above, the *outline of harmonic progression* attacks the problem at a higher level in hierarchy than the *outlineVoices* and the *fillinVoices*. Each of these definitions may be broken down into different control structures and may hold more than one control structure at one time. We illustrate this with the harmonic plan outline process below:

```
definition(outlineHarmonicProgression,
    repeat
        ( rule:selectPhrase(outlineHarmonicPlan)
        then outlinePhraseHarmonicPlan ) ).
```

```
    definition(outlinePhraseHarmonicPlan,
        rule:initialisePhrase(outlineHarmonicPlan)
        then rule:outlineChord(cadence)
        then rule:outlineChord(intro)
        then rule:outlineChord(body)
        then rule:closePhrase(outlineHarmonicPlan) ).
```

The *outlinePhraseHarmonicPlan* is a compound definition which may be defined in many different ways. We can compose the task so that it may do the introduction before the cadence or do the cadence before the introduction.

```
    definition(outlinePhraseHarmonicPlan,
        rule:initialisePhrase(outlineHarmonicPlan)
        then rule:outlineChord(intro)
        then rule:outlineChord(cadence)
        then rule:outlineChord(body)
        then rule:closePhrase(outlineHarmonicPlan) ).
```

As the harmonic progression of a phrase can be constructed in various ways, a process is usually dependent on its preceding processes. Again, this is how the hierarchy and dependency come into play in the control.

4.2 Express hard constraints with Tests

The list below shows some constraints in the harmonic vocabulary which should be maintained in the harmonisation task:

- Phrase should end with cadence pattern in some key (prefer perfect cadence).
- The end of the piece must be a perfect cadence in home key.
- The phrase before the last phrase should not end with tonic in home key.
- Harmony in the first phrase or the first two phrases should establish the tonality in the home key.
- Chorales normally modulate to other key in the middle part of the piece.
- etc.

One way to express these constraints over the data stream is by the means of test definitions. The control primitive **filter** *Control_definition* **with** *Test_definition* allows us to express this. Supposing we have a test:constrain(harmonicPlanOutline) which performs the constraints above, then we can write the above *outlineHarmonicProgression* definition in this way:

```
    definition(outlineHarmonicProgression,
      repeat
        ( rule:selectPhrase(outlineHarmonicPlan) then
        filter outlinePhraseHarmonicPlan
        with test:constrain(harmonicPlanOutline)) ).
```

By devising the tests in our control language, the tests can be plugged in/out as desired. This allows a greater flexibility in expressing the control.

4.3 Express preferences with Measures

We often do not want to disregard the plausible solution states but want to classify them according to our preferences. We express preferences using measures. The measurement provides an effective means to keep track of the search space globally.

```
    definition(outlineBass,
        nScore rule:outlineBass(transition) size 20 then
        sortScore measure:property(linearProgression(bass) ).
```

Measurements are usually applied to a set of plausible solutions, where the set is sorted according to preferences before picking the desired solution (as in the example above).

This approach should be contrasted with the system described in [3], where a particular control strategy is hard-wired into the system. Both systems make an effective use of the same domain theory; however, explicit control structures allow us to experiment with different controls and reuse earlier work much more easily.

Related use of notions from meta-programming in the context of musical composition can be seen for example in the work of [1].

4.4 Flexibility in expressing control

Flexibility in expressing control is possible from two main factors: the granularity of the atomic primitives (i.e. rules, tests and measures) and the control primitives (i.e. the ability to filter, rearrange, truncate data stream, etc.). With a careful design of these components, we are able to control the problem solving process in an effective manner.

To illustrate our claim of flexibility and effectiveness in expressing control, let us consider a scenario. Given a melody to be harmonised, we may write a computer program to solve a harmonisation problem from the beginning of the piece to the end of the piece. In this manner, all the voices are determined and filled in from left to right. This was the approach taken in the *CHORAL* [3] and the *Harmonet* [5]. As a matter of fact, most systems commit their problem solving procedures under particular control structures inherent in their problem solving paradigm which are not easy to be altered/modified.

Our explicitly structured control approach offers great flexibility in composing different control structures. We list some options (applicable to the music domain below) below:

- The harmonisation is done from the start of the piece to the end of the piece.
- Harmonise the first phrase, then the last phrase, then the rest of the work, which means that each musical phrase can be dealt with in any order.

- Outline all the harmonic progressions of all phrases then determine the bass line before filling in other voices. This is the control structure given in section 4.1 above.
- In each phrase, work out the beginning of the phrase then the cadence. If the outline of the beginning and the ending of each phrase is satisfactory then proceed with the rest (see Figure 5,6).
- The control structure for each phrase could be structured differently. Different control structures applied to the same input melody may yield different good answers (see Figure 7).

The example below illustrates the harmonisation of phrase 5 of the chorales *O Gott, du frommer Gott* No. 312[1][10]. The choice of chorale is arbitrary, the choice of phrase is deliberate (i.e. it should be something in the middle of the chorale since harmonic structure in the middle of the chorale has more flexibility in terms of key). It does not have to end in the home key. We choose the middle part for this example since it is is freer in the harmonic structure. In the first example, the harmonic progression is determined by deciding on the start then the cadence and then the rest of the body. In the second example, the harmonic progression is determined by deciding on the cadence before the start and the body.

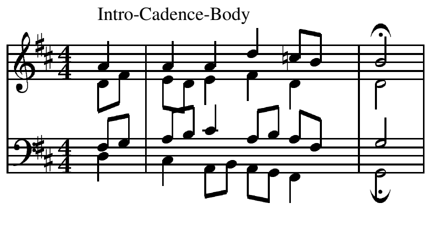


Figure 5: Control sequence: intro-cadence-body

In the above example (an intro-cadence-body order), the harmonic progression starts off in the key of D major before closing with a perfect cadence in the key of G major. In the second example below (a cadence-intro-body order), the harmonic progression starts off in the same D major key but closing in an imperfect cadence in the key of E minor.

The examples below are from the third phrase of the chorale 'Christus, der ist mein Leben' (R006). The cadence in the first example is a plagal cadence in the key of F major. The same phrase is reharmonised with a different control structure and results in a perfect cadence in the key of C major.

[1]the number referred to in this paper is the number of the chorales in the Riemenschneider's 371 Harmonised Chorales and 69 Chorales Melodies with Figured Bass

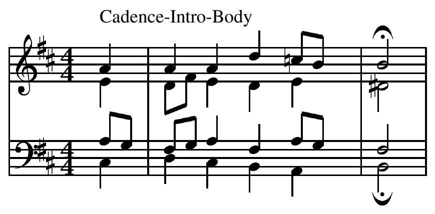


Figure 6: Control sequence: cadence-intro-body

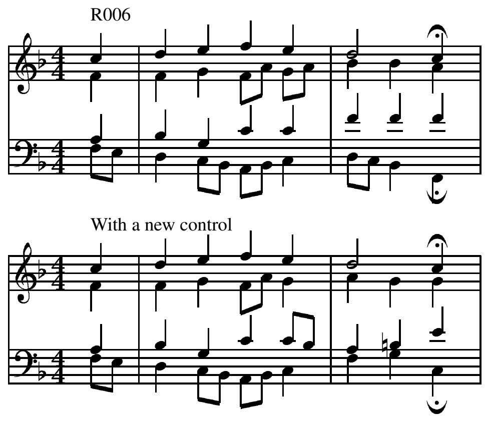
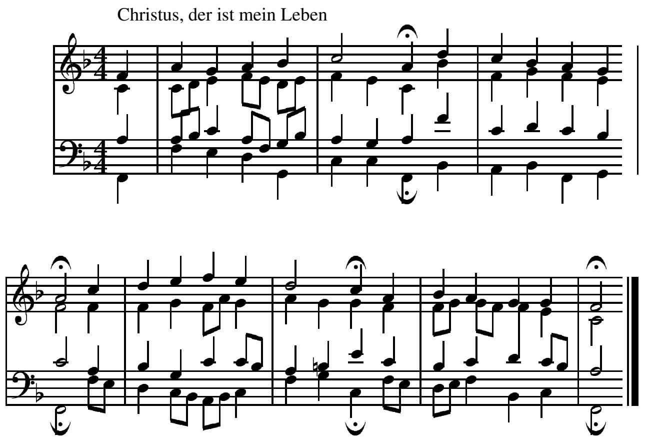


Figure 7: Applying different control structures

4.5 Limitations

We include a complete harmonisation of the chorale 'Christus, der ist mein Leben' (R006) here. The example is not flawless but we think the quality of the harmonised output is at an acceptable standard.



Computers sometimes generate very intelligent and convincing outputs, sometimes with acceptable outputs and sometimes with poor outputs that make us wonder what they are up to. We see inconsistencies in performance as a symptom of the lack of deeper knowledge and precise control. Most computers generate music suffers from this problem. At the current state, the harmonisation outputs are quite impressive and interesting. However, there are

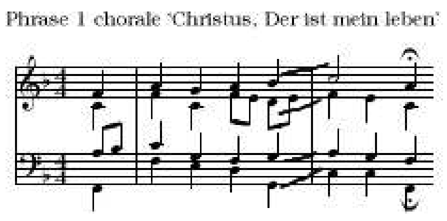


Figure 8: Ineffective cadential $^6_4$

still many places where the harmonisations are slightly dubious stylistically.

For example, in one of the outputs the experts suggest that a cadential $^6_4$ should not be arrived at by leaping all parts in the same direction and landing at the cadential $^6_4$ chord. Some auxiliary notes (e.g. neighbor notes, suspensions) are commented on as not appropriate or pointless decorations. We give three examples here. Figure 9 shows an example of an inappropriate use of a neighbor note and a proposed alternative. Figure 10 show examples of a not so effective usage of a suspension. Some auxiliary notes (e.g. neighbor notes, suspensions) are commented on as not appropriate or pointless decorations. We give three examples here. Figure 9 shows an example of an inappropriate use of a neighbor note and a proposed alternative. Figure 10 show examples of a not so effective usage of a suspension.

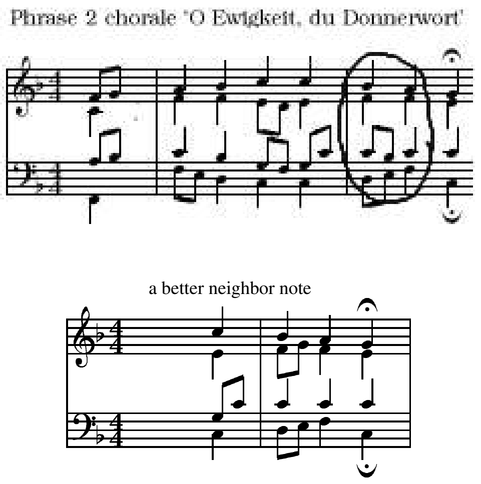


Figure 9: A poor neighbor note and an alternative (chorale R026)

The limitations listed here are hard problems (these points are also hard for human). As a matter of fact, all the outputs illustrated above are not wrong but can be improved on. Improvements are obtainable if the system has the appropriate knowledge and the knowledge is applied effectively (i.e. with appropriate control structures).

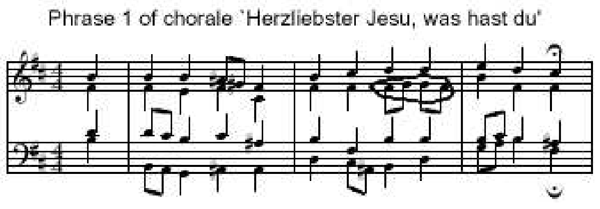


Figure 10: Poor suspension usage (chorale R078)

5 CONCLUSION

We view problem solving as a search problem. We have presented our argument in favour of a knowledge rich search control paradigm that makes control explicit. We have sketched our control language and illustrated its use in flexible reasoning over given domain knowledge. Our case study shows that an increased flexibility is achieved without engendering excessive computational overheads. The major benefit of this view is the flexibility in applying knowledge to guide search. Here, we have highlighted four main areas where knowledge can be applied in this framework.

Further work involves comparisons of the language with other meta-reasoning; and case studies in other domains.

References

[1] Carlos Agon, Gérard Assayag, Olivier Delerue, and Camilo Rueda. Objects, time and constraints in openmusic. In Ann Arbor, editor, *Proc. ICMC 98*, pages 267–274. Michigan, 1998.

[2] Saul Amarel. Problem solving. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence Vol II*, pages 1214–1229. Wiley-Interscience Publication, 1992.

[3] Kemal Ebcioglu. An expert system for harmonizing chorales in the style of J.S. Bach. *Journal of Logic Programming*, 8, 1990.

[4] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[5] H. Hild, J. Feulner, and W. Menzel. Harmonet: A neural net for harmonizing chorales in the style of J.S. Bach. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing 4*, pages 267–274. Morgan Kaufman, 1991.

[6] P. Jackson, H. Reichgelt, and Frank van Harmelen. *Logic-based knowledge representation*. The MIT Press, 1989.

[7] A. Newell, J. C. Shaw, and H. A. Simon. Report on a general problem-solving program for a computer. In *Proceedings of the International Conference on Information Processing*. UNESCO, Paris, 1960.

[8] S. Phon-Amnuaisuk. Control language for harmonisation process. In Christina Anagnostopoulou, Miguel Ferrand, and Alan Smaill, editors, *Music and Artificial Intelligence, Second International Conference, ICMAI 2002, Edinburgh, Scotland, UK, September 12-14, 2002, Proceedings*, volume 2445 of *Lecture Notes in Computer Science*. Springer, 2002.

[9] S. Phon-Amnuaisuk and G. Wiggins. The Four-Part Harmonisation Problem: A comparison between Genetic Algorithms and A Rule-based System. In Geraint Wiggins, editor, *Proceedings of The AISB'99 Symposium on Musical Creativity*, pages 28–34. AISB, 1999.

[10] Albert Riemenschneider. *371 Harmonized Chorales and 69 Chorale Melodies with Figured Bass*. G. Schirmer, Inc, 1941.

[11] L. Sterling and E. Shapiro. *The Art of Prolog*. MIT Press, 4th edition, 1994.