MusicSoar: Soar as an Architecture for Music Cognition

Don L. Scarborough, Peter Manolios and Jacqueline A. Jones

Department of Psychology and Department of Computer and Information Science
Brooklyn College of the City University of New York
Brooklyn, NY 11210
dosbc@cunyvm.bitnet, pete@sci.brooklyn.cuny.edu, jajbc@cunyvm.bitnet

Abstract

Newell (1990) argued that the time is ripe for unified theories of cognition that encompass the full scope of cognitive phenomena. Newell and his colleagues (Newell, 1990; Laird, Newell & Rosenbloom, 1987) have proposed Soar as a candidate theory. We are exploring the application of Soar to the domain of music cognition. MusicSoar is a theory of the cognitive processes in music perception. An important feature of MusicSoar is that it attempts to satisfy the real-time constraints of music perception within the Soar framework. If MusicSoar is a plausible model of music cognition, then it indicates that much of a listener's ability is based on a kind of memory-based reasoning involving pattern recognition and fast retrieval of information from memory: Soar's problem-solving methods of creating subgoals are too slow for routine perception, but they are involved in creating the knowledge in long-term memory that then can meet the processing demands of music in real time.

The Soar Architecture

This section is a brief introduction to Soar (Version 5) and is to a great extent based on Newell (1990). Soar is a goal-directed problem-solving cognitive architecture that is built on a parallel production system. Soar displays many of the characteristics of human cognition, and the temporal characteristics of Soar's cognitive behavior are consistent with much of what is known about human cognition.

Goals

A central premise of the Soar theory is that cognition is based on goal-directed problem solving. Soar's problem solving occurs within a *context* that has four predefined *attributes* or *slots*: a *goal*, a *problem space*, a *state*, and an *operator*. Goal-directed cognition begins with the selection of a goal followed by selection of a problem space which delimits the sets of states and of operators that will be considered. Next, a state is selected to represent the current state of the problem. Finally, an operator is selected to change the current state to reach the goal state. The Soar architecture can select a value for one of these context slots on each *decision cycle*.

Decisions about context slot values occur automatically within the Soar architecture if available knowledge is sufficient to guide the decision. Otherwise, an *impasse* occurs and the Soar architecture creates a subgoal which is a new context with the goal of resolving the impasse. This subgoal context then requires selection of a problem space, a state and one or more operators in order to resolve the impasse. Further impasses may occur within this subgoal, leading in turn to additional subgoals.

The initial top-level context is unique and is generally initialized with a predefined goal, problem space and state. Once the top context is initialized, an operator is selected to perform some task. Typically, Soar cannot implement this operator directly which causes an impasse, leading to the creation of a subgoal, as described above, to implement the desired operation.

Long term and working memories

Long term memory consists of productions that contain conditions and actions. If all of the conditions of a production match working memory elements, then the actions of that production fire, adding new information to working memory. Soar differs from conventional production languages, such as OPS5, in many important respects. For example, there is no conflict resolution; all productions that match fire in parallel, adding new and possibly conflicting information to working memory simultaneously. These additions to working memory may make it possible for new productions to fire. This process is called the *elaboration phase* of the decision cycle, and it continues until *quiescence*, when no new productions are triggered by information in working memory. The elaboration phase allows Soar access to all available, relevant knowledge for its decision making. At quiescence, Soar tries to select a value for a context slot. The elaboration phase followed by selection of a context-slot value is the decision cycle.

Working memory holds all the information about currently selected values for context slots. In addition, as long-term memory productions fire, they add to

working memory new information consisting of proposed values for context slots, preferences for previously proposed values, and augmentations of information already in working memory. All information in working memory is stored in a network that is linked to context slot values. If the value of a context slot changes, then all information linked to the old slot value is discarded. For example, when a subgoal resolves the impasse that created it, the subgoal context is deleted from working memory along with all information linked to that subgoal. Hence working memory is highly dynamic, allowing elements to disappear when no longer needed.

All I/O is mediated by the top context state. That is, in the Soar theory, perceptual input about the current state of the environment enters the top context state and this information can then be used by Soar in its decision making. Also, motor systems can access output commands that are placed in this top state.

Chunking

All learning in Soar occurs through *chunking*. When Soar finds a solution to an impasse, it creates a chunk, which is a new production that represents the solution to the impasse. The left-hand side of the new production contains the information in working memory that was available when the problem arose and that was used in finding the solution to the problem. The actions of the new production are the results of the problem solving. This production is added to long-term memory, and when Soar finds itself in a similar situation in the future, this production will fire and resolve the problem, thus eliminating the need to solve the problem again.

Mapping Soar to Human Cognition

Newell (1990) argues that the minimum functional neural circuit in the human brain takes about 10 ms to operate. Such a neural circuit can perform a function such as the memory access required to match the left-hand side of a production in long-term memory to conditions in working memory. This matching and firing of productions occurs in parallel in the elaboration phase of the decision cycle. The entire elaboration phase along with the subsequent decision phase takes place automatically in about 100 ms. The implementation of an operator will usually need a sequence of these decision cycles such that even the simplest cognitive tasks will require times on the order of about a sec.

MusicSoar

As noted above, cognition is based on goal-directed problem solving in the Soar theory. To apply Soar to music cognition, we must view listening to music as a form of problem solving. What problem confronts a person listening to music? It seems likely that any intelligent system should attempt to anticipate future events. In MusicSoar, we assume, following Narmour (1991), that the problem in listening to music is to anticipate what is to come, based on music that has already been heard. A listener's knowledge of a specific piece and its style, along with general musical knowledge, provide a basis for expectations of what is to come. If these expectations are accurate and match newly heard events, they become the basis for generating more expectations. On the other hand, expectations may not match what is heard. In our approach, this generates a subgoal to learn new expectations so that, if the same or similar music is heard again, the listener will be better able to anticipate the events that occur.

Listening to Music in MusicSoar

In MusicSoar, as musical events occur, they enter the top state. Musical "problem solving" begins in this top context with the selection of a *listen-to-music* operator. This immediately leads to a subgoal of implementing this operator, and, within this subgoal, MusicSoar creates a new state called *music-working-memory*. It is within this subgoal that MusicSoar listens to the music input and anticipates what will follow. Thus, MusicSoar has two primary states. The top state functions as a passive preattentive sensory input buffer like an "echoic memory," while the music-working-memory represents characteristics of the input that have been attended as well as expectations of what is to come.

Top State Input

Input notes appear in the top state state and disappear after some length of time. We assume that the representation of musical input in this top state is not in terms of waveforms, but rather is a representation of the output of earlier auditory perceptual preprocessing stages. Thus, in MusicSoar, each note in the top state is represented by its pitch and duration, as well as information about its temporal offset from the prior event. Input to the top state is handled by a Lisp function that reads a file containing symbolic representations of musical events. The input function creates an event attribute or augmentation linked to the top state for each new musical event. The value of an event augmentation is information about the event's temporal offset from the previous event as well as information about the pitch and duration of each note in the event. Currently, MusicSoar deals only with music containing a single voice, such as the melody of a folk song without accompaniment.

Music-Working-Memory

Once an intention of listening to music has been selected in the top context (as represented by the choice of the "listen-to-music" operator), MusicSoar creates a

listen-to-music subgoal. This subgoal then persists throughout the piece of music. The problem space in this subgoal represents MusicSoar's knowledge of music. The state associated with this first subgoal is called music-working-memory and contains information about top-state musical events that have been attended and processed. The initial representation of top-state events in music-working-memory encodes only some of the simple relational properties such as whether the most recent note is higher, lower or the same in pitch as the previous note, as well as whether the offset of the newest event is the same or longer or shorter than the previous offset. More complex encodings within music-working-memory, such as information about specific pitch intervals, depend upon additional processing. It is also within music-working-memory that anticipations arise. There are only two operators that can be selected within this listen-to-music subgoal context: *attend*, and *learn-expectation*. If a new event occurs in the top state and no operator has been selected, then the attend operator is proposed. Once the attend operator is selected, it is implemented by productions that copy part or all of the musical event information from the top state into the music-working-memory. The productions that implement the attend operator can operate without additional subgoals in a single decision cycle. Additional productions compare the attended event to expectations and, if it matches, the new event is added to a linked list of previously heard events in music-working-memory. Because the attend operator is implemented by productions that fire without requiring decisions about context slots, the attend operator requires a single decision cycle of about 100 ms. Thus, when music conforms to expectations, the attend operator can follow along at about 10 events per second. On the other hand, the learn-expectation operator is proposed whenever an attended event does not match expectations. This operator leads to a subgoal to learn new expectations that match what actually happened.

Expectations

Musical expectations in MusicSoar are stored in its long-term production memory. The left-hand sides of these productions specify particular patterns of events in music-working-memory, while the right-hand sides represent expectations of what should follow the occurrence of these patterns. If the left-hand side of a production matches the previously heard events in music-working-memory, the production fires and adds its expectation to working memory. Generating expectations in this way involves no decision making and thus can occur quickly. Newell (1990) has argued that retrieving information from memory (e.g., matching the conditions of a production to working memory elements) requires about 10 ms. Initially, MusicSoar has only a few default expectations, such as to expect that the next event will have the same properties, e.g. pitch, and offset, as the previous event. As MusicSoar experiences different musical patterns, it learns new productions that are added to long-term production memory.

Learning

Subgoals in Soar arise when it is unclear what to do next, and they result in decisions about what to do. Upon resolving a subgoal, Soar can learn new productions or chunks that store information about how the problem was resolved. The next time that problem arises, the resolution of the problem can be retrieved from long-term production memory without requiring problem solving again. Thus, after experience with a particular problem, Soar can subsequently solve the problem using a form of memory-based reasoning in which the old solution is retrieved from memory rather than solving the problem again from scratch. MusicSoar's expectations arise from learning in the learn-expectation subgoal.

The way in which the learn-expectation subgoal is instantiated depends upon the type of expectation mismatch that occurred and the subgoal problem space that is selected, e.g., metric, rhythmic, melodic, etc. For example, if a new event occurs at a time that is inconsistent with metric expectations, this requires reinterpreting the meter of the music; however, an unexpected event that occurs at a time that is consistent with metric expectations requires learning a rhythmic expectation. Problem solving within the learn-expectation subgoal can occur in several ways. For example, if a note occurs later than expected, MusicSoar can try look-ahead search to see if the expected note appears to lead, in terms of offset, duration and pitch, to the new note. In this case, the expected note may bridge the gap from the previous events to the event just heard. Alternatively, additional musical analysis may reveal new features of the music that has been heard, allowing other productions in long-term memory to fire, and these productions may propose the correct expectation. For example, MusicSoar might look back at previously heard events in working memory to see if some parallel sequence of events has occurred. Finally, the learn-expectation subgoal might use *data chunking* (Rosenbloom, Laird & Newell, 1987) to learn to expect the new event based on the immediately preceding events. In data chunking, some of the previously heard events are learned as a cue that will, in the future, trigger recall of the newly heard event in a sort of paired associate learning. That is, some of the previous events in music-working-memory become the pattern for the left- hand side of a new production that will trigger a new expectation. This lets MusicSoar memorize

specific songs. Although MusicSoar can learn expectations corresponding to specific pieces, this is not sufficient. People not only acquire specific knowledge about particular pieces of music, but they also acquire more general schematic knowledge that guides expectations when listening to new pieces of music (Narmour, 1991). For example, experience with Western tonal music leads people to expect particular melodic, harmonic and rhythmic progressions. Thus, a general problem for MusicSoar is to induce schematic musical knowledge and expectations from experience with specific pieces.

The representation of musical input has important effects on learning. People generally perceive events (visual, auditory, tactile, etc.) in terms of the relational properties of the event; e.g., they perceive relative luminance differences in vision rather than absolute luminance levels. In music, the salient perceptual properties involve relative pitch and time differences. To reflect this, the top state representation of musical events is encoded in terms of the pitch and duration of each event relative to previous events. Because MusicSoar learns expectations in terms of these relational properties, this learning will generalize to situations that preserve these relations. Thus, if MusicSoar learns to expect the next note in an arpeggio, this learning will apply to any arpeggio in the same mode (e.g. major or minor) regardless of the key. Thus, learning based on such relational information will generalize directly to transpositions in pitch and time. However, one issue that we have not yet resolved in the data chunking mechanism described above is how much information about previous events should be included in the left-hand side of new productions. Making the left-hand side too specific will prevent the new productions from generalizing to any other situation.

As just indicated, MusicSoar's ability to generalize is influenced by the representation of the music in music-working-memory. A key question for any inductive learning is what knowledge and learning biases exist before the learning actually begins (Dieterich, 1990). For example, we have assumed that human listeners have some understanding of pitch relations without any training. That is, if two sounds differ in frequency, listeners hear the higher frequency as higher in pitch. More complex characterizations of pitch, such as octave relations and perception of consonance may be learned early in life based on auditory stimuli in general (e.g. Terhardt 1991), and thus may be available almost from the very first musical experience. Some primitive temporal knowledge, such as the ability to hear differences in durations, also seems almost certainly innate or at least acquired at a very early age. These considerations determine the design of MusicSoar. We have assumed that, without prior experience, MusicSoar can determine the contour of a melody, i.e., the pattern of ups and downs in pitch. However, we assume that more specific knowledge of pitch relations requires learning about particular intervals. Further assumptions are that MusicSoar can perceive equality of time intervals, and time ratios of two to one and three to one (corresponding to duple and triple meters in music, respectively).

One interesting problem is that Soar has no explicit forgetting mechanism. That is, once a new production is learned, it is never forgotten. However, later learning will create productions that may interfere with older learning. That is, Soar can demonstrate retroactive interference wherein a new production may encode a new and different expectation for a pattern that is similar to the left-hand side patterns of already learned productions. When that pattern occurs, both productions may fire and generate different expectations. One general problem in MusicSoar is how to handle such expectation conflicts.

Knowledge Search

A listener can have both specific and schematic knowledge that is relevant to a particular listening experience. Given this, hearing a particular event sequence may trigger many expectations, some of which may conflict. We have considered and rejected two possibilities for handling such conflicts. First, we might let all expectations be added in parallel to working memory without differentiation. However, this is unacceptable, because if listeners know a particular song, they have specific expectations about what should come next, and they generally will not wander off the track, even though fragments and aspects of the song may have occurred in other previously heard pieces of music. For example, given familiarity with Beethoven's Fifth Symphony, there is no ambiguity about what follows "dit-dit-dit-dah." Thus, we cannot just let all possible expectations based on prior experience have equal status. A second possibility is to use different problem spaces for different pieces of music, e.g. a listen-to-Mozart's-40th and a listen-to-Beethoven's-fifth problem space. The learning that occurred within a problem space would be available only within that problem space. But this also cannot be the right solution, because it is clear that, while listening to Beethoven's Fifth, we do not cut ourselves off from all other musical knowledge. Separate problem spaces would also prevent generalization. What was learned for one piece would be available only in the problem space for that piece. With radically different types of music like Indian ragas and Western tonal music, the expectations and thus the problem spaces are likely quite different, but, given a particular genre or style of music, it seems probable that particular pieces of music within that style are heard within a problem space that is common to that style.

If a listener wants to follow along with Beethoven's Fifth, it must be possible to select quickly the appropriate specific expectations from a much larger set of expectations drawn from general musical knowledge. We are exploring two ways to limit choices in MusicSoar. First, working memory can contain specific elements that can control which expectations are activated. Thus, if music-working-memory contains the information that we are listening to Beethoven's Fifth, then previously learned productions for Beethoven's Fifth can be activated, but not, say, productions specific to Mozart's 40th symphony, because information about the identity of the piece would be included in the left-hand side of the expectation productions for Beethoven's Fifth. This is similar to the idea of setting up different problem spaces for different pieces but is less restrictive. A second related approach is to have production memory contain productions that express preferences for particular expectations. That is, when listening to Beethoven's Fifth, all productions that match a musical fragment may fire, resulting in a rich flood of expectations. However, other productions may also fire that express preferences for those expectations that are linked to MusicSoar's goals. Thus, if the goal is to follow along with Beethoven's Fifth, preferences for expectations specific to Beethoven's Fifth would be activated, keeping MusicSoar's expectations on track.

Temporal Constraints on Processing

Newell has argued that if the human brain implements a Soar-like architecture, a decision to select a value for a context slot (i.e. goal, problem space, state or operator) must take a decision cycle of approximately 100 ms (Newell, 1990). Music unfolds at rates that are not controlled by the listener, and the listener must cope with events as they occur. Thus, we can ask whether MusicSoar, when limited by a 100 ms decision cycle, can meet the real-time demands that are imposed by music. The attend operator in MusicSoar can be selected and implemented in a single decision cycle. This means that MusicSoar can attend to up to 10 events per second if the attend operator is selected repeatedly and without interruption. However, the attend operator for the next input event will be selected only if the current expectations match the most recently attended event. If the generation and selection of expectations were to also require selecting and implementing operators at a rate limited by a 100 ms decision cycle, MusicSoar would only be able to keep up with the slowest music. Thus, expectations must generally arise based on recognition of patterns in music-working-memory that match the left-hand-side of expectation generating productions already available in long-term production memory. As noted above, Newell (1990) argues that it is plausible to assume a memory access time of about 10 ms for production matching. In addition, preferences that select among various expectations must act on music-working-memory directly and cannot require decisions about goals, problem spaces, states or operators. They too can be implemented as quickly as they can be retrieved from production memory.

On the other hand, if expectations do not match the music, MusicSoar enters a learn-expectation subgoal which requires several decision cycles. Given that MusicSoar may be unable to complete this subgoal processing in time to attend to the next event, what happens? One possibility is that the next input event generates an interrupt, forcing MusicSoar to select a new attend operator over any currently selected operator. This would mean that MusicSoar would lose any current "thinking" about the last event. Another possibility is that MusicSoar might continue to think on the basis of the previously attended top state events. But if this occurs, MusicSoar may fall behind, perhaps by several events, which can cause MusicSoar to miss notes, and lead to incomplete processing and representation of the music. Then when MusicSoar completes its thinking and is ready to attend again, what should it attend to?

The temporal processing constraints in MusicSoar may also suggest why one can hear the same piece of music many times and continue to hear new things. A listener may not perceive all aspects of a complex musical event. In MusicSoar, this means that the representation of an event in the top state may be comprehensive, but the attend operator may encode only some of the top state information into working memory. The subset of information that is encoded will then determine the expectations that follow. If a different subset of information is encoded upon later rehearing, then a different set of expectations will arise. Another source of variability in MusicSoar can arise in the learning process. A new event may simultaneously mismatch expectations in several ways, e.g. metric, melodic, harmonic, etc., various musical characteristics which are perceived somewhat independently (Palmer & Krumhansl, 1987). MusicSoar breaks the problem down and tries to learn expectations for these characteristics, each of which is handled within a different subgoal problem space. Because each subgoal usually requires several decision cycles of about 100 ms each, only rarely is there sufficient time to deal with all the characteristics. In a single hearing, MusicSoar can learn only some of the characteristics of the music, and later rehearings then provide opportunities for additional learning. This account suggests why, when a piece is partially learned, a person may be able to imagine hearing it by following the sequence of expectations that are generated. However, any attempt to sing it out

loud would be embarrassing as the expectations do not completely specify all the characteristics required for performance.

Hierarchical structure of music

Lerdahl & Jackendoff (1983) argue that perception of music involves creating a subjective hierarchical structure involving meter, rhythm, grouping, tonal movement, etc. Such structures arise in MusicSoar from augmentations added to attended events in the listen-to-music subgoal. The productions that create these augmentations are learned in meter-analysis, grouping-analysis, and tonal-analysis subgoals that arise within the learn-expectation subgoal. The ability of MusicSoar to create such structures depends heavily on past experience, for there is insufficient time for extensive analysis of the music in subgoals. Rather, such structures must generally arise directly from previously learned productions that match the information in working memory.

Composition

One interesting property of the MusicSoar approach is that this system could, with few changes, compose music. That is, because MusicSoar is based on expectations, all that is required is to initialize music-working-memory with a musical fragment. This fragment would then trigger expectations, and these expectations would not be compared to actual inputs but would simply become new values in music-working-memory which are then used to control the generation of still more expectations. The quality of the composition represented by this chain of expectations would be governed by the quality of the learned expectations. However, this perspective brings another point into focus: When there are conflicting expectations, the system must show some indeterminacy. That is, given a set of conflicting expectations, the same one should not be selected on every run or else we have a system that can compose only a few pieces. It is likely that the same indeterminacy should also be true of listening: The same song may be re-heard in different ways.

Summary

There has been relatively little work on building processing theories of music. Soar provides a challenging and exciting framework for such explorations, and, to our knowledge, it has not previously been applied to music cognition. Soar poses several interesting constraints in its application to music cognition. One is that music cognition must be viewed as a problem solving activity. Intuitively, we think it is reasonable to view the problem in music cognition as one of anticipation. A second significant constraint on the application of Soar to music cognition is that Newell has linked the time that is required for a decision cycle in Soar to brain processes that require about 100 ms. This temporal constraint imposes important limitations on how the theory can be applied to a domain such as music and makes it possible to evaluate the Soar theory and its instantiation in MusicSoar in ways that are rarely true of cognitive theories. We think that the Soar framework may generate new insights and questions into the problems that are posed for cognitive theories for domains such as music. MusicSoar is an attempt in this direction.

References

Dieterich, T. G. 1990. Machine Learning. *Annual Review of Computer Science* 4:255-306. Palo Alto, CA: Annual Reviews Inc.

Laird, J., Newell, A., & Rosenbloom, P. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence* 33:1-64.

Lerdahl, F., & Jackendoff, R. 1983. *A Generative Theory of Tonal Music*. Cambridge, MA: MIT Press.

Narmour, Eugene. 1991. The Top-down and Bottom-up Systems of Musical Implication: Building on Meyer's Theory of Emotional Syntax. *Music Perception* 9:1-26.

Newell, Allen. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Palmer, C., & Krumhansl, C. 1987. Independent Temporal and Pitch Structures in Perception of Musical Phrases. *Journal of Experimental Psychology: Human Perception and Performance* 13:116-126.

Rosenbloom, P., Laird, J., & Newell, A. 1987. Knowledge Level Learning in Soar. In *Proceedings of AAAI-87: Sixth National Conference on Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann. pp. 499-504.

Terhardt, E. 1991. Music Perception and Sensory Information Acquisition: Relationships and Low-level Analogies. *Music Perception* 8:217-240.