Robotic Sailing and Automatic Music Generation/Learning

Jon Sneyers
KU Leuven
Department of Computer Science
Declarative Languages & Artificial Intelligence

March 2014
Robotic Sailing

- ASV Roboat: a project of the Austrian INNOC research team since 2006
Autonomous robotic sailing

- Autonomous in two senses:
  - No humans needed
  - Energy self-sufficient
- Sailing is very energy-efficient (compared to using a motor)...
- ... but it is not easy!
(Potential) Applications

- Cost-effective data collection for marine research
  - Static sensor buoys have limited range and need considerable maintenance
  - Manned research vessels...
    - are very expensive
    - can make too much noise

- Surveillance, e.g. to detect smuggler boats
- CO2-neutral transportation of goods
- Safety/comfort on human-operated sailing boats
Roboat

- Based on Laerling sailboat
  - 3.75m long, 300kg, 4.5m$^2$ sail area
  - Originally designed for kids to learn sailing
  - Virtually unsinkable :)
- Add a battery bank, 285Wp solar panels (and a 65W backup methanol fuel cell just in case)
- Add sensors (GPS, compass, radar, anemometer, etc.), communication systems (WLAN, UMTS/GPRS, satellite), actuators (sails, rig) and an onboard PC
Roboat system architecture

- Layered design:

```
Operator
  ↓
Abstractor
  ↓
  |     |     |     |
  ↓     ↓     ↓     ↓
  Strategic Long Term Routing
  ↓
  Short Course Routing
  ↓
  Manoeuvre Execution
  ↓
  Emergency Reflexes
  ↓
Abstractor
  ↓
Actuators
```
Shortest path problem

- Easy for cars on road maps:
  - sparse weighted graphs
  - Dijkstra's algorithm with an efficient priority queue (e.g. Fibonacci heaps)
  - $O(n \log n)$ if there are $n$ nodes (road crossings)
- Not so easy for sailboats on the open sea
Long Term Routing

- Polar diagram
  - Boat speed (relative to water) is related to wind speed and angle
  - Sailing directly against the wind is not possible
  - Tacking (zig-zag manoeuvres) → over long enough distances, the convex hull of the polar diagram can be used
  - Max speed (“hull speed”)
Boat speed

- Depends on wind vectors
- Depends on surface sea currents
- Wind and currents change over time
- Local measurements are OK for manoeuvres and short-term routing
- For long-term routing we have to rely on weather forecasts
Discretization

- Sea is continuous, we discretize to a grid (resolution: ~10 to 1000m)
- Implicit graph
- Nodes also have a timestamp (resolution: same as weather forecast temporal resolution, e.g. 3h or 30 mins)
Search

- A* search (best-first)
- Heuristic: straight-line distance at hull speed
  - Can be improved by taking static obstacles into account

(using a pre-processing step that does not take wind/currents into account)
Demo
Automatic POP Composer
And LEArner of ParameterS

APOPCALEAPS
Jon Sneyers and Danny De Schreye. APOPCALEAPS: Automatic Music Generation with CHRI SM. 22nd Benelux Conference on Artificial Intelligence (BNAIC'10), Luxembourg, October 2010.
Overview

APOPCALEAPS

GUI

query

manual parameter tuning

Probability parameters

Learning algorithm (PRISM)

CHiRiSM program

CHiRiSM observation

LilyPond file

GNU LilyPond

Output

Music (MIDI file)

Score (PDF file)

Training set

quality evaluation (selection)

Human

Human
Overview

Composer
(music synthesis)

Learner
(music analysis)
CHance Rules induce Statistical Models

CHRiSM
CHRiSM

- CHRiSM is based on CHR(PRISM)


- PRISM is essentially Prolog + probabilities + built-in general-purpose EM-learning algorithm

- CHR is a declarative rule-based language extension (available for various host languages like Prolog, Haskell, Java, C, ...)
Master thesis students have been using, extending and improving APOPCALEAPS:

- Kristof Helsen: “De automatische componist” (2010-2011)
Input constraints

% inputs

:- chrism  measures(+int), meter(+int,+duration),
key(+key),  tempo(+int),  voice(+voice),
shortest_duration(+voice, duration),  range(+voice,
+note,+int,+note,+int),  max_jump(+voice,+int),
instrument(+voice,+),...

:- chr_type  key ---> major; minor.

:- chr_type  voice ---> melody; chords; bass; drums.

:- chr_type  note ---> c; d; e; f; g; a; b; cis;
dis; fis; gis; ais; r.

:- chr_type  duration ---> 2; 4; 8; 16; 32.
Example query

\begin{verbatim}
meter(2,4), tempo(100), key(major),
% 2/4 time signature, 100 beats per minute, major key
voice(bass), voice(melody),
% two voices: a bass and a melody
range(bass,g,1,c,3), range(melody,g,3,e,5),
% bass ranges from g1 to c3, melody from g3 to e5
instrument(bass,'contrabass'), instrument(melody,'soprano sax'),
% MIDI instruments used to render the voices
max_jump(bass,12), max_jump(melody,5),
% maximal interval between consecutive notes in semitones
shortest_duration(bass,8), shortest_duration(melody,16),
% shortest possible bass note is an eighth, for melody a sixteenth
measures(8)
% generate 8 measures
\end{verbatim}
Output constraints

% outputs
:- chrism
  mchord(+measure,+chord),
  beat(+voice,+measure,+int,+float,+duration),
% beat(V,M,B,P,D): for voice V, a note starts at measure M, beat B, position P, with duration D
  note(+voice,+measure,+int,+float,+note),
  octave(+voice,+measure,+int,+float,+),
% note and octave at that measure-beat-position
  tied(+voice,+measure,+int,+float).
% the note at this position is tied to the next note
Rules for chords

- One chord per measure (for simplicity)
- First and last measure chords correspond to key:
  - key(major) ==> mchord(1,c).
  - key(major), measures(N) ==> mchord(N,c).
  - key(minor) ==> mchord(1,am).
  - key(minor), measures(N) ==> mchord(N,am).
- Other measure chords are assigned:
  \[ \text{mchord}(A,\text{Chord}), \text{next_measure}(A,B), \text{measures}(M) \]
  \[ ==> B < M \mid \]
  \[ msw(\text{chord_choice}(\text{Chord}), \text{NextChord}), \]
  \[ \text{mchord}(B, \text{NextChord}) \]
Default chord transition probabilities

values(chord_choice(_), [c, g, f, am, em, dm]).
:- set_sw(chord_choice(c), [0.2, 0.3, 0.25, 0.15, 0.05, 0.05]).
:- set_sw(chord_choice(g), [0.3, 0.15, 0.2, 0.2, 0.05, 0.1]).
:- set_sw(chord_choice(f), [0.25, 0.3, 0.1, 0.15, 0.05, 0.15]).
:- set_sw(chord_choice(am), [0.1, 0.2, 0.35, 0.1, 0.05, 0.2]).
:- set_sw(chord_choice(em), [0.05, 0.2, 0.3, 0.35, 0.05, 0.05]).
:- set_sw(chord_choice(dm), [0.05, 0.2, 0.35, 0.2, 0.05, 0.15]).

Markov Chain of order 1
(straightforward to extend to higher order or HMM)
Rhythm: “Beat splitting”
Rhythm: “Beat splitting”

% make initial beats (one beat per count)
meter(N,D), voice(V), measure(M) ==> make_beats(N,D,M,V).
make_beats(N,D,M,V) <= N > 0 |
  N1 is N-1, next_beat(V,M,N1,0,M,N,0),
  beat(V,M,N1,0,D), make_beats(N1,D,M,V).

% split some of the beats in two
split_beat(V) ??
meter(_,OD), shortest_duration(V,SD)
\ beat(V,M,N,X,D), next_beat(V,M,N,X,NM,NN,NX)
<= D<SD |
  D2 is D*2, X2 is X+1/(D2/OD),
  next_beat(V,M,N,X,M,N,X2), next_beat(V,M,N,X2,NM,NN,NX),
  beat(V,M,N,X,D2), beat(V,M,N,X2,D2).
Abstract beat positions

% abstract_beat(+meter1, +Beat, +SubBeat, -AbstractBeat)
abstract_beat(_,0,0,first) :- !.

% binary meter: middle is strong
abstract_beat(M,N,0,strong) :- 0 is M mod 2, N is M//2, !.
abstract_beat(3,_,0,strong) :- !.
abstract_beat(2,_,0.75,prestrong) :- !.
abstract_beat(3,2,0.5,prestrong) :- !.
abstract_beat(4,1,0.5,prestrong) :- !.
abstract_beat(N,M,0.5,prestrong) :- N =:= M+1, !.
abstract_beat(2,0,0.5,weak) :- !.
abstract_beat(2,1,0.5,weak) :- !.
abstract_beat(3,_,0.5,weak) :- !.
abstract_beat(_,_,0,weak) :- !.
abstract_beat(_,_,_,weakest).
Melody Notes

\[ \text{beat}(V, M, N, X, D), \quad \text{mchord}(M, C), \quad \text{next}\_\text{beat}(V, M_1, N_1, X_1, M, N, X), \quad \text{octave}(V, M_1, N_1, X_1, 00) \]
\[\implies V \equiv \text{drums}, \ V \equiv \text{chords} \]
\[\quad | \quad \text{abstract}\_\text{beat}(M, N, X, AB), \quad \text{soft}\_\text{msw}((\text{note}\_\text{choice}(V, C, AB), \text{Note}), \quad \text{note}(V, M, N, X, \text{Note}), \quad \text{(Note} = r \rightarrow \text{octave}\_d(V, M, N, X, 0) \quad ; \quad \text{find}\_\text{octave}\_d(V, M, N, X, 00) ) \). \]
Melody Notes

\[
\begin{align*}
\text{beat} & (V, M, N, X, D), \\
\text{mchord} & (M, C), \\
\text{next\_beat} & (V, M1, N1, X1, M, N, X), \\
\text{octave} & (V, M1, N1, X1, OO) \\
& \rightarrow \\
V & \Leftarrow \text{drums}, \ V \Leftarrow \text{chords} \\
& \mid \\
\text{abstract\_beat} & (M, N, X, AB), \\
\text{soft\_msw} & (\text{note\_choice}(V, C, AB), \text{Note}), \\
\text{note} & (V, M, N, X, \text{Note}), \\
( \text{Note} & = r \rightarrow \text{octave\_d}(V, M, N, X, 0) \\
& ; \text{find\_octave\_d}(V, M, N, X, OO) ).
\end{align*}
\]
Melody Notes

\begin{align*}
\text{beat}(V, M, N, X, D), \quad & mchord(M, C), \quad \text{next\textunderscore beat}(V, M_1, N_1, X_1, M, N, X), \\
& \text{octave}(V, M_1, N_1, X_1, OO) \\
\Rightarrow & V \rightarrow \text{drums}, \ V \rightarrow \text{chords} \\
& | \\
& \text{abstract\textunderscore beat}(M, N, X, AB), \\
& \textbf{soft\textunderscore msw}(\text{note\textunderscore choice}(V, C, AB), \text{Note}), \\
& \text{note}(V, M, N, X, \text{Note}), \\
& (\ \text{Note} \rightarrow r \rightarrow \text{octave\textunderscore d}(V, M, N, X, 0) \\
& \quad ; \ \text{find\textunderscore octave\textunderscore d}(V, M, N, X, 0)) .
\end{align*}

\textbf{Soft\textunderscore msw} ?

This is a variant of msw that allows backtracking (even during sampling).
Constraints

- For example, maximum interval between two consecutive notes:

\[ \text{max\_jump}(V, M\text{Int}), \ \text{octave}(V, M_1, N_1, X_1, 00), \ \text{note}(V, M_1, N_1, X_1, ON), \ \text{note}(V, M, N, X, NN), \ \text{next\_beat}(V, M_1, N_1, X_1, M, N, X), \ \text{octave}(V, M, N, X, NO) \implies \]
\[ \text{interval}(ON, OO, NN, NO, Interval), \ Interval > Mint \]
\[ | \]
\[ \text{fail}. \]
Rhythm (2): note joining

\[
\text{join_notes}(V, \text{cond } M=M2, \text{cond } N=N2) \quad ??
\]
\[
\text{note}(V, M, N, X, \text{Note}), \quad \text{note}(V, M2, N2, X2, \text{Note}),
\]
\[
\text{next\_beat}(V, M, N, X, M2, N2, X2)
\]
\[
==> \ V \ \underline{\text{\textit{\ equals}}} \ \text{drums} \quad | \quad \text{tied}(V, M, N, X).
\]
Full program

- About 50 rules (no time to show them all)
  - Plus some auxiliary rules and predicates
    - For writing the output LilyPond file
    - For more efficient learning
    - To compute intervals, abstract beats, etc.
- All in all about 500 lines of code → very small!
- 107 probability distributions (8 parametrized experiments), 324 parameters to be learned
Demo

Robotic Sailing and Automatic Music Generation/Learning – Jon Sneyers
Other potential uses

- Current system: generation and learning
- Other tasks can also be done (with the same underlying program!)
  - Classification
  - Improvisation
  - Automatic accompaniment
  - Complete an unfinished composition
  - ...
Classification

- Preparation steps:
  - Learn from training set A (e.g. *The Beatles*) → model $M_A$
  - Learn from training set B (e.g. *The Rolling Stones*) → model $M_B$

- Classification:
  - Given an unknown song, compute its probability in $M_A$ and in $M_B$
  - Classify accordingly (highest probability wins)

- Categories can be anything (style, composer, …)
Improvisation, Accompaniment

- Given a partial song (bass, drums, chords), generate a melody that “fits”
  - Put the known voices in the query, the rest will be generated
- Given a song, find underlying chord sequence
  - Compute Viterbi explanation of a partial observation consisting of everything but the chords
Thank you!

Questions?
Extra slides

- Here are some extra slides to explain CHR and CHRIISM
Syntax of CHR

- Propagation rule:

  head \implies guard \mid body.

Example: \texttt{dist}(A,D), \texttt{road}(A,B,L) \implies \texttt{dist}(B,D+L).

“If A is within distance D, and there is a road from A to B of length L, then B is within distance D+L.”

- Simplification rule:

  head \iff guard \mid body.

Example: \texttt{dist}(A,X), \texttt{dist}(A,Y) \iff X \leq Y \mid \texttt{dist}(A,X).

“If A is within distance X and also within distance Y, where X \leq Y, then this is equivalent to A being within distance X.”
Operational semantics of CHR

**IF head IN STORE (AND guard HOLDS), THEN...**

- **Propagation rule:** … ADD body TO STORE
  - Keep head, add body \[\text{head} \Rightarrow \text{guard} | \text{body}.\]

- **Simplification rule:** … REPLACE head BY body
  - Remove head, add body \[\text{head} \Leftrightarrow \text{guard} | \text{body}.\]

- “Simpagation” rule combines the above:
  - Remove removed_head, keep kept_head, add body
    \[\text{kept\_head} \setminus \text{removed\_head} \Leftrightarrow \text{guard} | \text{body}.\]
vegetable
red

vegetable
blue

propagation
rules

simplification
rule

CHR program

clouds \implies \text{forecast(rainy)}.

\text{forecast(rainy)} \implies \text{bring(coat)}.

\text{forecast(sunny)} \implies \text{bring(sunscreen)}.

\text{bring(coat)}, \text{bring(sunscreen)} \iff \text{bring(umbrella)}.
Chance rules

- A chance rule is a CHR rule with a probability:
  \[
  \text{prob} \ ?\ k_{\text{head}} \ \text{\backslash} \ r_{\text{head}} \Leftrightarrow \text{guard} \ | \ \text{body}.
  \]

- For example:
  \[
  0.4 \ ?\ \text{flu}(X), \ \text{friend}(X,Y) \Rightarrow \text{flu}(Y).
  \]
  
  "If X has the flu, and Y is a friend of X, then there is a 40% chance that Y also gets the flu."

- Every result from a query has some probability:
  \[
  Q = \{ \ \text{flu}(\text{jon}), \ \text{friend}(\text{jon,thom}), \ \text{friend}(\text{thom,slim}) \ \}
  \]
  
  - \( Q \rightarrow Q \) \hspace{1cm} 60\% \text{ chance } (1 - 0.4)
  - \( Q \rightarrow Q, \ \text{flu}(\text{thom}) \) \hspace{1cm} 24\% \text{ chance } (0.4 \times 0.6)
  - \( Q \rightarrow Q, \ \text{flu}(\text{thom}), \ \text{flu}(\text{slim}) \) \hspace{1cm} 16\% \text{ chance } (0.4 \times 0.4)
Chance rules (2)

- The body of a chance rule can contain probabilistic disjunctions

- For example:

  \[
  \text{coin\_flip} \leftrightarrow \text{head}:0.5 \ ; \ \text{tail}:0.5. \\
  \text{(you can also write “coin\_flip} \leftrightarrow \text{?? head} \ ; \ \text{tail.”)}
  \]

- One of the disjuncts is chosen at random (no backtracking)
PRISM

- CHRiSM is implemented in CHR(PRISM)
- PRISM is a probabilistic extension of Prolog
- PRISM builtin: `msw(Experiment,Outcome)`
  - Define outcomes with `values/2`
  - Define probabilities with `set_sw/2`
- For example:
  ```prolog
  values(coin,[head,tail]).
  :- set_sw(coin, [0.5, 0.5]).
  coin_flip(Result) :- msw(coin,Result).
  ```
Sampling

- If you just execute a query, the random choices are made according to the probability distributions, so you are **sampling** the statistical model represented by the program.

- For example:

  ```prolog
  ?- sample coin_flip
  tail
  ?- sample coin_flip, coin_flip
  head, tail
  
  The **sample** keyword is optional.
Observations

- One sample gives a full observation, denoted with \( \text{===>} \) (do not confuse with \( \leftrightarrow \))
  
  \[
  \text{coin\_flip} \leftrightarrow \text{tail} \\
  \text{coin\_flip, coin\_flip} \leftrightarrow \text{head, tail}
  \]

- If we can see only a part of the result, we have a partial observation, denoted with \( \text{====>} \)
  
  \[
  \text{flu(jon), friend(jon,thom), friend(thom,slim)} \text{====> flu(slim)}
  \]
Probability calculation

- The probability of an observation is the chance that a random sample agrees with the observation.

?- prob coin_flip <=> head
The probability is 0.5

?- prob coin_flip, coin_flip ==> head
The probability is 0.75

?- prob flu(jon), friend(jon,thom), friend(thom,slim) ==> flu(thom)
The probability is 0.4
Unknown probabilities

- If the probabilities are not given as a number, they are unknown
- Initially the probabilities are set to the uniform distribution
- They can be *learned* from observations
- For example, a biased coin with unknown bias:
  
  ```
  bias_flip <=> my_coin ?? head ; tail.
  ```
Learning

- Given a list of observations, learning will try to set the probabilities to maximize the likelihood of the observations

```prolog
?- learn([(bias_flip <=> head),
           (bias_flip <=> head),
           (bias_flip <=> tail)]),
       show_sw.
```

Switch choice my_coin:

1 (p: 0.66667)  2 (p: 0.33333)