Caracterización del Rendimiento de Librerías para Programación por Restricciones para Problemas de Satisfacción de Restricciones en Armonia Musical

Autor:

Steven Augusto Villegas Castellanos[1]

Director:

David Alejandro Przybilla
Ingeniero de Sistemas

UNIVERSIDAD DEL VALLE

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS

TULUA

[1]sagrath23@gmail.com

"*En la música todos los sentimientos vuelven a su estado puro y el mundo no es sino música hecha realidad.*" - Arthur Schopenhauer.

"*La música es la aritmética de los sonidos, como la óptica es la geometría de la luz.*" - Claude Debussy.

Quiero dar las gracias entre muchas personas a las siguientes:

- A mis padres, que a pesar de no estar presentes fisicamente durante mi proceso de formación profesional, siempre fueron el motor que me impulso a seguir adelante a pesar de las adversidades.
- A mi director, maestro, pero ante todo amigo, el Ingeniero David Alejandro Przybilla, quien desinteresadamente me guió durante el desarrollo del proyecto.
- A mis compañeros y amigos con quienes he compartido alegrias y tristezas durante estos 5 años de formación profesional.

Índice general

Capítulo 1

Introducción

El objetivo de este capítulo es brindar una descripción del problema que aborda este trabajo de grado, justificar su solución y establecer los objetivos generales y específicos que se cumplirán para su solución.

1.1. Descripción del Problema

Una de las aplicaciónes del modelamiento y resolución de problemas basado en restricciones que no se asociaría normalmente con el tipo de problemas modelados usando esta técnica es la música. Una de las ventajas de la música respecto a otras artes, como el dibujo, el baile o la arquitectura, es que esta es la que ha sido objeto de rigurosas formalizaciones desde sus mas tempranas etapas. En especial la llamada *música tonal*[1], basada en la idea de *tonalidad*[2], ha sido desarrollada en un marco de trabajo que está particularmente bien adaptado al modelamiento computacional y a su solución por medios computacionales. Las aplicaciones de la programación por restricciones en música pueden cubrir un amplio rango de estructuras e ideas musicales, ya que muchos de estos son problemas de carácter combinatorio.

Normalmente, se espera que una libreria de programación por restricciones encuentre al menos una respuesta que satisfaga todas las restricciones que definen el modelo del problema. A este tipo de soluciones se les denomina como "soluciones completas"; pero debido a las rigurosas formalizaciones de las que ha sido objeto la música, la naturaleza de los problemas musicales es sobre-restringida, lo cual afecta gravemente el rendimiento de la libreria de programación por restricciones que se utilice.

Por este motivo se desarrollaron librerias de programación por restricciones que no buscan satisfacer todas las restricciones definidas en el modelo del problema, sino que buscan soluciones que satisfagan el mayor numero de restricciones posible. A este tipo de soluciones se les denomina como "soluciones parciales". Resulta obvio que una libreria que no busca soluciones completas tardara menos en encontrar soluciones a un problema en comparación con una libreria que solo busque soluciones completas al mismo problema.

A partir de esta situación, y del estado actual de las librerias de programación por restricciones, se plantean las siguientes preguntas:

- ¿es posible para una libreria de programación por restricciones que busque soluciones completas encontrar dichas soluciones para un problema de armonia musical?
- ¿puede encontrarlas en un tiempo razonable?

[1] En la música tonal, el papel de la tónica (primera nota de una escala) se afirma a través de las relaciones que con ella establecen los otros grados de la escala y los acordes (conjuntos de sonidos) que se forman sobre ellos. Este entramado armónico, es conocido como armonía funcional.

[2] jerarquización de todos los sonidos alrededor de uno principal (tónica).

- ¿puede una libreria como GECODE (que busca soluciones completas) competir en tiempo con una libreria como OMClouds (que busca soluciones parciales)?

1.2. Formulación del Problema

Después de este análisis se plantearon las siguientes preguntas:

- ¿Puede un CSL (Libreria de programación por restricciones) completo como GECODE encontrar soluciones completas a problemas de armonía musical, en un tiempo razonable en comparación al tiempo que emplea OMClouds para encontrar soluciones parciales?
- ¿Puede un CSL completo como GECODE encontrar soluciones completas a problemas de armonía musical, usando menos memoria que la usada por OMClouds para encontrar soluciones parciales?
- ¿Puede GECODE encontrar soluciones a otro tipo de problemas musicales (ritmicos, de composición), usando menos memoria y en menor tiempo que OMSituation y OmRC?
- ¿Que impacto tiene el tipo de modelamiento que se haga en el rendimiento del CSL?

1.3. Justificación

El aspecto mas relevante de los problemas de satisfacción de restricciones en armonia musical resulta ser su caracter sobre-restringido, por este motivo las siguientes afirmaciones justifican este trabajo:

- El estudio de problemas de satisfaccion de restricciones en armonia musical permitirá identificar tipos de modelamiento que se adapten mas a problemas de caracter sobre-restringido, y que mejore el comportamientos de los CSL en los que se implemente el modelo.
- El estudio de problemas de satisfacción de restricciones en armonia musical, su modelamiento y las herramientas que se utilizan para encontrar soluciones pueden ser utilizadas en otros problemas que tengan la caracteristica de ser sobre-restringidos, pero de naturalezas completamente diferente a la musical.
- A partir de las conclusiones que se puedan obtener de este trabajo, se pueden identificar escenarios en los cuales puede ser mas ventajoso utilizar un CSL completo como GECODE, y en cuales usar un CSL parcial como OMClouds.
- La comparación de las soluciones completas y las soluciones parciales de un mismo problema, permitirá determinar la relevancia musical de cada una, permitiendo establecer si es necesaria o no la busqueda de soluciones completas a este tipo de problemas.

1.4. Objetivos

1.4.1. Objetivo General

Caracterizar aspectos propios de GECODE que mejoren el rendimiento de este en la busqueda de soluciones a los tres casos de estudio planteados, con respecto a OMClouds.

1.4.2. Objetivos Específicos

- Definir los tres casos de estudio sobre los que se realizarán las pruebas con GECODE y OMClouds.
- Implementar los tres casos de estudio seleccionados en GECODE y OMCLouds.
- Comparar las soluciones obtenidas en GECODE contra las soluciones obtenidas en OMClouds para cada caso de estudio.
- Comparar los tiempos de ejecución de GECODE y OMClouds para cada caso de estudio.
- Comparar el espacio usado por el CSP usado en GECODE contra el espacio usado por el CSP implementado en OMClouds.
- Identificar aspectos técnicos y parámetros inherentes a GECODE que pueden afectar sustancialmente el desempeño en tiempo y consumo de memoria en la solución de problemas de armonía musical.
- Identificar aspectos y técnicas de modelamiento que puedan afectar sustancialmente el desempeño en tiempo y consumo de memoria en la solución de problemas de armonía musical.

Capítulo 2

Marco de Referencia

El objetivo de este capítulo es brindar el conocimiento necesario para entender el área en general donde se ubica este proyecto que es Programación por Restricciones y conceptos basicos de teoria musical y armonia musical.

2.1. Programación por Restricciones

2.1.1. Problemas de Satisfacción de Restricciones (CSPs)

En Programación por Restricciones el modelamiento de problema se hace convirtiendo la descripción del problema en un CSP (Problema De Satisfacción de Restricciones o Constraint Satisfaction Problem, en inglés), es decir identificando:

- Las variables del problema.
- El dominio de cada una de las variables del problema.
- Las condiciones que deben cumplir los valores solución para las variables que involucra el problema.

2.1.2. Utilidad de Programación por Restricciones

La Programación por Restricciones (CP o Constraint Programming, en inglés) trata del estudio de sistemas computacionales basados sobre restricciones. La idea de la programación por restricciones es resolver problemas declarando restricciones (condiciones o propiedades) las cuales deben ser satisfechas por la solución[8].

Existen una gran cantidad de aplicaciones de Programación por Restricciones dado la cantidad de problemas que involucran restricciones; para más información [8].

2.1.3. Concepto de Restricción

Una restricción es una relación lógica entre variables.
Por ejemplo, la pareja de variables X, Y puede estar relacionada por la restricción X < Y, las variables X, Y por si solas no están relacionas pero al introducir la restricción < se introduce información que deben cumplir las variables; algo lógico es que una variable pueda tomar un conjunto determinado de valores a esto se le denomina el dominio de la variable[8].

2.1.4. Definición Formal de un CSP

De manera formal un CSP P se define como:

$P =< X_s, D_s, C_s >$ *donde* $X_s = \{X_1 ... X_n\}$ es el conjunto de variables.

D_s es el conjunto de dominios $D_s =< D_1, D_2...D_n >$ de tal forma que $X_i \, \epsilon \, D_i$ Siendo U el conjunto universal de todos los posibles valores. Cada variable $X \, \epsilon \, X_s$tiene un posible conjunto de valores $D(X) \subseteq U$ que puede ser asignado a la variable y el cual es llamado el dominio de la variable.

$C_s =< C_1, C_2....C_t >$ es el conjunto de restricciones. Una restricción C_j es una tupla (R_{s_j}, S_j) donde R_{s_j} es una relación sobre las variables S_j, donde $S_i = Scope(C_i)$ (el conjunto de las variables afectadas por la restricción). Es decir R_i es un subconjunto del producto cartesiano de las variables en S_i.

Una solución al CSP P es una tupla $A =< a_1, a_2...a_n >$ donde $a_i \epsilon D_i$ y cada C_j es satisfecho de tal forma que$< a_1, a_2...a_n > \epsilon \, C_s$ [**?**, 31, 32].

2.1.4.1. Propiedades Importantes de Los CSPs

- **Equivalencia:**
  Considere una secuencia de Variables $X := x_1, ..., x_n$ con una secuencia de dominios $D_1, ..., D_n$.Tome un elemento $d := (d_1, ..., d_n)$ de $D_1 \times .... \times D_n$ y una subsecuencia $Y = x_{i_1}, ..., x_{i_l}$ de X. Entonces se denota la secuencia $(d_{i_1}, ..., d_{i_l})$ como $d[Y]$ y se lee como la proyección de d sobre Y[32].
  Con la proyección se define el primer concepto de equivalencia:

  Dos CSPs nombrados respectivamente P_1 y P_2 se consideran CSPs equivalentes si:

  $\{d[X] \,|\, d \; es \; una \; solucion \; a \; P_1\} = \{d[X] \,|\, d \; es \; una \; solucion \; a \; P_2\}$

  El segundo concepto de equivalencia:

  Considere los CSPs $P_0..P_m$ donde $m \geq 1$ y una secuencia X de sus variables comunes. Se dice que la unión de $P_1..P_m$ es equivalente a P_0 con respecto a X Si:

  $\{d[X] \mid d \; es \; una \; solucion \; a \; P_0\} = \bigcup_{i=1}^{m} \{d[X] \mid d \; es \; una \; solucion \; a \; P_i\}$.

  Es decir un CSPs puede verse como la unión de muchos otros CSPs.

- **Consistencia:**
  Un CSP es consistente o no fallido, si $Soluciones(X, D, C) \neq \emptyset$, es decir si existe al menos una solución[32].

- Existen varias representaciones en CSP de un mismo problema[32].

- Un CSP está resuelto, si todas las restricciones han sido resultas y para cada X_i que pertenece a X, $Dominio(X_i) \neq \phi$[32].

2.1.5. Resolviendo CSPs

Una vez el CSP está planteado se procede a encontrar su solución. Para ello primeramente tiene que definirse si el CSP es consistente (si existe una solución). En caso de que el CSP cumpla con la propiedad de Consistencia el paso a seguir sería identificar a qué tipo de solución se pretende llegar:

- Encontrar una solución.
- Encontrar todas las soluciones.
- Encontrar la solución óptima.

Una vez definida la solución que se espera, se escoge el método para llegar a la solución. Existen métodos específicos y generales. Los específicos son llamados Solucionadores de Restricciones (Constraint Solvers, en inglés)y se aplican en problemas con dominios específicos y características específicas. Por

ejemplo existen Solucionadores de Restricciones para resolver problemas que involucran ecuaciones lineales, problemas de variables del dominio de los reales, grafos, entre otros[31, 32].

Una vez planteados el CSP y el tipo de solución a la que se pretende llegar, empieza la resolución del CSP.

1. En primera instancia un CSP inicial generará otros CSPs equivalentes al CSP inicial, al ser transformado por los propagadores y reglas de transformación. Los propagadores en la primera etapa podarán los dominios de las variables según las reglas que le hayan sido especificadas.
2. Una vez los propagadores han podado los dominios de las variables del CSP, si no existe aun una solución, el motor de búsqueda y la estrategia de distribución se encargaran de explorar los dominios de varias de las variables. La estrategia de distribución (Branching, en inglés) define en cual orden se exploran las variables (e.g la variable con más restricciones, la variable con el dominio más pequeño) y en qué orden serán asignados los valores del dominio a esa variable (e.g el valor más grande del dominio, el valor más pequeño del dominio).Esta exploración del árbol y de los dominios de las variables genera nuevos CSPs equivalentes.
3. Los propagadores podan los dominios de los nuevos CSPs equivalentes.
4. Se itera entre el numeral 2 y numeral 3 explorando el árbol, los dominios de las variables y propagando. Hasta que se encuentre una solución o el dominio de una de las variables del CSP sea vacío, en cuyo caso no se ha encontrado una solución.

2.1.6. Propagadores (Filtering algorithms, en inglés)

Es un algoritmo que durante la búsqueda de las soluciones a un CSP, se encarga de reducir los dominios de las variables, sin descartar soluciones[31].

Un propagador siempre debe reducir el dominio , por tanto debe cumplir con las siguientes propiedades: Sean D_1 y D_2 almacenes (un almacén D define un dominio D(X) para cada variable X del CSP) y f un propagador asociado a la restricción C . f debe satisfacer las siguientes propiedades:

1. **Contractante:** $f(D_1) \subseteq D_1$
2. **Monótono:** $D_1 \subseteq D_2 \implies f(D_1) \subseteq f(D_2)$
3. **Correcto:** $D_1 \cap C \subseteq f(D_1)$

Un propagador convierte un CSP P en un P' de tal forma que sean CSPs equivalentes.

2.2. Teoría Musical

La teoría musical es un campo de estudio que tiene por objeto la investigación de los diversos elementos de la música, entre ellos el desarrollo y la metodología para analizar, escuchar, comprender y componer música. Mientras que la musicología puede incluir cualquier declaración, creencia o concepción de lo qué es la música, la teoría musical está limitada a las discusiones concernientes a los eventos sincrónicos (o diacrónicos) de una composición específica (o varias composiciones), y a los capítulos músico-teoréticos abstractos (por ejemplo teoría de conjuntos, teoría de grupos, teoría de tensión tonal, etc.).

2.2.1. Armonia

La armonía es una arte-ciencia, basada en la física que busca una clasificación de acordes e intervalos, buscando formas de combinarlos para producir distintas sensaciones: reposo, tranquilidad, pena, nostalgia ó tensión, desesperación, ira, excitación. Hay una armonía para cada grupo de sentimientos;

armonía consonante y disonante, respectivamente.

La armonía se refiere al aspecto «vertical» (simultáneo en el tiempo) de la música, que se distingue del aspecto horizontal (la melodía, que es la sucesión de notas en el tiempo). La idea de vertical y horizontal es una metáfora explicativa, relacionada a la disposición de las notas musicales en una partitura: verticalmente se escriben las notas que se interpretan a la vez, y horizontalmente las que se interpretan en forma sucesiva.

En la escolástica musical, el contrapunto es una disciplina complementaria a la armonía (y que se confunde con ella), pero que se centra más en la elaboración de melodías que sean combinables simultáneamente que en los acordes resultantes de tal combinación. Es decir: se centra más en la percepción de las partes que en la del todo. Como disciplina creativa (y no como disciplina académica), el contrapunto tuvo su auge durante el Barroco, particularmente con la figura de Johann Sebastian Bach.[6]

2.2.2. Contrapunto

El contrapunto se refiere a una parte de la teoría musical que estudia la técnica que se utiliza para componer música polifónica mediante el enlace de dos o más melodías (también voces o líneas) independientes que se escuchan simultáneamente. El término contrapunto deriva de punctum contra punctum, «nota contra nota» o «melodía contra melodía» y por sí mismo describe el pasaje musical consistente en dos o más líneas melódicas que suenan simultáneamente.[7]

2.2.3. Acorde

Se llama acorde a un grupo de sonidos, colocados en orden de terceras superpuestas, que se producen simultaneamente.[5]

Formalmente, un acorde consta de entre tres y siete notas de las doce que componen una octava; las notas pueden pertenecer a la misma o a diferentes octavas. La distancia entre dos notas musicales se conoce como intervalo musical; los intervalos musicales, combinados, determinan los diferentes tipos de acordes. Cada tipo de acorde puede presentar como tono fundamental cualquiera de las doce notas musicales (*do, do#, re, re#, mi, fa, fa#, sol, sol#, la, la#, si*). Este tono fundamental (también conocido como nota fundamental, nota tónica, fundamental o tónica) determina la tonalidad del acorde y constituye la referencia para los intervalos del mismo. En la fig. 2.1 se muestra un ejemplo de un acorde mayor en piano.

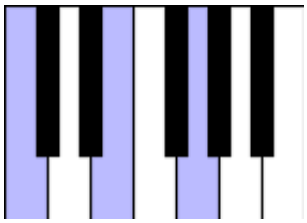

Figura 2.1: Do mayor en estado fundamental. El intervalo entre las dos primeras notas desde la izquierda (do y mi, respectivamente) define su carácter de acorde mayor.

2.2.4. Progresión Armónica

En música, una progresión armónica es una sucesión de acordes, explícitos o implícitos. Las más comunes son las que se basan en el círculo de quintas (ejemplo: I got Rhythm, de George Gershwin),

pero existe igualmente una cantidad comparable de piezas que no. Por extensión se añade el adjetivo progresivo a los subgéneros musicales que utilizan este tipo de composición. En la fig. 2.2 se muestra un ejemplo de una progresión armónica.

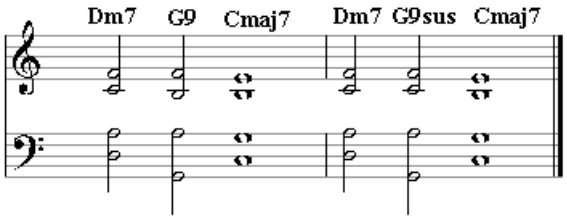

Figura 2.2: Progresión armónica de 6 acordes. Nótese que, en las notas del pentagrama, al acorde de Sol novena (*G9*) le falta la tercera y que al acorde de *Sol* novena con cuarta suspendida (*G9sus*) le falta la quinta.

2.2.5. Intervalo

Es la distancia entre dos notas. Un intervalo se mide a partir de un tono inicial, contando semitonos hasta una nota final; por ejemplo: En una escala de *Do* Mayor (*Do-Re-Mi-Fa-Sol-La-Si*), entre *Do* y *Mi* hay un intervalo de tercera ya que se recorren tres notas para llegar a *Mi* (*Do-Re-Mi*). El intervalo se mide en semitonos (fig. 2.4) y se expresa en grados (fig. 2.3).

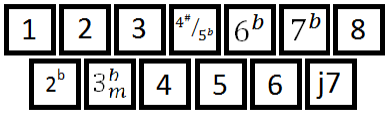

Figura 2.3: Diagrama de los intervalos musicales de la escala cromática en la primera octava a partir de la tónica.

Este tipo de clasificación para los intervalos musicales ayudan a ordenarlos según su sonoridad. Así, se pueden tener intervalos mayores, menores, justos, aumentados, disminuidos, super-aumentados y sub-disminuidos.

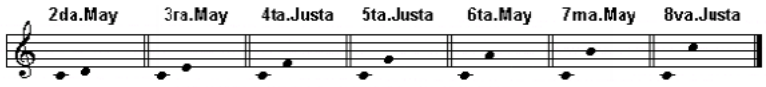

Figura 2.4: Intervalos musicales, mostrados sobre el pentagrama con la nota Do como tono fundamental.

Se puede tomar la tónica (primera nota) de una escala mayor e ir recorriendo toda la escala, identificando los intervalo correspondientes entre la tónica y cada una de las notas de la escala, comprobando que solo contiene intervalos mayores y justos.

2.2.6. Canon

El canon es una forma de composición musical de carácter polifónico basada en la *imitación estricta*[1] entre dos o más voces separadas por un intervalo temporal. La primera voz interpreta una melodía y es seguida, a distancia de ciertos compases, por sucesivas voces que la repiten, en algunos casos transformadas en su tonalidad o en otros aspectos. A la primera voz se le llama propuesta o «antecedente», y a las voces que le siguen respuesta o «consecuente». El canon aportó un gran beneficio al estudio del contrapunto, y, en consecuencia, al desarrollo de las distintas formas musicales.

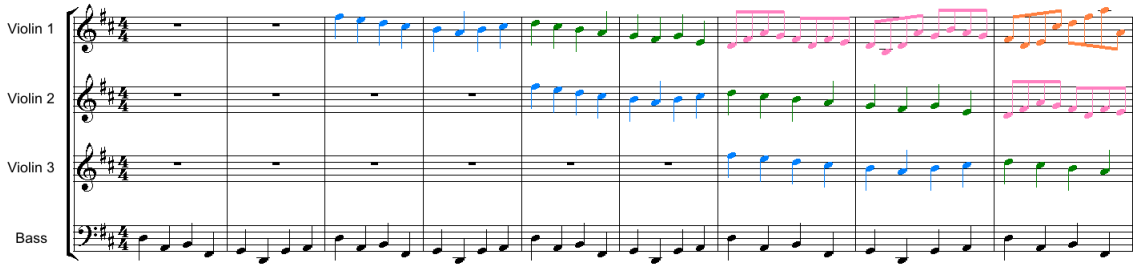

Figura 2.5: Canon en Re de Pachelbel

[1]En música, la imitación es la repetición posterior de un patrón musical en una forma diferente, pero manteniendo su carácter original.

Capítulo 3

Identificación y Análisis del Problema

El objetivo de este capítulo es presentar:

1. Los tres casos de estudio que se usaron en este trabajo.
2. Las caracteristicas de estos casos de estudio que los hace problemas sobre-restringidos.
3. Las caracteristicas de las librerias de programación por restricciones a estudiar.

Para cumplir con los anteriores objetivos fue necesario:

- Estudiar cada uno de los modelamientos de los casos de estudio seleccionados, para esto se buscaron problemas enfocados en armonia musical en articulos de investigación, tesis de maestria y tesis doctorales tales como [18].
- Estudiar las caracteristicas de las librerias de programación por restricciones seleccionadas.
- Estudiar las estrategias de distribución y busqueda de las librerias de programación por restricciones seleccionadas, y su impacto en el tiempo y espacio requerido en memoria para encontrar una solución.

3.1. Casos de Estudio

En esta sección se definiran las caracteristicas matematicas de los problemas enfocados en armonia musical, su respectivo modelamiento y aquellas caracteristicas propias del problema que lo convierten en un problema sobre-restringido.

3.1.1. Serie con Todos los Intervalos

Como se define en [19] este problema consiste en encontrar una permutación de n números consecutivos $0, 1, 2, ..., n-1$ tal que el valor absoluto de la resta entre cada par de numeros de la permutación sea a su vez una permutación de un conjunto de $n-1$ números consecutivos $1, 2, ..., n-1$.

Formalmente, tenemos que:

1. Sea $Z = \{0, 1, 2, ..., n-1\}$, se busca un conjunto $S = \{S_1, S_2, ..., S_n\}$tal que $P(Z, n) = S$.
2. Sea $Z_1 = Z - \{0\} = \{1, 2, ..., n-1\}$, se busca un conjunto $I = \{\|S_2 - S_1\|, \|S_3 - S_2\|, ..., \|S_n - S_{n-1}\|\}$ tal que $P(Z_1, n-1) = I$.

un ejemplo con $n = 12$, que representaría los 12 tonos de la escala cromatica occidental, se ilustra en la fig. 3.1:

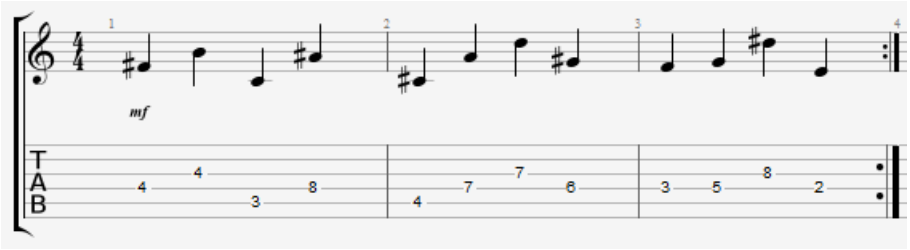

Figura 3.1: una solución al problema de la serie con todos los intervalos, para $n = 12$

Para este ejemplo, con $n = 12$, en la tabla 3.1 se muestra la representación de los 12 tonos que componen la escala cromatica.

Tono	do	do#	re	re#	mi	fa	fa#	sol	sol#	la	la#	si
Z	0	1	2	3	4	5	6	7	8	9	10	11

Cuadro 3.1: Equivalencia numerica de tonos de la escala cromatica

Entonces, para el ejemplo de la fig. 3.1 se tiene que $S = \{6, 11, 0, 10, 1, 9, 2, 8, 5, 7, 3, 4\}$ e $I = \{5, 11, 10, 9, 8, 7, 6, 3, 2, 4, 1\}$, lo que verifica que S es solución al problema de la serie con todos los intervalos para $n = 12$.

Segun [20], para $n = 12$ existen 3856 soluciones conocidas. Encontrar una solución para este problema es un proceso relativamente sencillo, ya que bastará con distribuir los n valores, así:

$$S = \{n,\ 1,\ n-1,\ 2,\ ...,\ floor(n/2) + 2,\ floor(n/2) - 1,\ floor(n/2) + 1,\ floor(n/2)\}$$

Obteniendo los intervalos:

$$I = \{n-1,\ n-2,\ ...,\ 2,\ 1\}$$

La dificultad de este problema se encuentra en encontrar todas las soluciones posibles para n valores.

Resulta obvio que este problema presenta simetrias en sus respuestas, ya que cualquier secuencia se puede rotar, generando una nueva respuesta para el problema, es decir, si $S = \{S_1, S_2, ..., S_n\}$, entonces $S = \{S_n, S_1, S_2, ..., S_{n-1}\}$tambien será una permutación que cumpla con las restricciones del problema. Esta simetria, denominada *simetria de espejo* se puede eliminar facilmente, haciendo que $S_1 < S_2$.[1]

Otra simetria que se presenta en este problema, se debe a la posibilidad de invertir la secuencia, es decir, si $S = \{S_1, S_2, ..., S_n\}$, entonces $S = \{S_n, S_{n-1}, ..., S_2, S_1\}$tambien será una permutación que cumpla con las restricciones del problema. Esta simetria, que genera soluciones iguales, se puede eliminar de la misma forma que se elimina en el problema de la *regla de golomb [29]*, haciendo que $I_1 > I_{n-2}$.[2]

Al aplicar estas restricciones sobre el problema, se obtienen solo 332 soluciones, como se demuestra en [30].

[1] *en esta parte debería explicar por que esta restricción elimina la simetria?*
[2] *en esta parte debería explicar por que esta restricción elimina la simetria?*

Resumen del Modelamiento

Datos: $n \in \mathbb{Z}^+$

Variables: $S_1, ..., S_n$ (tonos); $I_1, ..., I_n$ (Intervalos)

Dominio: $\mathbb{Z}^+$

Restricciones:

1. $I_i = \|S_i - S_{i+1}\| \ \forall i < n$
2. $alldif(S)$
3. $alldif(I)$

3.1.2. Serie de Acordes con Notas en Común

Como se describe en [18], este problema consiste en encontrar una permutación de un conjunto de acordes tal que el numero de notas en comun entre cada par de acordes sea mayor a 0.

Formalmente, tenemos que:

1. Sea n el numero de acordes y k el numero de notas que componen cada acorde.
2. Entonces, tenemos que un acorde $a_i = \{t_1, t_2, ..., t_k\}$
3. Sea $A = \{a_1, a_2, ..., a_n\}$, se busca un conjunto $S = \{s_1, s_2, ..., s_n\}$ tal que $P(A, n) = S$.
4. S debe cumplir que $\forall i \leq n-1, \ s_i \cap s_{i+1} \neq 0$.

En la fig. 3.2 se ilustra una solución al problema con exactamente una nota en comun entre cada par de acordes.

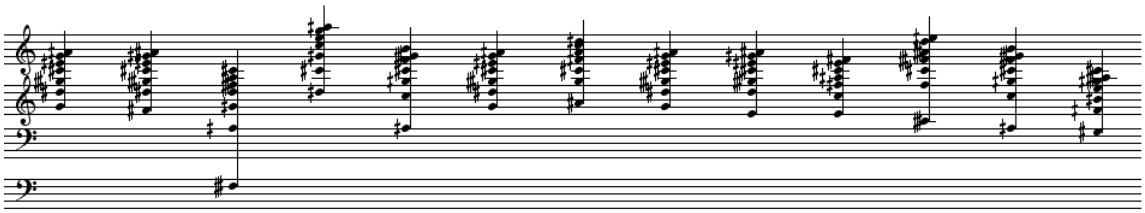

Figura 3.2: una solución con una nota común sobre toda la secuencia.

Ahora, debido a una particularidad del tipo de acordes que se usan en este problema, se puede cambiar la representación de los acordes, usando dos valores enteros f_i, que representa el tono fundamental del acorde; y t_i que representa el intervalo (distancia) entre los tonos.

De esta forma, tenemos que:

- Un acorde $a_i = \{f_i + (t_i * j), j \leq k\}$.

Tambien se puede condicionar la solución de este problema de tal forma que el numero de notas en comun entre cada par de acordes se encuentre en un rango determinado por dos valores enteros m y M, tales que:

- $m \leq \sharp(a_i \cap a_{i+1}) \leq M$

Siendo $\sharp$ la cardinalidad del conjunto generado por la intersección de a_i y a_{i+1}.

Una solución con $m = M = 2$ se muestra en la figura 3.3.

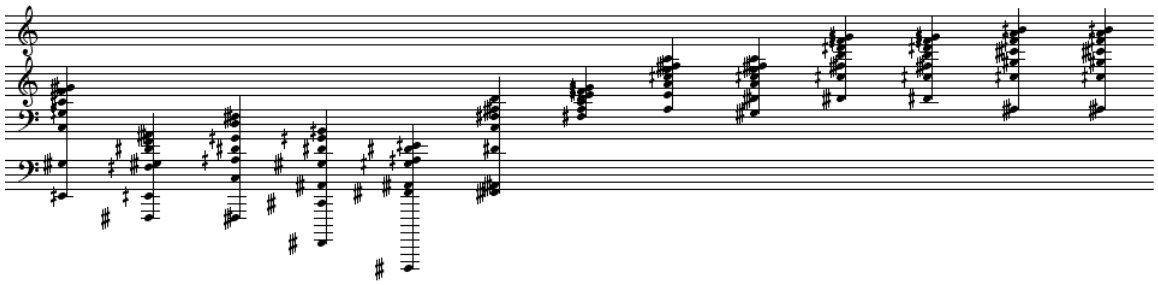

Figura 3.3: Una solución con exactamente dos nos notas comunes de un acorde al siguiente.

Modificando la representación de los acordes, se pueden incluir otras restricciones sobre el orden de las notas fundamentales y los intervalos de las notas, haciendo que las notas fundamentales aumenten al igual que los intervalos; de esta forma, tenemos que:

- $t_i < t_{i+1}$.
- $f_i < f_{i+1}$.

Una solución a este problema aplicando las restricciones sobre las notas fundamentales y los intervalos se muestra en la figura 3.4.

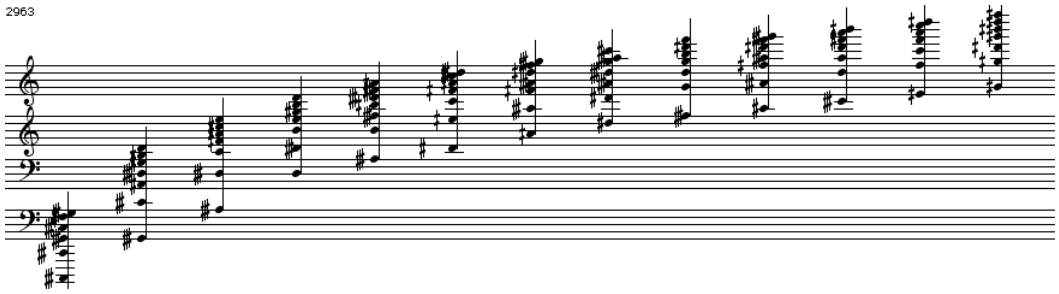

Figura 3.4: Una solución con dos restricciones sobre el movimiento global: notas fundamentales crecientes en intervalos decrecientes.

Resumen del Modelamiento

Datos: $I,\ J,\ m,\ M \in \mathbb{Z}^+$

Variables: $t_1, ..., t_n$ (intervalos), $f_1, ..., f_n$ (notas fundamentales)

Dominio: $\mathbb{Z}^+$

Restricciones:

1. $\alpha_i = \{f_i + (t_i * j), j \leq J\}$

2. $m \leq \sharp(\alpha_i \cap \alpha_{i+1}) \leq M$
3. (opcional) $t_i < t_{i+1}$
4. (opcional) $f_i < f_{i+1}$

3.1.3. Clasificación de Acordes

Como se define en [18, 33], dado un conjunto X de n acordes, $A_1...A_n$ (siendo A_i un conjunto de valores enteros), se busca ordenarlos de tal forma que el numero de notas en común entre dos acordes sucesivos sea máximo. Denotamos el numero de notas en común entre A_i y A_j por $C_{i,j}$, tal que $C_{i,j} = \sharp(A_i \cap A_j)$. El dominio es el conjunto de las permutaciones de A_i, que se obtiene con una restricción *alldif* (todos diferentes). La única restricción entonces se escribe: $max \sum_{1 \leq i \leq n-1} C_{i,i+1}$.

Esta restricción, en toda la secuencia, no impone la continuidad musical entre dos acordes sucesivos, es decir, no garantiza que exista una relación armonica entre todos los acordes del conjunto. A veces encontramos una solución óptima, pero con "agujeros" en la secuencia, por ejemplo cero notas en común entre dos acordes consecutivos. Para evitar esto, y mantener la continuidad musical, se pueden imponer $n-1$ restricciones $C_{i,i+1} \geq 1$[3].

Este es evidentemente equivalente al *problema del Viajante* (TSP por sus siglas en ingles, tomando como ciudades los acordes, y tomando como distancia entre la ciudad i y j $N - C_{i,j}$[4] tomando un N ($N \geq max_{1 \leq i,\, j < n} C_{i,j}$). Para un orden dado en los acordes, la suma de las notas comunes entre dos acordes consecutivos debe ser igual a $N * n$ menos la distancia de la ruta correcta entre las ciudades. Maximizar las notas comunes se convierte en minimizar la distancia total recorrida.[5]

Resumen del Modelamiento

Datos: n acordes $A_1, ..., A_n$

Variables: $A_1, ..., A_n$

Dominio: permutaciones de $A_1, ..., A_n$

Restricciones:

C_1 $Max \sum_{i \neq j} \sharp A_i \cap A_j$

C_2 (opcional) $\sharp A_i \cap A_j \geq 1$

3.1.4. Armonizar una Estructura Rítmica

Segun [18, 27] este problema consiste en armonizar un canon de longitud p y de n voces. Si se desean mas detalles sobre la estructura de un Canon ritmico, se puede consultar [28]. El objetivo de este problema es armonizar la estructura ritmica del canon, osea dar un valor definido a cada tono. Las notas se sostienen, y el acorde formado en cada instante varia solo en una nota con respecto al acorde anterior, debido a la estructura del canon. Pare esre fin se define C_i como el acorde tocado en el instante i.
La primera restricción consiste en minimizar una función de distancia E entre los acordes, denominada *distancia estrada.* Para calcular E se debe calcular la *textura* del acorde, que es el conjunto de todos los intervalos internos del acorde transportados a la misma octava, adicionando el complemento para

[3] Esta restricción causa que no se encuentre una solución para el problema si no existe una relación armonica entre los acordes de entrada.

[4] Usando esta representación, se soluciona la sobre-restricción del problema, generando un grafo completo entre los n acordes, pero no se garantiza la continuidad musical.

[5] *Nota: Seria interesante evaluar el concepto de modulación, una practica que permite cambiar el "tono" de la armonia y que permitiria garantizar la continuidad músical y las restricciones del modelo.*

que la suma de estos valores sea 12. por ejemplo, los acordes son sol#1,do#7, fa8[6]. Hay do#-fa-sol#, si consideramos la distancia entre los tonos, tenemos que entre do# y fa hay 4 tonos, y entre fa y sol# hay 3, y añadimos 5 para obtener la suma 12. el resultado es la textura 3-4-5.

La distancia d es entonces el número de valores diferentes en la textura, es decir, el número de notas en el acorde menos el numero de intervalos comunes enla textura, menos 1. Formalmente, tenemos que $E(Text_1, Text_2) = (numero\ de\ notas\ del\ acorde) - (numero\ intervalos\ comunes\ entre\ Text_1\ y\ Text_2) - 1$ representa el valor de la distancia estrada entre dos texturas. Una opción es minimizar la distancia estrada entre los acordes y un acorde fijo.

La segunda restricción consiste en minimizar la diferencia entre las fundamentales virtuales f de dos acordes consecutivos. Para este fin, se define f como la nota mas baja en un intervalo de quinta si se encuentra en el acorde, si no se encuentra, f sería la nota mas baja en un intervalo de tercera si se encuetra en el acorde, si no se encuentra, f sería la nota mas baja del acorde. Por este motivo, se define D_f, para valores de $d = (F(C_1) - F(C_2)) mod 12$, los cuales concuerdan con el circulo de quintas, mostrado en el cuadro 3.2.

d	0	1	2	3	4	5	6	7	8	9	10	11
D_f	0	9	7	6	5	2	8	1	4	3	11	10

Cuadro 3.2: Tabla de distancias

Una Tercera restricción condiciona el movimiento melodico de las voces. Se impone sobre cada voz un perfil comun definido, con el mismo espacio en las entradas de las voces, para reforzar la precepción del canon.

Este CSP esta sobrerestringido, y un solver dará como solucion la misma nota siempre. Para evitar esto, se añaden restricciones *alldiff* , dejando dos opciones sobre el movimiento de las voces: una vertical con respecto a las otras voces, y una horizontal sobre la misma voz. Verticalmente, todas las notas debe ser diferentes, y horizontalmente, dos notas sucesivas deben ser diferentes. La opción vertical tiene dos inconvenientes: no tiene un verdadero significado musical, y restringe aun mas el problema. la opción horizontal es una mejor opción por dos motivos: consiste solo en añadir una pequeña restricción y asegura un movimiento melodico para cada voz.

Resumen del Modelamiento

Datos: d_i turnos de entrada de voces, las funciones $voix$, $prev - voix$ y $next - voix$ para generar los índices de las voces, Distancias Estrada E, distancias entre las fundamentales virtuales D_f, un perfil melódico p_i dado en intervalos.

Variables: $V_1, ..., V_n$ (notas)

Dominio: $\mathbb{Z}^+$(múltiplos de doce)

Restricciones: Sea $A_i = \{V_i,\ V_{prev-voix(j)},\ j \neq voix(i)\}$

$Min\ E(A_i,\ A_{i+1})$

Variaciones:

C_1 $Min\ E(A_i,\ A)$ para un acorde A fijo

C_2 $Min\ D_f(A_i,\ A_{i+1})$

C_3 $Min\ \sum_{j \leq k} \mid V_{j+d_i} - p_j \mid$

C_4 $V_{prev-voix(i)} \neq V_i$ (preferencia)

[6]el numero al lado del tono indica la octava en la que se esta tocando el tono.

3.1.5. Primer Tipo de Contrapunto Fuxiano

Este tipo de contrapunto definido en [35], y formalizado en [34]consiste en encontrar una melodia que armonice una melodia dada. Dentro de las caracteristicas de este tipo de contrapunto tenemos que la duración de las notas y el numero de las notas de la melodia dada (*Cantus Firmus*) y la melodia buscada (*Counterpoint* o *Contrapunto*) son iguales. Un ejemplo de este tipo de contrapunto se muestra en la figura 3.7.

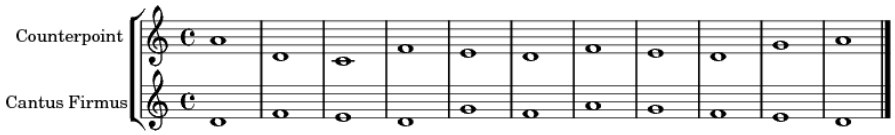

Figura 3.5: Ejemplo del primer tipo de contrapunto fuxiano

Formalmente tenemos que:
sea $F = \{f_1, f_2, ..., f_n\}$[7] se desea encontrar un conjunto $C = \{c_1, c_2, ..., c_n\}$ tal que:

1. El *intervalo melodico*[8] entre f_i y c_i esta permitido hasta la cuarta, la quinta y la octava, es decir, $\forall i < n \; \|f_i - c_i\| \in \{1, 2, 3, 4, 5, 7\}$.
2. Solo se permiten valores *diatonicos* (no se permiten intervalos aumentados, disminuidos o cromaticos), es decir, $\forall i < n - 1 \; \|c_i - c_{i+1}\| \in \{1, 2\}$.
3. se prohibe la repetición de la nota, es decir$c_i \neq f_i \; \forall c_i \in C$.

Resumen del Modelamiento

Datos: El cantus firmus $F = \{f_1, f_2, ..., f_n\}$ de n notas

Variables: $C = \{c_1, ..., c_n\}$

Dominio: $Z = \{0, 2, 4, 5, 7, 9, 11\}$

Restricciones:

C_1 $\forall i < n \; \|f_i - c_i\| \in \{1, 2, 3, 4, 5, 7\}$

C_2 $\forall i < n \; \|f_i - c_i\| \in \{1, 2\}$

C_3 $c_i \neq f_i \; \forall c_i \in C$

3.2. Librerias de Programación por Restricciones

3.2.1. Gecode

El ambiente de programación de restricciones Gecode fue diseñado utilizando el paradigma orientado a objetos. El modelo de propagación utilizado por Gecode es centrado en los propagadores; en este modelo los propagadores son planificados y ejecutados. Una desventaja del anterior modelo radica en que tiene un costo alto mantener información completa sobre los cambios hechos en el dominio de las variables modificadas; por el contrario, otros sistemas de restricciones utilizan un modelo propagación centrado en las variables. En éste la propagación se basa en planificar las variables modificadas, donde

[7] los valores de V_i se pueden dar en MIDI
[8] el intervalo melodico es aque en el que las notas se tocan de forma sucesiva.

el costo de almacenar información completa de las modificaciones hechas al dominio de las variables es bajo.

Una vista gráfica general Gecode 3.5.0 es proporcionada en [10].

Dependiendo de la tarea que se desee realizar en Gecode se trabaja con un componente distinto. En la siguiente lista están los componentes principales de Gecode 3.5.0:

3.2.1.1. Modelos y variables

La programación de un CSP en Gecode 3.5.0 va de la mano de la utilización de objetos variable. Las variables son clases disponibles al usuario final para que lleve a cabo el modelamiento del problema.

Si se desea programar un CSP en Gecode 3.5.0 se debe hacer:

- Crear una clase para modelar el problema la cual debe de extender de la clase Space de Gecode.
- Declarar las variables (los objetos variables permiten declarar un dominio inicial, fijar restricciones sobre estas y consultar el estado de su dominio) del CSP y su respectivo dominio.
- Especificar las restricciones y relaciones entre las variables declaradas en el punto anterior
- Especificar la estrategia de distribución que desea aplicar para la solución del problema.
- Definir un constructor de copia (Gecode 3.5.0 explora bajo copying)
- Crear una instancia de la clase Options de Gecode 3.5.0 e indicar cuantas iteraciones desea realizar en el proceso de búsqueda u otras opciones de configuración pertinentes al problema
- Crear una instancia de la clase que modela el problema.
- Pasar esta instancia a uno de los motores de busqueda definidos.

3.2.1.2. Motores de búsqueda y espacio

El encargado de encontrar las soluciones a un CSP en Gecode es el motor de búsqueda; este tiene 2 tareas que son: encontrar todas las soluciones al CSP y garantizar la correctitud de estas soluciones. Para esto el motor de búsqueda utiliza dos técnicas que son la búsqueda y la inferencia; la búsqueda (implementada con *la estrategia de distribución*) asegura encontrar todas las soluciones al CSP, la inferencia (implementada con *propagadores*) asegura que cada solución cumpla las restricciones del CSP.

Durante la búsqueda de las soluciones a un CSP el motor de búsqueda genera un árbol de búsqueda. Un nodo en el árbol de búsqueda representa un punto fijo (una foto del sistema en ejecución) calculado por la propagación, este punto fijo se denomina un espacio. Cada espacio se caracteriza por tener:

- Cola de propagadores: indica cuales propagadores deben ser corridos.
- Cola de estrategias de búsqueda.
- Dominios de variables.
- SpaceStatus.

El SpaceStatus de un espacio representa el estado del espacio actual, el cual puede ser: una solución al CSP, no ser una solución al CSP (por ejemplo una variable es vacía) o puede contener variables no asignadas (en el caso de variables enteras, variables con más de un valor), a nivel de implementación esto es:

- **SS_SOLVED**: Ningún propagador retorno ES_FAILED y ninguna distribución es permitida.

- **SS_FAILED**: Un propagador retornado es fallido.
- **SS_BRANCHED**: haga distribución, ya que las variables no están asignadas y el propagador no retorno falla así que pueden haber soluciones al CSP.

Para profundizar en como Gecode utiliza un espacio se puede consultar [11].

3.2.1.3. Propagador, condición de propagación y evento de modificación

Un propagador es una abstracción la cual brinda la implementación de una restricción.Un propagador a nivel conceptual está representado por unas reglas de filtrado; estas reglas de filtrado reflejan la forma en que el propagador va a podar los dominios de las variables y deben garantizar la correctitud del propagador, es decir, que genera las soluciones correctas a la restricción. En Gecode se pueden implementar nuevos propagadores que apliquen a las variables por defecto de Gecode, pero también para variables "nuevas", creadas a partir de las variables por defecto de Gecode. Cada propagador retorna un Status que indica al estado actual que ocurrió durante la propagación.

Desde el punto de vista del número de restricciones que toman los propagadores, los propagadores de Gecode se pueden clasificar en:

- **UnaryPropagators**: Se refieren a restricciones de una sola variable, e.g Sea X una variable entera, la restricción X>3 es una restricción unaria pues implica una sola variable de restricciones.
- **BinaryPropagators**: Se refieren a restricciones que involucran dos variables e.g: X && Y=0 donde X, Y son variables del dominio de los booleanos

3.2.1.4. Estrategias de distribución

La estrategia de distribución hace parte de la búsqueda de la solución a un CSP; ésta determina la forma la forma del árbol de búsqueda escogiendo la próxima variable y valor a distribuir.

La búsqueda de la solución comienza con los propagadores, una vez los propagadores han alcanzado un punto fijo, el proceso de distribución comienza.En este punto existen tres posibilidades:

- Todas las variables del CSP están asignadas.En este caso la búsqueda se detiene ya que una solución se ha obtenido.
- Existen variables que todavía no están asignadas.En este caso, se realiza un nuevo paso de distribución.Un paso de distribución consiste de:

  1. Seleccionar próxima variable a explorar.
  2. Seleccionar el valor (alternativa) de la variable escogida.

Una vez esta selección se ha hecho, se agrega la correspondiente restricción al almacén .Esta adición de restricción posiblemente active algunos propagadores, así que se debe esperar hasta que todos los propagadores queden fijos para realizar una nueva iteración de distribución.

Si hay al menos una variable cuyo dominio es vacío, en este caso, se vuelve al estado anterior, y se prueba una alternativa distinta esperando por la estabilización de los propagadores (que alcancen un punto fijo).

3.2.2. Strasheela

Strasheela es un sistema de restricciones musicales generico[34] altamente programable y extensible. Strasheela soporta CSP´s musicales que son extremadamente dificiles o imposibles de resolver en sistemas anteriores.

Strasheela emplea un modelo de restricciones basado en espacios computacionales, adaptando este

enfoque a los CSP´s musicales, y utulizando una representación generica muy poderosa, que ha sido diseñada especificamente para definir CSP´s musicales complejos.

3.2.2.1. Estrategia de Busqueda

Los espacios computacionales que emplea Strasheela encapsulan calculos basados en restricciones (los cuales son calculos especulativos), lo que hace que este modelo sea programable a un alto nivel. Al ser adaptado a la programación por restricciones de música, este modelo elimina muchos de los problemas de rendimiento de sistemas anteriores y hace posible la resolución de CSP's musicales que eran demasiado complejos para otros sistemas.

3.2.2.2. Propagación y Busqueda

El modelo de restricciones basado en espacios soporta un enfoque de busqueda llamado *propagar y buscar* (tambien conocido como propagar y distribuir). Algunas de las caracteristicas de este enfoque son:

- Este enfoque mantiene información parcial del estado de los dominios de las variables.
- *Propagar y buscar* realiza deducciones locales (por ejemplo propagación de restricciones).
- *Propagar y buscar* soporta procesos de busqueda programados. En un momento en el que no se puedan realizar deducciones locales, el motor de busqueda debe tomar una decisión. Para realizar un proceso de busqueda eficiente, es de vital importancia tomar decisiones donde una solución sea encontrada principalmente por deducciones locales. Este enfoque reduce el riesgo de tomar "malas decisiones".

3.2.2.3. Busqueda de Partituras (cifrados)

Strasheela adapta el modelo general basado en espacios a programación por restricciones de problemas musicales. El diseño de la representación musical de Strasheela le permite al proceso de desición aprovechar plenamente una partitura parcial (es decir, toda la información disponible de la representación musical) mientras se está tomando una decisión. Por lo tanto, decisiones adecuadas se pueden calcular de forma automática durante la busqueda de una partitura solución.[9]

3.2.3. OMClouds

Como se define en [2]OMClouds es una libreria para programación por restricciones incluida en OpenMusic, una herramienta para composición asistida por computador (CAC) que implementa un paradigma grafico de programación.

3.2.3.1. Criterios de Diseño

Durante el proceso de diseño de OMClouds, se estudiaron docenas de CSP's musicales, y de las conclusiones de este estudio, se determino que:

1. Es importante notar el gran numero de estructuras musicales que deben ser modeladas: acordes, notas, ritmos, tempo, armonias, movimientos, etc. Desde un punto de vista computacional, estas son variables de dominio finito, pero desde un punto de vista musical, significa que OMClouds debe estar en capacidad de representar cualquier tipo de objetos musicales. En algunos casos, las variables pueden representar directamente objetos musicales, en otros, se encuentran a lo largo de la partitura y se requiere un calculo para pasar de las variables al objeto musical. Por este motivo, no se le da un significado musical a las variables en OMClouds, simplemente se consideran como enteros que deben ser transformados a partir de las especificaciones del usuario.

[9] en esta parte seria conveniente dar un ejemplo??

2. Las restricciones son muy variadas. Se deben manejar restricciones aritmeticas, pero tambien restricciones de alto nivel como restricciones de capacidad o cardinalidad. Debido a esto, no hay forma de optimizar el solver para cualquier tipo de restricciones en una aplicación de composición musical.

3. Dado que los CSP's musicales muchas veces estan sobrerestringidos, la meta de OMClouds es encontrar soluciones aproximadas con sentido musical.

3.2.3.2. Busqueda Adaptativa

Los metodos heuristicos han sido usados en problemas de *Optimización Combinatoria* para encontrar soluciones optimas o cercanas a la optima durante decadas. El algoritmo basico consiste en empezar en una configuración aleatoria, explorar el vecindario (configuraciones adyacentes a la seleccionada), seleccionar la mejor de estas configuraciones y moverse a esta configuración; este proceso se repite hasta que se encuentra una solución adecuada al problema.

La *busqueda adaptativa* es un metodo heuristico propuesto por Philippe Codognet y Daniel Diaz para resolver CSP's. La entrada de este metodo es un problema en formato CSP. En este metodo se aprovecha la formulación del problema como un CSP, analizando cuidadosamente la configuración actual de las variables y restricciones, en comparación con una función de costo global a ser optimizado. Este metodo no esta limitado a un tipo especifico de restricción, ya que para cada restricción, se calcula una *función de error* que dá un indicador de "por cuanto" se esta violando esa restricción. La idea basica es calcular la *función de error* de cada restricción, entonces se combina para cada variable el error de todas las restricciones en las que aparece, lo que proyecta los errores de las restricciones en las variables involucradas. Finalmente, la variable con con el error mas grande, es seleccionado como "culpable" y su valor es modificado. En este segundo paso, se selecciona el mejor valor del dominio de la variable, es decir, se selecciona el valor que minimice el error total de la variable en la configuración resultante.[10]

[10] en esta parte resulta relevante hablar sobre como este metodo evita los minimos locales??

Capítulo 4

Implementación y Ejecución de los Problemas Seleccionados

Despues de formalizar los CSP's musicales descritos en el capitulo 3 de este trabajo, se procedio a realiza su implementación en las respectivas librerias de programación por restricciones seleccionadas.

4.1. Serie con Todos los Intervalos

Despues de implementar este CSP en las librerias GECODE, OMClouds y Strasheela, se realizaron varias corridas del modelo resultante, variando el único parametro de este problema, que es el número de notas que debe contener la escala que se solicita.

4.1.1. GECODE

En esta ejecución, se definio un nivel de consistencia ICL_BND, y 5 iteraciones para buscar soluciones con $n = 12$.

n	Soluciones	Memoria requerida	Tiempo requerido	Altura del arbol	Nodos del Arbol
12	332	52KB	3.200 seg (3200 mseg)	27	102369

Cuadro 4.1: Datos de la Ejecución de GECODE con ICL_BND

En esta ejecución, se definio un nivel de consistencia ICL_VAL, y 5 iteraciones para buscar soluciones con $n = 12$.

n	Soluciones	Memoria requerida	Tiempo requerido	Altura del arbol	Nodos del Arbol
12	332	52KB	7.132 seg (7132 mseg)	30	954553

Cuadro 4.2: Datos de la Ejecución de GECODE con ICL_VAL

En esta ejecución, se definio un nivel de consistencia ICL_DOM, y 5 iteraciones para buscar soluciones con $n = 12$.

n	Soluciones	Memoria requerida	Tiempo requerido	Altura del arbol	Nodos del Arbol
12	332	151KB	3.276 seg (3276 mseg)	23	19069

Cuadro 4.3: Datos de la Ejecución de GECODE con ICL_DOM

4.1.2. OMClouds

Datos OMClouds

4.1.3. Strasheela

En esta ejecución, se le especifico a strasheela que buscara todas las soluciones al problema.

n	Soluciones	Memoria requerida	Tiempo requerido	Altura del arbol	Nodos del Arbol
12	3856	151KB	5.60 seg (5600 mseg)	40	80214

Cuadro 4.4: Datos de la Ejecución de Strasheela con propagación FF

En esta ejecución, se le especifico a strasheela que buscara todas las soluciones al problema.

n	Soluciones	Memoria requerida	Tiempo requerido	Altura del arbol	Nodos del Arbol
12	3856	151KB	5.65 seg (5650 mseg)	40	79811

Cuadro 4.5: Datos de la Ejecución de Strasheela con propagación naive

4.2. Serie de Acordes con Notas en Comun

pruebas

4.3. Clasificación de Acordes

pruebas

4.4. Armonizar una Estructura Ritmica

pruebas

4.5. Primer Tipo de Contrapunto Fuxiano

mas pruebas

Bibliografía

[1] Jorgen Nixdorf, Constraint Programming, http://en.wikipedia.org/wiki/Constraint_programming. 2003

[2] Charlotte Truchet, OMClouds, a constraint solver for musical constraints, Metaheuristics International Conference (2003), págs.1–5

[3] Marc Battier, Open Music RC Library Versión 1.1, Segunda Edición Marzo 2000

[4] Guido Tack, Presentación GECODE Open-Source Software for Integer and Contraint Programming (OSSICP'08) Workshop, 2008

[5] Abraham Jurafsky, Manual de Armonia 10ma Edición, Editorial Ricordi

[6] http://es.wikipedia.org/wiki/Armon%C3%ADa

[7] http://es.wikipedia.org/wiki/Contrapunto

[8] http://ktiml.mff.cuni.cz/~bartak/constraints/intro.html

[9] Luis Quesada, Solving Constrained Graph Problems using Reachability Constraints based on Transitive Closure and Dominators, 2006.Tesis(Doctorado en Ciencias Aplicadas).Université catholique de Louvain. http://edoc.bib.ucl.ac.be:81/ETD-db/collection/available/BelnUcetd-11272006-164211/unrestricted/thesis.pdf

[10] Equipo de Gecode,Benchmark de Gecode con respecto a otras alternativas Similares. http://www.gecode.org/benchmarks.html

[11] Sven Lämmermann,Christian Schulte,Thomas Sjöland.Constraints. Royal Institute of Technology Stockholm, Sweden. http://web.it.kth.se/~cschulte/talks/Constraints@KTH.pdf

[12] Cambios de Gecode 2.0.0 con respecto a Gecode 1.3.1 http://www.gecode.org/gecode-doc-latest/PageChanges_2_0_0.html

[13] Equipo de Gecode,Sitio Web Oficial de Gecode. http://www.gecode.org

[14] http://en.wikipedia.org/wiki/Kneser_graph

[15] Tack G,Constraint Propagation -Models , Techniques , Implementation Saarland University , Germany ,2009

[16] CP(Graph): Introducing a Graph Computation Domain in Constraint Programming ,Gregoire Dooms, Yves Deville, Pierre Dupont,Department of Computing Science and Engineering Université catholique de Louvain

[17] Gecode, An Open Constraint Solving Library,Guido Tack, Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP)

[18] Charlotte Truchet, Contraintes, Rechearche Locale et Composition Assitée par Ordinateur, Tesis de Doctorado, Universidad de Paris 7

[19] Prob007 de CSPLib, http://www.cs.st-andrews.ac.uk/~ianm/CSPLib/prob/prob007/spec.html

[20] Robert Morris y Daniel Starr, The Structure of All-Interval Series, Journal of Music Theory Vol. 18, No. 2 (Autumn, 1974), pp. 364-389

[21] Charlotte Truchet, Some Constraint Satisfaction Problems in Computer Assisted Composition and Analysis, Perspectives in Mathematical and Computational Music Theory, 2004

[22] CP(Graph) Version No Oficial - http://www.cs.brown.edu/~gdooms/Software.html

[23] Gecodes ChangeLog - http://www.gecode.org/changes.html

[24] Gecode's Architecture, Raphael Reischuk,Graduate Seminar PS Lab,Marzo, 2008,http://www.ps.uni-sb.de/~raphael/bachelor/downloads/talk2-80313.pdf

[25] http://foldoc.org/

[26] Didier Croes, http://www.bigre.ulb.ac.be/Users/didier/pathfinding/images/badconnection.png

[27] http://www.epos.uni-osnabrueck.de/music/templates/buch.php?id=48&page=/music/books/m/ma_nl004

[28] http://www.epos.uni-osnabrueck.de/music/templates/buch.php?id=48&page=/music/books/m/ma_nl004

[29] http://www.cs.st-andrews.ac.uk/~ianm/CSPLib/prob/prob006/spec.html

[30] Ian Gent, Conditional Simmetry in All Interval Series Problem, University of St Andrew, 2003

[31] Gregoire Dooms.The CP(Graph) Computation Domain in Constraint Programming 2005-2006. Tesis(Doctorado en Ciencias Aplicadas). Université catholique de Louvain. www.info.ucl.ac.be/~dooms/thesis.pdf

[32] F. Rossi, P.Van Beek, T.Walsh. Handbook of constraint programming. Elsevier, 2006. ISBN 978-0-444-52726-4

[33] http://www.epos.uni-osnabrueck.de/music/templates/buch.php?id=48&page=/music/books/m/ma_nl004

[34] Torsten Andersen,Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System, Tesis de Doctorado, Queen's University, 2007

[35] Johan Joseph Fux (1965, orig. 1725). The Study of Counterpoint. from Johann Joseph Fux's Gradus ad Parnassum. W.W. Norton & Company. traducido y editado por Alfred Mann