The Adaptive Search Method for Constraint Solving and its application to musical CSPs

Philippe Codognet
University of Paris 6
LIP6, case 169
8, rue du Capitaine Scott
75 015 Paris, FRANCE
Philippe.Codognet@lip6.fr

Daniel Diaz
University of Paris 1
CRI, bureau C1407
90, rue de Tolbiac
75 634 Paris, FRANCE
Daniel.Diaz@inria.fr

Charlotte Truchet
IRCAM
1, place Igor Stravinsky
75 001 Paris, FRANCE
Charlotte.Truchet@ircam.fr

Abstract: We here propose a generic, domain-independent local search method called Adaptive Search for solving Constraint Satisfaction Problems (CSP). We design a new heuristics that takes advantage of the structure of the problem in terms of constraints and variables and can guide the search more precisely than a global cost function to optimize (such as for instance the number of violated constraints). We also use an adaptive memory in the spirit of Tabu Search in order to prevent stagnation in local minima and loops. This method is generic, can apply to a large class of constraints (e.g. linear and non-linear arithmetic constraints, symbolic constraints, etc) and naturally copes with over-constrained problems. Preliminary results on some classical CSP problems show very encouraging performances. We also applied this method is the area of computer-assisted composition in music and we also present results of an implementation of the Adaptive Search algorithm for musical CSPs that come from collaboration with contemporary music composers.

Keywords: constraint satisfaction techniques, CSP, local search techniques, metaheurisitics, computer music

1 Introduction

Heuristic (i.e. non-complete) methods have been used in Combinatorial Optimization for finding optimal or near-optimal solutions since a few decades, originating with the pioneering work of Lin on the Traveling Salesman Problem [15]. In the last few years, the interest for the family of Local Search methods for solving large combinatorial problems has been revived, and they have attracted much attention from both the Operations Research and the Artificial Intelligence communities, see for instance the collected papers in [1] and [25], the textbook [17] for a general introduction, or (for the French speaking reader) [13] for a good survey. Although local search techniques have been associated with basic hill-climbing or greedy algorithms, this term now encompasses a larger class of more complex methods, the most well-known instances being simulated annealing, Tabu search and genetic algorithms, usually referred as "meta-heuristics". They work by iterative improvement over an initial state and are thus anytime algorithms well-suited to reactive environments. Consider an optimization problem with cost function which makes it possible to evaluate the quality of a given configuration (assignment of variables to current values) and a transition function that defines for each configuration a set of "neighbors". The basic algorithm consists in starting from a random configuration, explore the neighborhood, select an adequate neighbor and then move to the best candidate. This process will continue until some satisfactory solution is found. To avoid being trapped in local optima, adequate mechanisms should be introduced, such as the adaptive memory of Tabu search, the cooling schedule of simulated annealing or similar stochastic mechanisms. Very good results have been achieved by dedicated and finely tuned local search methods for many problems such as the Traveling Salesman Problem, scheduling, vehicle routing, cutting stock, etc. Indeed, such techniques are now the most promising approaches for dealing with very large search spaces, when the problem is too big to be solved by complete methods such as constraint solving techniques.

In the last years, the application of local search techniques for constraint solving started to raise some interest in the CSP community. Localizer [18, 19] proposed a general language to state different kinds of local search heuristics and applied it to both OR and CSP problems, and [23] integrated a constraint solving component into a local search method for using constraint propagation in order to reduce the size of the neighborhoods. GENET

[9] was based on the Min-Conflict [21] heuristics, while [22] proposed a Tabu-based local search method as a general problem solver but this approach required a binary encoding of constraints and was limited to linear inequalities. Very recently, [12] developed another Tabu-based local search method for constraint solving. This method, developed independently of our adaptive search approach, also used so-called "penalties" on constraints that are similar to the notion of "constraint errors" that will be described later. It is worth noticing that the first use of such a concept is to be found in [5].

We propose a new heuristic method called Adaptive Search for solving Constraint Satisfaction Problem, that has been originally presented in [6, 8]. Our method can be seen as belonging to the GSAT [27], Walksat [28] and Wsat(OIP) [31] family of local search methods. But the key idea of our approach is to take into account the structure of the problem given by the CSP description, and to use in particular variable-based information to design general meta-heuristics. This makes it possible to naturally cope with heterogeneous problem descriptions, closer to real-life application that pure regular academic problems.

Preliminary results on classical CSP benchmarks such as the simple "N-queens" problem or the much harder "magic square" or "all-intervals" problems show that the adaptive search method performs very well w.r.t. traditional constraint solving systems.

2 Adaptive Search

The input of the method is a problem in CSP format, that is, a set of variables with their (finite) domains of possible values and a set of constraints over these variables. A constraint is simply a logical relation between several unknowns, these unknowns being variables that should take values in some specific domain of interest. A constraint thus restricts the degrees of freedom (possible values) the unknowns can take; it represents some partial information relating the objects of interest. Constraint Solving and Programming has proved to be very successful for Problem Solving and Combinatorial Optimization applications, by combining the declarativity of a high-level language with the efficiency of specialized algorithms for constraint solving, borrowing sometimes techniques from Operations Research and Numerical Analysis [26]. Several efficient constraint solving systems for finite domain constraints now exists, such as ILOG Solver [24] on the commercial side or clp(FD)[7] and GNU-Prolog [10] on the academic/freeware side. Although we will completely depart in adaptive search from the classical constraint solving techniques (i.e. Arc-Consistency and its extensions), we will take advantage of the formulation of a problem as a CSP. Such representation indeed makes it possible to structure the problem in terms of variables and constraints and to analyze the current configuration (assignment of variables to values in their domains) more carefully than a global cost function to be optimized, e.g. the number of constraints that are not satisfied. Accurate information can be collected by inspecting constraints (that typically involve only a subset of all the problem variables) and combining this information on variables (that typically appear in only a subset of all the problem constraints).

Our method is not limited to any specific type of constraint, e.g. linear constraints as classical linear programming or [31]. However we need, for each constraint, an *error function* that will give an indication on how much the constraint is violated. Consider a n-ary constraint $C(X_1, \ldots, X_n)$ and domains $D_1, \ldots, D_n$ for variables $\{X_1, \ldots, X_n\}$. An error function f_c associated to the constraint C is simply a real-valued function from $D_1 \times \ldots \times D_n$ such that $f_C(X_1, \ldots, X_n)$ has value zero if $C(X_1, \ldots, X_n)$ is satisfied. We do not impose the value of f_C to be different from zero when the constraint is not satisfied, and this will indeed not be the case in some of the examples we will describe below. The error function will in fact be used as a heuristic value to represent the "degree of satisfaction" of a constraint and thus to check how much a constraint is violated by a given tuple. For instance the error function associated to an arithmetic constraint $|X - Y| \leq C$ will be $max(0, |X - Y| - C)$. Adaptive search relies on iterative repair, based on variables and constraint errors information, seeking to reduce the error on the worse variable so far. The basic idea is to compute the error function of each constraint, then combine for each variable the errors of all constraints in which it appears, therefore projecting constraint errors on involved variables. Finally, the variable with the maximal error will be chosen as a "culprit" and thus its value will be modified. In this second step we use the well-known min-conflict heuristics [21] and select the value in the variable domain that has the best error immediate value, that is, the value for which the total error in the next configuration is minimal. This is similar to the steepest ascent heuristics for traditionnal hillclimbing.

In order to prevent being trapped in local minima, the adaptive search method also includes an adaptive memory as in Tabu Search : each variable leading to a local minimum is marked and cannot be chosen for the few next iterations. A local minimum is a configuration for which none of the neighbor improve the current configuration. This corresponds in adaptive search to a variable whose current value is better than all alternative values in its domain. It is worth noticing that conversely to most Tabu-based methods (e.g. [11] or [12] for a CSP-oriented framework) we mark variables and not couples $< variable, value >$, and that we do not systematically mark variables when chosen in the current iteration but only when they lead

to a local minimum. Observe however that, as we use the min-conflict heuristics, the method will never choose the same variable twice in a row.

It is worth noticing that the adaptive search method is thus a generic framework parametrized by three components :

- A family of error functions for constraints (one for each type of constraint)
- An operation to combine for a variable the errors of all constraints in which it appears
- A cost function for a evaluating configurations

In general the last component can be derived from the first two one. Also, we could require the combination operation to be associative and commutative.

3 General Algorithm

Let us first detail the basic loop of the adaptive search algorithm, and then present some extra control parameters to tune the search process.

Input :

Problem given in CSP form :

- a set of variables $V = \{V_1, V_2, \ldots, V_n\}$ with associated domains of values
- a set of constraints $C = \{C_1, C_2, \ldots, C_k\}$ with associated error functions
- a combination function to project constraint errors on variables
- a (positive) cost function to minimize

Output :

a sequence of moves (modification of the value of one of the variables) that will lead to a solution of the CSP (configuration where all constraints are satisfied) if the CSP is satisfied or to a partial solution of minimal cost otherwise.

Algorithm :

Start from a random assignment of variables in V

Repeat

1. Compute errors of all constraints in C and combine errors on each variable by considering for a given variable only the constraints on which it appears.
2. select the variable X (not marked as Tabu) with highest error and evaluate costs of possible moves from X
3. if no improving move exists
   then mark X tabu for a given number of iterations
   else select the best move (min-conflict) and change the value of X accordingly

until a solution is found or a maximal number of iterations is reached

Some extra parameters can be introduced in the above framework in order to control the search, in particular the handling of (partial) restarts. One first has to precise, for a given problem, the *Tabu tenure* of each variable, that is, the number of iteration a variable should not be modified once it is marked due to local minima. Thus, in order to avoid being trapped with a large number of Tabu variables and therefore no possible diversification, we decide to randomly reset a certain amount of variables when a given number of variables are Tabu at the same time. We thereafter introduce two other parameters : the *reset limit*, i.e. the number of simultaneous Tabu variables to reach in order to randomly reset a certain ratio of variables (*reset percentage*). Finally, as in all local search methods, we parametrize the algorithm with a maximal number of iterations (*max iterations*). This could be used to perform early restart, as advocated by [28]. Such a loop will be executed at most *max restart* times before the algorithm stops.

This method, although very simple, is nevertheless very efficient to solve complex combinatorial problems such as classical CSPs, as we will see in the next section. It is also worth noticing that this method has several sources of stochasticity. First, in the core algorithm, both in the selection of the variable and in the selection of the value for breaking ties between equivalent choices (e.g. choosing between two variables that have the same value for the combination of their respective constraint errors). Second, in the extra control parameters that have just been introduced, to be tuned by the user for each application. For instance if the reset limit (number of simultaneous tabu variables) is very low, the algorithm with restart very often, enhancing thus the stochastic aspects of the method; but on the other hand if the reset limit is too high, the method might show some trashing behavior and have difficulties in escaping local minima. Last but not least, when performing a restart, the algorithm will randomly modify the values of a given percentage of randomly chosen variables (the reset percentage). Thus a reset percentage of 100 % will amount to restart each time from scratch.

4 Examples

Let us now detail how the adaptive search method performs on two classical CSP examples, and we will compare

our algorithm with two other local search methods. We have tried to choose benchmarks on which other CSP-oriented local search methods have been applied in order to obtain comparison data, but is it worth noticing that all these benchmarks have satisfiable instances. For a detailed performance evaluation and comparison, see [8].

For each benchmark we give a brief description of the problem and its modeling in the adaptive search approach. Then, we present performance data averaged on 10 executions, including:

- instance number (i.e. problem size)
- average, best and worst CPU time
- total number of iterations (within a single run, on average)
- number of local minima reached (within a single run, on average)
- number of performed swaps (within a single run, on average)
- number of resets (within a single run, on average)

Then we compare those performance results (essentially the execution time) with other methods among the most well-known constraint solving techniques: constraint programming systems [10, 24], general local search system [18, 19], Ant-Colony Optimization [29]. We have thus favored academic benchmarks over randomly generated problems in order to compare to literature data.

Obviously, this comparison is preliminary and not complete but it should give the reader a rough idea of the potential of the adaptive search approach. We intend to make a more exhaustive comparison in the near future.

4.1 Magic to the square

The magic square puzzle consists in placing on a NxN square all the numbers in $\{1, 2, \ldots, N^2\}$ such as the sum of the numbers in all rows, columns and the two diagonal are the same. It can therefore be modeled in CSP by considering N^2 variables with initial domains $\{1, 2, \ldots, N^2\}$ together with linear equation constraints and a global all_different constraint stating that all variables should have a different value. The constant value that should be the sum of all rows, columns and the two diagonals can be easily computed to be $N(N^2 + 1)/2$.

The instance of adaptive search for this problem is defined as follows. The error function of an equation $X_1 + X_2 + \ldots + X_k = b$ is defined as the value of $X_1 + X_2 + \ldots + X_k - b$. The combination operation is the absolute value of the sum of errors (and not the sum of the absolute values, which would be less informative : errors with the same sign should add up as they lead to compatible modifications of the variable, but not errors of opposite signs). The overall cost function is the addition of absolute values of the errors of all constraints The method will start by a random assignment of all N^2 numbers in $\{1, 2, \ldots, N^2\}$ on the cells of the NxN square and consider as possible moves all swaps between two values.

The method can be best described by the following example which depicts information computed on a 4x4 square:

Values and Constraint errors

				-8
11	7	8	*15*	7
16	2	4	12	0
10	**6**	5	3	-10
1	14	9	13	3
4	-5	-8	9	-3

Projections on variables

8	2	1	8
4	8	16	9
6	**23**	21	1
1	2	5	9

Costs of next configurations

39	54	51	*33*
53	67	61	41
45	57	57	66
77	43	48	41

The table on the left shows the configuration of the magic square at some iteration (each variable corresponds to a cell of the magic square). Numbers on the right of rows and diagonals, and below lines, denote the errors of the corresponding constraints. The total cost is then computed as the sum of the absolutes values of those constraints errors and is equal to 57. The table immediately on the right shows the combination of constraint errors on each variable. The cell (3,2) with value **6** (in bold font on the left table) has maximal error (**23**) and is thus selected for swapping. We should now score all possible swaps with other numbers in the square; this is depicted in the table on the right, containing the cost value of the overall configuration for each swap. The cell (1,4) with value *15* (in italic) gives the best next configuration (with cost *33*) and is thus selected to perform a move. The cost of the next configuration will therefore be 33.

Constraint programming systems such as GNU-Prolog or ILOG Solver perform poorly on this benchmark and cannot solve instances greater than 10x10. We can nevertheless compare with another local search method: this benchmark has been attacked by the Localizer system with a Tabu-like meta-heuristics. Localizer [18, 19, 20]

is a general framework and language for expressing local search strategies which are then compiled into C++ code. The following table compares the CPU times (in seconds) of Adaptive Search and the Localizer++ system on instances from 16x16 up to 50x50 . For a problem of size NxN the following settings are used for Adaptgive Search: Tabu tenure is equal to N-1 and 10 % of the variables are reset when $N^2/6$ variables are Tabu. The programs were executed on a PentiumIII 733 MHz with 192 Mb of memory running Linux RedHat 7.0. Timings for Localizer come from [20] and have been measured on a PentiumIII-800 and thus on a machine similar to ours (PentiumIII-733), but it is worth noticing however that the method used in Localizer consists in exploring at each iteration step the whole single-value exchange neighborhood (of size n^2).

size	Localizer	Adaptive
16x16	50.82	1.14
20x20	313.2	3.4
30x30	1969	18.09
40x40	8553	58.07
50x50	23158	203.4

Our results compare very favorably with those obtained with the Localizer system, as the adaptive search is two orders of magnitude faster. Moreover its performances could certainly be improved by careful tuning of various parameters (global cost function, Tabu tenure, reset level and percentage of reset variables, ...) in order to make the method truly adaptive indeed...

4.2 God saves the queens

This puzzle consists in placing N queens on a NxN chessboard so that no two queens attach each other. It can be modeled by N variables (that is, one for each queen) with domains $\{1, 2, \ldots, N\}$ (that is, considering that each queen should be placed on a different row) and $3 \times N(N-1)/2$ disequation constraints stating that no pair of queens can ever be on the same column, up-diagonal or down-diagonal :

$$\forall (i,j) \in \{1,2,\ldots,N\}^2,\ s.t.\ i \neq j :$$

$$Q_i \neq Q_j,\ Q_i + i \neq Q_j + j,\ Q_i - i \neq Q_j - j$$

Observe that this problem can also be encoded in Constraint Programming with three all_different global constraints.

We can define the error function for disequation as follows, in the most simple way : 0 if the constraint is satisfied and 1 if the constraint is violated. The combination operation on variables is simply the addition, and the overall cost function is the sum of the costs of all constraints.

The N-queens problem is a favorite in the CSP and AI communities , it is not very interesting in fact, as deterministic algorithms to solve this problem exists. Moreover the performance of classical (consistency-based) constraint solver on this problem vary quite a lot depending on the modelling of the problem : with a simple formulation, consistency-based solvers (e.g. Ilog Solver of GNU Prolog) will not perform very well for instances greater than one thousand, even with a good (first-fail) labelling heuristics or the "all_different" global constraint. Nevertheless fine tuned formulations can be made very efficient and give a quasi-deterministic behavior. Thus we will only compare here Adaptive Search with and an ant-colony optimization method (Ant-P solver), and the Local search Localizer++ system. Timings (in seconds) for the Ant-P solver are taken from [29] and divided by a factor 7 corresponding to the SPECint 95 ratio between the processors. Timings for Localizer come again from [20] and have been measured on a PentiumIII-800 and thus on a machine slightly more performant than ours. For a problem of size NxN, the following settings are used for the Adaptive Search : Tabu tenure is equal to 2 and 10 % of the variables are reset when $N/5$ variables are Tabu. The programs were executed on a PentiumIII 733 MHz with 192 Mb of memory running Linux RedHat 7.0.

size	Ant-P	Localizer	Adaptive
50	0.39		0.00
100	2.58		0.00
150	20.6		0.00
200	40.37		0.01
256	*	0.16	0.01
512	*	0.43	0.04
1024	*	1.39	0.15
2048	*	5.04	0.54
4096	*	18.58	2.14
8192	*	68.58	9.01
16384	*	260.8	46.29
32768	*	1001	270.2
65536	*	10096	1320

Surprisingly the behavior of the adaptive search is almost linear and the variance between different executions is quasi non-existent. The above table clearly show that adaptive search is much more efficient on this benchmark, which might not be very representative of real-life applications but is a not-to-be-missed CSP favorite...

5 Adaptive Search for Musical CSPs

An interesting application area of constraint modelling and solving, quite distinct from the mainstream OR-related problems is music, and more precisely the field of Computer Assisted Composition (CAC). CAC is now

a well-established research area, see [4]. It deals with a symbolic representation of music, mainly at the score level (or before), in opposition with sound synthesis, accoustic, etc. CAC provides the composer with computing tools to handle with formal representations of music and make computations on these. We started working with contemporary music composers at IRCAM (the French Research Institute on Music and Accoustics) on their Constraint Satisfaction Problems (CSP), or, precisely, on the composition problems that could be modelled as CSPs. This gave us both a catalogue of music-based CSPs and a panorama of the way composers can use this AI technique. This work takes place in the broader development of the CAC software OpenMusic [2], a full visual programming language based on CommonLisp / CLOS, developped at IRCAM. OpenMusic is a functional, object, visual programming language. A set of provided classes and libraries make it a very convenient environment for music composition. OpenMusic is developed jointly by computer scientists and composers, which gave us the opportunity to consult musician users about the use they could do of constraint modelling, before developing a constraint solving system.

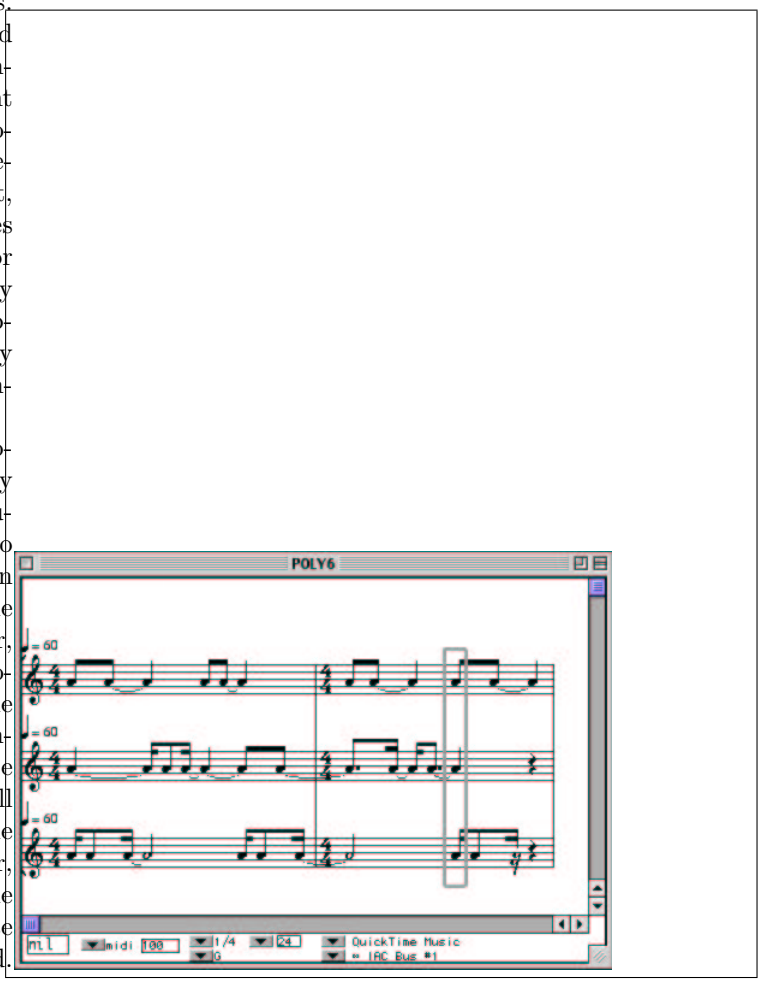
It is important to note that this composition problems differ from classical combinatorial problems usually tackled by CSP systems in several respects. In particular, most of these CSPs are over-constrained and have no solution. So they have to be considered as optimization problems, where the objective function is to maximize the degree of satisfaction of the overall constraints. Moreover, and this is maybe more surprising for the Constraint Programming community, the goal is not really to solve the CSP, but to provide interesting instanciations for the composer. It happens that an approximate solution could be musically more interesting than an exact one. We shall never forget the exact place of constraint solving in the compositional process : it is just a tool for the composer, among many others. It is at the rough draft level of the composition, not only because it is CAC, but also because a "solution" is not guaranteed of sounding musically good. In fact, the solutions produced by the system are nearly always re-written by hand. So the usual CSP paradigm "problem → solution" is no longer valid, we rather consider it as "specify some partial information on musical objects → make suggestions to the composer".

Let us look at three examples of musical CSP to detail those issues.

5.1 Asynchronous Rhythms

This problem is given by the italian composer Mauro Lanza (in Erba Nera Che Cresci, Segno nero Tu Vivi for voice and electronic, Aschenblume for ensemble, and Burger Time, for tuba and electronics). The goal is to construct rythmical patterns played repetitively and asynchronously, see the figure below. Formally, we have n voices, and a fixed time unit. Each voice plays repetitively a rythmic pattern of fixed length l_i, with n_i onsets, which can be represented as a set of integers $\{V_{i,j}, j \leq n_i\}$, where i is the voice number.

The figure below describes an instance of this problem for which there is no exact solution. The figure shows an approximate solution with four errors, indicated by the arrows



More generally this problem can be modelled by distinct types of constraints. The first constraint states that for a fixed duration D, we never hear two onsets at the same time. It can be written $all\_different(V_{i,1} \dots V_{i,n_i})$, so that the pattern is not degenerated. We could eliminate this constraint by taking the durations as variables and removing 0 from the domains, but it makes the second constraint more complex, as a capacity constraint. Second constraint, for two different voices, $i_1 \neq i_2$, two onsets k_1 and k_2 and for all p_1 such that $V_{i_1,k_1} + p_1 * l_1 \leq D$ (the onset is repeated while it doesn't exceed the duration),

$$V_{i_1,k_1} + p_1 * l_1[l_2] \neq V_{i_2,k_2}$$

An important parameter of this CSP is the average density of onsets. A density of 2 means that we hear in average one onsets every 2 time units. Of course, it fixes the number of onsets by pattern, that is the number of variables for our CSP. There is obviously no solution when the density of onsets becomes too big.

For two variables on two different voices, the constraint is exactly the negation of the chinese theorem, ie solve $x = V_1[l_1]$ and $x = V_2[l_2]$. Thus, if $D = ppcm(l_1, l_2)$, there is no solution. More precisely, the constraint states that the solution given by the chinese theorem have to be greater than D. So for n variables, we'd better take prime numbers for the l_i, so that for every couple i, j, $D < ppcm(l_i), l_j)$. Actually, the composer already had chosen prime l_i, before modelling the problem.

5.2 Gestures

This problem has been given by the french composer Gilbert Nouno (in Droben, by Michaël Jarrell, for double bass, ensemble and electronics). The goal is to find a melody with both intervals and notes in fixed domains. Before stating the other constraints, notice some duality in the problem. Depending on the sets chosen for the notes, N, and for the intervals, I, it could be unsatifiable. We can either favor the notes or the intervals in the CSP model. Here the composer wanted to work on gestures, represented as intervals or sets of intervals. Following the musical intent, we have to choose these gestures as variables, $V_1 \dots V_n$.

The domain is given in midi values and closed under minus, e.g. $\{(3, 8), (-3, -8), (6), (-6), (11), (-11)\}$. The starting note SN is fixed, such as the set of allowed values for the notes $Harm$. Then the harmonic constraint is $NC + \sum_{1 \leq j \leq i} V_j \in Harm$ for all $i \leq n$ (analogous to a capacity constraint).

A second constraint forbids local repetitions, $|V_{i+1}| \neq |V_i|$.

A third constraint restricts the number of apparition for each value of the domain. Fixing an integer P_j for some, resp. all, values G_j of the domain (with $\sum_j P_j \leq n$, resp. $\sum_j P_j = n$), the constraint can be written $Card\{i, |V_i| = G_j\} = P_j$. Note that we can switch to a Permut-CSP if all the domain values have a cardinality constraint.

The last constraint restricts the melodic motion of the whole sequence, and is better integrated in the first one as a reduction of $Harm$.

5.3 Harmonization of a rythmical canon

This problem has been proposed by the french composer Georges Bloch (in Une empreinte sonore de la Fondation Bayeler). We start from a given rhythmical canon structure, of length p, with n voices. This canon has the property that exactly one onset is played for each time unit and it is played repetitively. There are only a few possible values of p and n, and we will have to deal with 6 and 12 voices, and periods in a range from 108 to 216.

The goal is to harmonize the canon, *ie* to give a pitch value to each note. The note are sustained, and the chords formed at any time unit vary of exactly one note from one to the next due to the canon structure. We'll write C_i for the chord played at time i. Here, the variables are the notes, and the domain is a fixed ambitus.

The first constraint is to minimize a distance-like function d, called Estrada distance, on the chords. To define d, we need to calculate the texture of the chord, which is the set of all inner intervals of the chord transposed in one octave, plus their completion to 12. For instance the chord $G\sharp 1$, $C\sharp 7$, $F8$ is transposed in $C\sharp$, F, $G\sharp$, with intervals $\{4, 3\}$ and texture $\{4, 3, 5\}$. The distance d is then defined as the number of different values in the textures, precisely the number of notes in the chords minus the number of common intervals in the textures minus 1. An option is the minimize the Estrada distance between the chords and a fixed chord.

The second constraint is to minimize the difference between the virtual fundamentals f between two successive chords. Georges Bloch uses as definition of f the lowest note of a fifth interval in the chord if there is one, if not, the lowest note of a third interval in the chord if there is one, and if not, the lowest note of the chord (some alternative definitions were also used). He then defines another distance d' for values of $diff = |f(C_i) - f(C_{i+1})|$, which matches the circle of fifths, see table below, which depicts the values for the virtual fundamental distance.

$diff$	0	1	2	3	4	5	6	7	8	9	10	11
d'	0	9	7	6	5	2	8	1	4	3	11	10

A third constraint restricts the melodic motion of each voice. It imposes for each voice a common fixed profile, of course with the same time gap as the voices entries, in order to enforce the canon perception.

This CSP is overconstrained and a solver will give as solution the same note everywhere. To avoid this, we have to add some all_different constraints, and there are two possibilities, a vertical one and a horizontal one. Either all the notes of a chord should be different, or two successive notes of a voice should be different. The first option is bad in two ways : it has no real musical meaning (think of the case with 12 voices), and it constrains the CSP too much. The second option is good in two ways : it only adds a small constraint (easy to handle with any kind of filtering), and it ensures a melodic motion for each instrument.

5.4 Adaptive Search Solver in Open Music

This algorithm satisfies many of our goals, mainly because the assignments are directed towards improvement of the configuration, and it uses a notion of distance ("error functions") which is very natural in music. Moreover, it deals naturally with partial solutions. The user can choose a treshold, and ask for the best local minimum reached so far to be printed, provided the global error is lower than this treshold. When there is no exact solution to the CSP, it naturally allows to get partial solutions. For the harmonization problem, the composer even directly used this possibility to compose : he collected the partial solutions, sorted them from the one with largest error to the one with lowest error, and put them in the score (remember that in this problem the canon is played repetitively). This gives a convergence toward the final texture. It is thus very interesting to observe as a feature algorithm has been turned into a new creative process with some natural cognitive and musical by the composer Georges Bloch.

The representation with constraint errors, instead of classical yes/no boolean constraints, gives the possibility to put weight on the constraints, by multiplying their cost by a chosen weight. Thus, playing with the treshold, we can make a distinction between the (mandatory) constraints that must be satisfied and the constraints that are only preferred. Finally, the global constraints are easily handled. Their only feature is their cost functions being constants, which is not a problem for the solver.

Therefore a new implementation of the Adaptive Search method has been implemented in the Open Music system [1] is developped at IRCAM, [2, 3]. More details on this adaptive search solver can be found in [30]. Obviously this implementation, in Common Lisp, is less optimized and efficient than the previously reported direct C implementation. Nevertheless it achieveds its goal of solving in reasonable time the musical problems : a solution is found (or a good quasi-solution is proposed) within a few seconds in general. As those problems have never been tackled previously with other methods, it is difficult to compare and evaluate theses results.

6 Conclusion and Perspectives

We have presented a new heuristic method called adaptive search for solving Constraint Satisfaction Problems by local search. This method is generic, domain-independent, and uses the structure of the problem in terms of constraints and variables to guide the search. It can apply to a large class of constraints (e.g. linear and non-linear arithmetic constraints, symbolic constraints, etc) and naturally copes with over-constrained problems. Preliminary results on some classical CSP problems show very encouraging results, about one or two orders of magnitude faster than competing methods on large benchmarks. Nevertheless, further testing is obviously needed to assess these results.

We have also experimented the Adaptive Search method for solving a new kind of combinatorial problems : musical CSPs. It appears that this constraint solving technique is well-suited to the particular aim of Computer-Assisted Composition. It is an anytime algorithm working by iterative improvement (and can thus begin from any a priori starting point), it can naturally cope with over-constrained problem and produce an approximate solution minimizing an overall error function that can be tuned by the user, and it can produce several solution among which the user can select.

It is also worth noticing that the current method does not perform any planning, as it only computes the move for the next time step out of all possible current moves. It only performs a move if it immediately improves the overall cost of the configuration, or it performs a random move to escape a local minimum. A simple extension would be to allow some limited planning capability by considering not only the immediate neighbors (i.e. nodes at distance 1) but all configurations on paths up to some predefined distance (e.g. all nodes within at distance less than or equal to some k), and then choose to move to the neighbor in the direction of the most promising node, in the spirit of variable-depth search [16] or limited discrepancy search [14]. We plan to include such an extension in our model and evaluate its impact. Further work is needed to assess the method, and we plan to develop a more complete performance evaluation, in particular concerning the robustness of the method, and to better investigate the influence of stochastic aspects and parameter tuning of the method. Future work will include the development of dynamic, self-tuning algorithms.

References

[1] E. Aarts and J. Lenstra (Eds). *Local Search in Combinatorial Optimization*. Wiley, 1997.

[2] Augusto Agon. An environment for computer assisted composition. Ph.D. Thesis, IRCAM/University of Paris 6, 1998.

[3] Gérard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon and Olivier Delerue. Computer Assisted Composition at Ircam : PatchWork & OpenMusic. Computer Music Journal, vol 23 no 3, 1999.

[1] which could be downloaded at URL http://www.ircam.fr/equipes/repmus/OpenMusic/

[4] Gérard Assayag. Applications on Contemporary Music Creation, Esthetic and Technical aspects". 1st Symposium on Music and Computers, Corfu, Greece, 1998.

[5] A. Borning, B. Freeman-Benson and M. Wilson. Constraint Hierarchies. *Lisp and Symbolic Computation*, vol. 5 no. 3, 1992, pp 223-270.

[6] P. Codognet. Adaptive search : preliminary results, In *proceedings ERCIM / CompulogNet Workshop on Constraints*, Venise, Italy, June 2000.

[7] P. Codognet and D. Diaz. Compiling Constraint in `clp(FD)`. *Journal of Logic Programming*, Vol. 27, No. 3, June 1996.

[8] P. Codognet and D. Diaz. Yet Another Local Search Method for Constraint Solving. In *Proc. SAGA01*, 1st International Symposim on Stochastic Algorithms : Foundations and Applications, LNCS 2246, Springer Verlag 2001

[9] A. Davenport, E. Tsang, Z. Kangmin and C. Wang. GENET : a connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *proc. AAAI 94*, AAAI Press, 1994.

[10] D. Diaz and P. Codognet. The implementation of GNU Prolog. In *proc. SAC'OO, 15th ACM Symposium on Applied Computing*. Como, Italy, ACM Press 2000.

[11] F. Glover and M. Laguna. *Tabu Search*, Kluwer Academic Publishers, 1997.

[12] P. Galinier and J-K. Hao. A General Approach for Constraint Solving by Local Search. *draft*, 2001.

[13] J-K. Hao, P. Galinier and M. Habib. Metaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, vol.2 no. 13, 1999, pp 283-324.

[14] W. Harvey and M. Ginsberg. Limited Discrepancy Search. In *proc. IJCAI'95, 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[15] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, vol. 44 (1965), pp 2245-2269.

[16] S. Lin and B. Kerninghan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, vol. 21 (1973), pp 498-516.

[17] Z. Michalewicz and D. Fogel. *How to solve it: Modern Heuristics*, Springer Verlag 2000.

[18] L. Michel and P. Van Hentenryck. Localizer : a modeling language for local search. In *proc. CP'97, 3rd International Conference on Principles and Practice of Constraint Programming*, Linz, Austria, Springer Verlag 1997.

[19] L. Michel and P. Van Hentenryck. Localizer. *Constraints*, vol. 5 no. 1&2, 2000.

[20] L. Michel and P. Van Hentenryck. Localizer++ : an open library for local search. Research Report, Brown University 2001.

[21] S. Minton, M. Johnston, A. Philips and P. Laird. Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, vol. 58, 1992, pp 161-205.

[22] K. Nonobe and T. Ibaraki. A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, vol. 106, 1998, pp 599-623.

[23] G. Pesant and M. Gendreau. A Constraint Programming Framework for Local Search Methods. *Journal of Heuristics*, vol. 5 no. 3, 1999, pp 255-279.

[24] J-F. Puget. A C++ implementation of CLP. In *proc. SPICIS'94*, Singapore, 1994.

[25] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, G. D. Smith. *Modern Heuristic Search Methods*. Wiley, 1996.

[26] V. Saraswat, P. Van Hentenryck et al. Constraint Programming, *ACM Computing Surveys*, vol. 28 no. 4, December 1996.

[27] B. Selman, H. Levesque and D. Mitchell. A new method for solving hard satisfiability problems. In *proc. AAAI'92*, AAAI Press 1992.

[28] B. Selman, H. Kautz and B. Cohen. Noise strategies for improving local search. In *proc. AAAI'94*, AAAI Press 1994.

[29] C. Solnon. Solving permutation problems by ant colony optimization. In *proc. ECAI'2000*, Berlin, Germany, Wiley, 2000.

[30] C. Truchet, P. Codognet and G. Assayag. Visual and Adaptive Constraint Programming in Music. In *proc. ICMC'2001*, International Computer Music Conference, La Habana, Cuba, 2001.

[31] J. P. Walser. *Integer Optimization by Local Search : A Domain-Independent Approach*, LNAI 1637, Springer Verlag 1999.