Constraint programming for optimising satellite validation plans

Caroline Maillet
Astrium, Toulouse, France
ONERA, Toulouse, France
Caroline.Maillet@astrium.eads.net
Caroline.Maillet@onera.fr

Gérard Verfaillie
ONERA, Toulouse, France
Gerard.Verfaillie@onera.fr

Bertrand Cabon
Astrium, Toulouse, France
Bertrand.Cabon@astrium.eads.net

Abstract

Telecommunication satellite payload validation lasts several weeks in thermal vacuum chambers, and mobilise large validation teams. Validation is organised over several thermal transitions, with each transition allowing the validation of a subset of the equipment under different constraints. Minimising the number of transitions is a crucial objective for Astrium. Constraint Programming was used to elaborate a mathematical model of the problem. This model may induce thousands of variables with hundreds of thousands of binary and n-ary constraints. To solve it, specific algorithms were developed, using conflict directed backjumping mechanisms, new adaptive variable weighting heuristics, and restarting strategies. Experiments showed significant gains in terms of quality of the plans produced within a limited computing time.

Industrial context

Astrium is an EADS company in charge of designing and making observation and telecommunication satellites. Telecommunication satellites are geostationary relays which provide telephony, high definition television, and internet services. Orbiting at 36000km from Earth, they receive very low power radio-frequency (RF) signals emitted by ground stations. Received signals are amplified and converted in terms of frequency in order to avoid interferences between received and emitted signals. They are then filtered and amplified again before being emitted to Earth. To perform this task, telecommunication satellite payloads are made of many amplifiers, filters , frequency converters, and connections between them. All these equipment units are duplicated to allow reconfigurations in case of failure. The payload of a given telecommunication satellite depends on its mission. The increase in mission complexity (number of RF signals) results in an increase in payload complexity.

Before launch, thousands of payload units must be validated in conditions close to space environment. To do so, most of the validation operations are performed in thermal vacuum chambers under extreme temperature conditions. Validation is organised over several thermal transitions with an increasing or decreasing temperature. With each thermal transition, is associated a payload configuration. Each transition allows some of the validation requirements to be satisfied. All transitions must cover all the validation requirements defined by Astrium.

Validation operations associated with one thermal transition last more than a dozen hours. Validating the whole payload mobilises large validation teams present 24 hours 24 over several weeks. Hence, minimising the number of thermal transitions that allow all the validation requirements to be covered is a crucial objective for Astrium.

Problem of optimisation of telecommunication payload validation plans

With each thermal transition, is associated a payload configuration. With each configuration is associated a set of paths. A path represents an RF signal that goes through the payload from an reception antenna to an emission one, using a set of units (amplifiers, filters, converters, switches, and connections between them). Switches are configurable units that allow RF signals to be routed to specific units. For example, Fig.1 shows a part of the red path which uses switches 3508 and 4227, amplifier 5005, switches 7988 and 7990, and filter CH17. During a thermal transition, all the paths of the associated configuration and thus all the units present on these paths are concurrently validated.

However, the set of paths associated with a transition must meet some constraints.

Fisrt, all the paths must be compatible with each other in terms of switch positions. For example on Fig.1, red and blue paths are compatible because they do not share any unit. Green and blue path are compatible as well because the only common unit is switch 3507, used in the same position by both paths. On the contrary, red and green paths are incompatible because both use switch 7990 in different positions.

Second, due to thermal and measure constraints, each transition has a limited capacity: the number of associated paths must be less than a maximum number.

Third, due to local thermal constraints, some payload areas are subject to thermal limitations: the number of active units must be less than a maximum number. For example, only two units among amplifiers 5005, 5006, 5007, 5008, and 5009 of Fig.1 can be used by paths during a decreasing thermal transition.

Finally, some paths must be compulsorily validated and

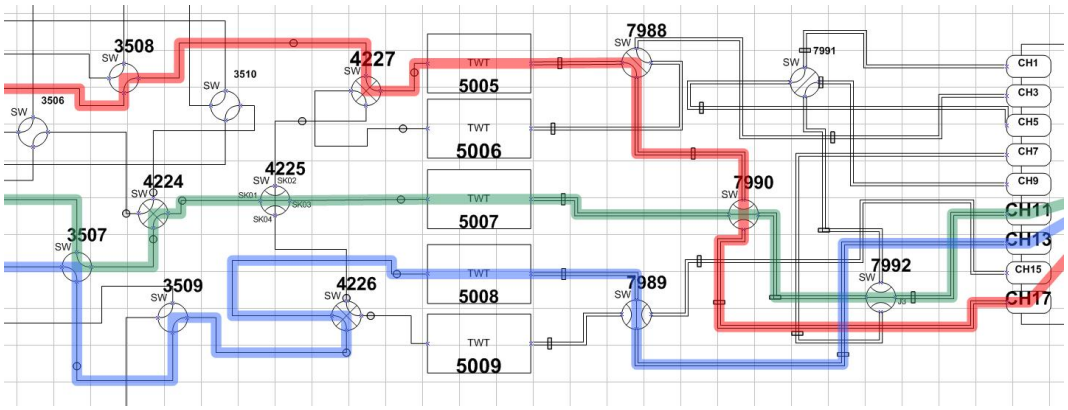

Figure 1: Small part of a satellite telecommunication payload

thus must be present in at least one transition. For other paths, some transitions are forbidden.

Validation requirements must be covered by the set of thermal transitions. They are of different types.

The most usual ones enforce the validation of a unit in at least one thermal transition. This requires that at least one path going through this unit be validated in at least one thermal transition. For example on Fig.1, using red or green path allows switch 7990 to be validated. Globally, all the payload units must be validated in at least one thermal transition.

Other requirements enforce a minimum number (greater than one) of validation tests for some units.

Others enforce a unit to be validated under some thermal conditions. This reduces the set of usable transitions.

Finally, others enforce a unit to be validated under some frequency conditions. This reduces the set of usable paths. For example on Fig.1, if switch 7990 must be validated using a high frequency, green path which goes through filter CH11 of too low frequency does not allow this requirement to be satisfied. On the contrary, red path which goes through CH17 of high frequency allows it to be satisfied.

It must be stressed that the presence of a path in a transition allows a set of requirements associated with path units to be covered.

Two optimisation criteria can be considered when building validation plans:

- the minimisation of the number of used thermal transitions allowing all the validation requirements to be satisfied,
- the maximisation of the number of satisfied validation requirements using a given number of thermal transitions.

Since the number of necessary transitions is generally about a dozen, it is possible to minimise it manually. This is why, our study focused on the second criterion: maximisation of the number of satisfied validation requirements using a given number of thermal transitions.

The problem data is the following:

- a number of usable transitions (about a dozen),
- a set of usable paths (several thousand; this set is precomputed by an semi-automatic tool),
- a set of validation requirements (several thousand),
- a set of constraints on used paths:
  - binary constraints of compatibility between paths (several hundreds of thousands),
  - n-ary constraints of capacity and thermal limitation on transitions (several hundred),
  - unary constraints of restriction on path assignments (several hundred).

The problem output is a validation plan which assigns each path p the transition in which p is present, possibly null if p is present in no transition. This output must be produced in few minutes only.

As far as we know, this application has not been studied yet by the scientific community. The problem is closed to a knapsack problem with multiple sacks, multiple sack dimensions, and conflicts (Kellerer, Pferschy, and Pisinger 2004; Epstein, Levin, and Van Stee 2007). Thermal transitions are sacks and paths are objects. Conflicts are binary constraints of incompatibility between paths and unary constraints of restriction on path assignments. Dimensions are n-ary contraints of capacity and thermal limitation on transitions. However, usually in knapsack problems, object utilities are assumed to be additive. This is not the case in our problem: path utility is defined by the set of validation requirements it can satisfy. A validation requirement may be satisfied by several paths. These utilities are not additive. This is why we built a mathematical model dedicated to the problem of optimisation of the validation plans of telecommunication payloads.

Mathematical model

We first elaborated a mathematical model in Constraint Programming (CP) (Dechter 2003) using the CP Optimizer tool

(IBM ILOG). This model associates with each path p a variable which represents the transition in which p is present. Its domain of value is the set of usable transitions to which the null value is added to represent the fact that p may be not selected.

Data

- Number of paths: $PATH$
- Number of units: $UNIT$
- Presence of units in paths:

$$UNIT\_PATH[u][p], \forall u \in \{1, \ldots, UNIT\}, \\ \forall p \in \{1, \ldots, PATH\}$$

- Number of thermal transitions: $TRANS$
- Transition capacities in terms of number of paths:

$$CAP[t], \forall t \in \{1, \ldots, TRANS\}$$

- Compatibilities between paths:

$$COMP[p][p'], \forall p, p' \in \{1, \ldots, PATH\}$$

- Restrictions on path assignments:

$$RESTR[p][t], \forall p \in \{1, \ldots, PATH\}, \\ \forall t \in \{0, \ldots, TRANS\}$$

- Number of validation requirements: REQ
- Minimum number of paths to be selected to satisfy validation requirements:

$$MIN\_REQ[r], \forall r \in \{1, \ldots, REQ\}$$

- Usable paths to satisfy validation requirements:

$$PATH\_REQ[p][r], \forall p \in \{1, \ldots, PATH\}, \\ \forall r \in \{1, \ldots, REQ\}$$

- Usable transitions to satisfy validation requirements:

$$TRANS\_REQ[t][r], \forall t \in \{1, \ldots, TRANS\}, \\ \forall r \in \{1, \ldots, REQ\}$$

- Number of thermal limitations: LIM
- Maximum number of active units per transition to satisfy thermal limitations:

$$MAX\_LIM[l], \forall l \in \{1, \ldots, LIM\}$$

- Units involved in thermal limitations:

$$UNIT\_LIM[u][l], \forall u \in \{1, \ldots, UNIT\}, \\ \forall l \in \{1, \ldots, LIM\}$$

- Transitions involved in thermal limitations:

$$TRANS\_LIM[t][l], \forall t \in \{1, \ldots, TRANS\}, \\ \forall l \in \{1, \ldots, LIM\}$$

Variables

For each path, the used thermal transition or the null value to represent the absence of selection:

$$aff[p] \in \{0, \ldots, TRANS\}, \forall p \in \{1, \ldots, PATH\}$$

Expressions

- For each unit, the fact that it is used or not:

$$\forall u \in \{1, \ldots, UNIT\}, \forall t \in \{1, \ldots, TRANS\}, \\ usedUnit[u][t] = \\ \Big(\sum_{p=1}^{PATH/UNIT\_PATH[u][p]=1} (aff[p] = t) \Big) > 0$$

- For each validation requirement, the number of selected paths satisfying it:

$$\forall r \in \{1, \ldots, REQ\}, \\ testReq[r] = \sum_{p=1,t=1}^{\substack{PATH/PATH\_REQ[p][r]=1, \\ TRANS/TRANS\_REQ[t][r]=1}} (aff[p] = t)$$

- For each validation requirement, its satisfaction level:

$$\forall r \in \{1, \ldots, REQ\}, \\ satReq[r] = \min(testReq[r], MIN\_REQ[r])$$

- Criterion to be optimised (sum of the satisfaction levels over all validation requirements):

$$obj = \sum_{r=1}^{REQ} satReq[r]$$

Objective

$$\text{maximise } obj$$

Constraints

Unary constraints Restriction on path assignments:

$$\forall p \in \{1, \ldots, PATH\}, \forall t \in \{0, \ldots, TRANS\} / \\ RESTR[p][t] = 0, \quad aff[p] \neq t$$

Binary constraints Compatibility between paths in the same transition:

$$\forall p, \forall p' \in \{1, \ldots, PATH\} / COMP[p][p'] = 0, \\ \Big(aff[p] \neq aff[p'] \Big) \| \\ \Big((aff[p] = 0) \&\& (aff[p'] = 0) \Big)$$

N-ary constraints

- Capacity of each transition:

$$\forall t \in \{1, \ldots, TRANS\},$$
$$\sum_{p=1}^{PATH} (aff[p] = t) \leq CAP[t]$$

- Thermal limitations:

$$\forall l \in \{1, \ldots, LIM\}, \forall t \in \{1, \ldots, TRANS\} / TRANS\_LIM[t][l] = 1,$$
$$\sum_{u=1}^{UNIT/UNIT\_LIM[u][l]=1} (usedUnit[u][t] = 1) \leq MAX\_LIM[r]$$

This mathematical model was validated using CP Optimizer (IBM ILOG) on instances associated with satellites in production. On small instances, it allows optimal solutions to be quickly produced and sometimes optimality to be proven. Another model, involving boolean variables, was elaborated in Integer Linear Programming (Nemhauser and Wolsey 1988). This model was validated using CPLEX (IBM ILOG). On the same instances, it allows more often optimality to be proven at the price of higher computing times. However, on instances of higher size, these generic solvers (CP Optimizer and CPLEX) encounter problems of memory overflow. This is why we decided to develop a specific algorithm. To avoid memory overflow, this algorithm manages constraints implicitly thanks to bit vector operations. It also uses numerous CP techniques.

Algorithms

The specific algorithm developed by Astrium to solve this optimisation problem, is a depth-first tree search algorithm, including:

- dynamic variable and value ordering heuristics,
- a bound computing updating the objective variable (obj) domain,
- a propagation mechanism based on forward checking (Haralick and Elliott 1980),
- a chronological backtracking mechanism (BT) (Bitner and Reingold 1975).

These mechanisms have been then improved, using:

- intelligent backtracking mechanisms: backjumping (BJ) (Gaschnig 1979), conflict directed backjumping (CBJ) (Prosser 1993; Schiex and Verfaillie 1994) with production of explanations for binary and n-ary constraints,
- dynamic specific adaptive variable ordering heuristics: learning of constraint impacts on variables inspired from weighted degree (wdeg) (Boussemart et al. 2004) and from wvar (Karoui et al. 2007), use of the last conflict (Lecoutre et al. 2009),
- restarting mechanisms (Rs): restart with randomised heuristics (Gomes, Selman, and Kautz 1998; Walsh 1999), adaptive restart using impacts (Grimes 2008), and randomised adaptive restart.

Even if the arc consistency maintenance mechanism (MAC) (Sabin and Freuder 1994) is generally very efficient, it was not implemented in this specific algorithm. Because of the presence of the null value in the domain of most variables, the problem is arc consistent and remains arc consistent at most nodes of the search tree and MAC cannot remove values from domains.

Backtracking mechanisms

Three backtracking mechanisms were implemented. For each of them, a conflict is an inconsistency between a variable assignment and previous variable assignments. A failure is the fact that all values of a variable are in conflict. In case of failure, it is necessary to backtrack.

Chronological backtracking (BT) (Bitner and Reingold 1975) consists in coming back to the last variable assignment.

Backjumping (BJ), introduced in (Gaschnig 1979), consists in coming back to the assignment of the most recent variable involved in the last conflict that led to failure. In case of nested failures, the first backtrack is a backjump, the others are chronological backtracks.

Conflict directed backjumping (CBJ), introduced in (Prosser 1993) and in (Schiex and Verfaillie 1994), memorises conflict explanations. A conflict explanation is the set of previously assigned variables that are responsible for conflict. In case of failure for a variable v, a backjump is triggered to the most recent variable involved in the conflict explanations of v. CBJ allows multiple nested backjumps.

Variable ordering heuristics

Dynamic specific variable ordering heuristics In CP, dynamic variable ordering heuristics allow the next variable to assign to be chosen according to the previous variable assignments. Usually, these heuristics follow the first-fail principle: choose the variable that allows failure to be detected as soon as possible. However in our problem, it is important to handle differently variables that do not have the null value in their domain and variables that do have it. The first ones represent paths that must be compulsorily selected in one of the transitions. The second ones represent paths that may be selected or not. For the first ones, the first-fail principle seems to be relevant. On the contrary, for the second ones, an opposite first-success principle seems to be more relevant.

As a consequence, three variable types are identified and ordered in hierarchical way:

1. variables with only one value in their domain are assigned first,
2. variables without null value in their domain are then assigned following the first-fail principle,
3. variables with null value in their domain are finally assigned following the first-success principle.

For variables without null value in their domain, the classical $MinDomain$ dynamic heuristic is relevant: choose a variable of smallest current domain size (Haralick and Elliott 1980).

For variables with null value in their domain, we can draw inspiration from knapsack heuristics. The most classical consists in ordering statically objects according to decreasing values of the ratio between utility and weight (Kellerer, Pferschy, and Pisinger 2004). In our problem, it is difficult to define what utility and weight of a path are. For this reason, variables are dynamically ordered according to decreasing values of the immediate impact on the objective: $MaxObjective$ heuristic. For a variable v, this impact is the maximum value of the increase in the requirement satisfaction level that is possible to get by assigning v a value from its current domain.

These orderings can be refined to take into account the initial compatibility between paths: for each path, the number of paths with which it is compatible. This information allows the least (resp. the most) compatible variable to be chosen: $MinCompatible$ (resp. $MaxCompatible$) heuristic.

The resulting hierarchical dynamic specific heuristic (HDS) is the following:

1. variables with only one value in their domain are assigned first,
2. variables without null value in their domain are then assigned following the $MinDomain$ heuristic and, in case of equality, the $MinCompatible$ one,
3. variables with null value in their domain are finally assigned following the $MaxObjective$ heuristic and, in case of equality, the $MaxCompatible$ one.

A weakness of the HDS heuristic is that it does not take into account n-ary constraints. This is why we looked for some refinements.

Dynamic specific adaptive variable ordering heuristics In CP, the first-fail $MinDomain$ heuristic is usually refined by taking into account the degree deg of each variable (number of constraints in which it is involved), its dynamic degree $ddeg$ (number of active constraints in which it is involved) (Bessière and Régin 1996), or its weighted degree $wdeg$ (sum of the weights of the active constraints in which it is involved) (Boussemart et al. 2004) . A constraint is active if at least one of its variables has not been assigned yet. Constraint weights are learned during search. More precisely, the weight of a constraint c is incremented each time the propagation of c leads to a failure. The $MinDomain$ heuristic is then replaced by the $Min\frac{Domain}{deg}$, $Min\frac{Domain}{ddeg}$, or $Min\frac{Domain}{wdeg}$ heuristic.

However these refinements use a large amount of memory. They are incompatible with our implementation which manages constraints implicitly to avoid memory overflow. It would be better to associate weights with variables. This is what is proposed in (Karoui et al. 2007) which introduced the $Min\frac{Domain}{wvar}$ heuristic. Variable weights ($wvar$) are learned during search. More precisely, the weight of a variable v is incremented each time a failure occurs on v.

However, this heuristic does not take into account failure explanations. This is why we propose a new adaptive heuristic based on the learning of failure explanations. As with $wvar$, variable weights, now called $wcvar$ for weighted culprit variable, are learned during search. More precisely, the weight of a variable v is incremented each time v is involved in the last conflict that leads to a failure. The resulting heuristic is referred to as $Min\frac{Domain}{wcvar}$.

As a result, the HDS heuristic is refined as follows and becomes the HDSA_WCVar heuristic:

1. variables with only one value in their domain are assigned first,
2. variables without null value in their domain are then assigned following the $Min\frac{Domain}{wcvar}$ heuristic and, in case of equality, the $MinCompatible$ one,
3. variables with null value in their domain are finally assigned following the $Max\frac{Objective}{wcvar}$ heuristic and, in case of equality, the $MaxCompatible$ one.

The $Max\frac{Objective}{wcvar}$ heuristic is close to the classical knapsack $Max\frac{value}{weight}$ heuristic, if we interpret $wcvar$ as being the weight of paths.

Heuristics based on the last conflict A refinement of usual CP heuristics has been proposed in (Lecoutre et al. 2009). In case of failure, after backtracking, it consists in assigning first the variable on which the failure occurred and then in following standard heuristics.

According to this idea, the HDS heuristic is refined as follows and becomes the HDSA_LC heuristic:

1. the previous failure variable is assigned first,
2. variables with only one value in their domain are then assigned,
3. variables without null value in their domain are then assigned following the $MinDomain$ heuristic and, in case of equality, the $MinCompatible$ one,
4. variables with null value in their domain are finally assigned following the $MaxObjective$ heuristic and, in case of equality, the $MaxCompatible$ one.

Combination The previous two heuristics can be combined, resulting in the following HDSA_WCVar_LC heuristic:

1. the previous failure variable is assigned first,
2. variables with only one value in their domain are then assigned,
3. variables without null value in their domain are then assigned following the $Min\frac{Domain}{wcvar}$ heuristic and, in case of equality, the $MinCompatible$ one,
4. variables with null value in their domain are finally assigned following the $Max\frac{Objective}{wcvar}$ heuristic and, in case of equality, the $MaxCompatible$ one.

Value ordering heuristics

As for value ordering heuristics, values in variable domains are dynamically ordered according to decreasing values of the immediate impact on the objective.

Restart mechanisms

Even if intelligent backtracking mechanisms and dynamic specific adaptive heuristics are used, first assignments are crucial due to the huge number of variables. Restart mechanisms allow first assignments to be undone. After a given number of backtracks, search is started again from scratch. Restart after restart, the number of allowed backtracks is geometrically increased (Walsh 1999). The increased number of allowed backtracks guarantees algorithm completeness. However, it is necessary not to explore twice the same part of the search tree. For that, it suffices to modify the variable ordering. We implemented several types of restart.

The randomised restart (Rs_Rand) introduces noise in heuristic values (Gomes, Selman, and Kautz 1998). More precisely, before making choices, a small positive or negative noise is randomly added to the heuristic value of each variable. This mechanism allows a random search to be performed around the initial heuristic. It is thus possible to randomise the four previously defined heuristics to get four randomised versions: HDS_Rs_Rand, HDSA_WCVar_Rs_Rand, HDSA_LC_Rs_Rand, and HDSA_WCVar_LC_Rs_Rand. The first one is a randomised restart, the following three ones are adaptive randomised restarts.

Another way of guaranteeing search diversity is to use learning. An adaptive restart using weighted degrees ($wdeg$) was introduced in (Grimes 2008). As weighted degrees are modified during search and exploited by variable ordering heuristics, variable orderings differ restart after restart. We propose a similar restart based on $wcvar$, we refer to as HDSA_WCVar_Rs. Another adaptive restart was implemented on top of the combined HDSA_WCVar_LC heuristic. We refer to it as HDSA_WCVar_LC_Rs.

All these mechanisms (backtracking, variable ordering, and restart) were experimented on instances associated with satellites in production in Astrium.

Experiments

Experiments were carried out on five telecommunication payloads, referred to as SatA, SatB, SatD, SatE, and SatF. In order to produce a large number of different instances, several dozen instances were randomly derived from these initial instances. To do that, validation requirements taken into account by each instance were randomly fired from a set of requirements. Instances of type SatA, and SatB, involve several hundred variables, those of type SatD, SatE, and SatF involve several thousand. Instances of type SatA and SatB are simpler than the others in terms of number of constraints and requirements to be satisfied. As an example, instances of type SatE involve 5 transitions, $7,000$ paths, $2,400$ validation requirements, $13,000,000$ binary constraints and 135 n-ary constraints. On all these instances, twelve algorithms were compared. All of them perform a depth-first tree search and use a forward checking propagation mechanism. They differ according to the used backtracking mechanisms, variable ordering heuristics, and restart mechanisms. These twelve algorithms are the following:

- chronological backtrack and dynamic specific heuristic (BT_HDS),
- backjumping and dynamic specific heuristic (BJ_HDS),
- conflict directed backjumping (CBJ) and dynamic specific heuristic (CBJ_HDS),
- CBJ and dynamic specific adaptive heuristic with $wcvar$ (CBJ_HDSA_WCVar),
- CBJ and dynamic specific heuristic with last conflict (CBJ_HDSA_LC),
- CBJ and dynamic specific adaptive heuristic with $wcvar$ and last conflict (CBJ_HDSA_WCVar_LC),
- CBJ, dynamic specific heuristic, and randomised restart (CBJ_HDS_Rs_Rand),
- CBJ, dynamic specific adaptive heuristic with $wcvar$, and restart (CBJ_HDSA_WCVar_Rs),
- CBJ, dynamic specific adaptive heuristic with $wcvar$, and randomised restart (CBJ_HDSA_WCVar_Rs_Rand),
- CBJ, dynamic specific heuristic with last conflict, and randomised restart (CBJ_HDSA_LC_Rs_Rand),
- CBJ, dynamic specific adaptive heuristic with $wcvar$ and last conflict, and restart (CBJ_HDSA_WCVar_LC_Rs),
- CBJ, dynamic specific adaptive heuristic with $wcvar$ and last conflict, and randomised restart (CBJ_HDSA_WCVar_LC_Rs_Rand).

These twelve algorithms were also compared with the CP Optimizer solver (IBM ILOG) for the instances that do not lead to memory overflow. This generic solver was used in two modes:

- depth-first (CP_Optimizer),
- restart (CP_Optimizer_Rs).

For each instance and each algorithm, we measured the number of satisfied test requirements of the best validation plan produced within a limited time : one minute for the simplest instances of type SatA and SatB, and of five minutes for the others. For each instance type and each algorithm, a mean of these measures is calculated. These means are then normalised between 0 and 1 to give a quality index. Since CP Optimizer does not produce any results for instance type SatD, and SatE, these normalisations are calculated from specific algorithm results. Thereby, the quality index of the best algorithm is 1, the worst is 0. Quality indices of CP Optimizer results are also calculated from specific algorithm results. For this reason, CP optimizer quality indices could be greater than 1 if they provide better results than specific algorithms, lower than 0 if they provide worst results. Tab.1 gathers quality indices of results obtained by the twelve specific algorithms and by the generic solver for each instance type. Fig.2 shows the mean value of these quality indices over the five instance types for specific algorithms.

Algorithm	SatA	SatB	SatD	SatE	SatF
BT_HDS	0.76	0	0	0	0.46
BJ_HDS	0.76	0.03	0.01	0.43	0.47
CBJ_HDS	0.81	0.14	0.08	0.63	0.48
CBJ_HDSA_WCVar	0.81	0.09	0.08	0.51	0.49
CBJ_HDSA_LC	0.81	0.14	0.08	0.65	0.48
CBJ_HDSA_WCVar_LC	0	0.07	0.08	0.53	0.49
CBJ_HDS_Rs_Rand	0.93	1	0.53	0.79	0.05
CBJ_HDSA_WCVar_Rs	0.9	0.74	1	1	1
CBJ_HDSA_WCVar_Rs_Rand	0.13	0.71	0.08	0.55	0.15
CBJ_HDSA_LC_Rs_Rand	1	1	0.44	0.86	0
CBJ_HDSA_WCVar_LC_Rs	0.9	0.72	0.97	0.99	0.99
CBJ_HDSA_WCVar_LC_Rs_Rand	0.13	0.71	0.08	0.52	0.14
CP_Optimizer_DF	-23.5	-8.47	NA	NA	-17.94
CP_Optimizer_Rs	0.36	1.26	NA	NA	0.7

Legend: NA grey cells point out non applicable experiments.

Table 1: Quality index obtained by each algorithm for each instance type.

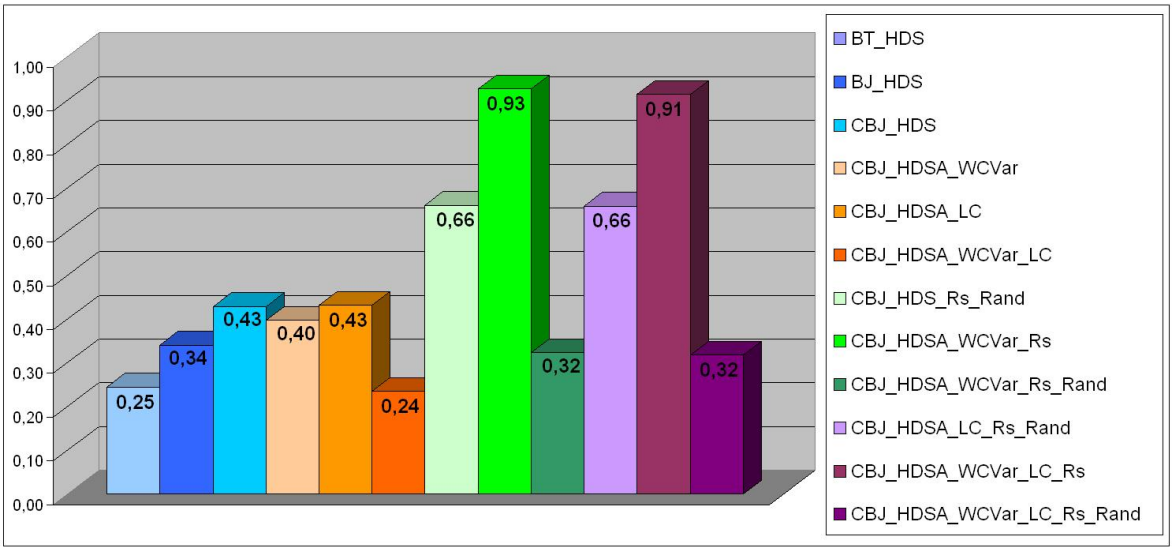

Figure 2: Mean quality index obtained by each algorithm.

Results for backtracking mechanisms

Tab.1 shows that backjumping (BJ_HDS) improves the solution quality with regard to chronological backtrack (BT_HDS). A 0.43 increase in the quality index is observed on instances of type SatE. Whatever the instance type is, CBJ (CBJ_HDS) improves further the solution quality. A 0.63 gain is obtained on instances of type SatE. These results are verified in Fig.2. Whereas backjumping (BJ_HDS) results in a mean index of 0.34, CBJ (CBJ_HDS) brings a mean index of 0.43. For this reason, variable ordering heuristics and restart mechanisms were then tested using only CBJ.

Results for variable ordering heuristics

The solutions produced by the adaptive heuristic with $wcvar$ and without restart (CBJ_HDSA_WCVar) are not of better quality than those produced by the reference heuristic (CBJ_HDS), except for instances of Type SatE whose index increase slightly of 0.01. The heuristic based on the last conflict (CBJ_HDSA_LC) produces small gain 0.02 for instances of type SatE with regard to the reference heuristic. The combined adaptive heuristic (CBJ_HDSA_WCVar_LC) decreases the solution quality by almost 0.20 on average with regard to the reference heuristic. For instances of type SatA, this combined heuristic produces solutions of worse quality than using the reference algorithm (BT_HDS). These learning mechanisms do not seem to be efficiently used without restart.

Results for restart mechanisms

Among the specific algorithms, Tab.1 shows that the best results are obtained:

- for instances of type SatA, by the randomised restart with last conflict (CBJ_HDSA_LC_Rs_Rand),

- for instances of type SatB, by both the randomised restart (CBJ_HDS_Rs_Rand) and the randomised restart with last conflict (CBJ_HDSA_LC_Rs_Rand),
- for instances of type SatD, SatE, and SatF by the adaptive restart with $wcvar$ (CBJ_HDSA_WCVar_Rs).

On average, the adaptative randomised restart with $wcvar$, and the adaptive randomised restart with $wcvar$ and last conflict (CBJ_HDSA_WCVar_LC_Rs_Rand) do not improve results with regard to CBJ_HDS. These restarts makes results even worse for instances of type SatA with regard to BT_HDS. On average, use of the last conflict (CBJ_HDSA_WCVar_LC_Rs_Rand or respectively CBJ_HDSA_WCVar_LC_Rs) decreases the quality index with regards to the one obtained by the randomised restart (CBJ_HDS_Rs_Rand) or respectively by the adaptive restart with $wcvar$ (CBJ_HDSA_WCVar_Rs). Even if CBJ_HDSA_WCVar_LC_Rs_Rand founds the best results for instances of type SatA and SatB. On average, the adaptive restart with $wcvar$ (CBJ_HDSA_WCVar_Rs) produces the best solutions.

This analysis proves the positive impact of CBJ and restart mechanisms for this optimisation problem with a large benefit for the adaptive restart with $wcvar$. This learning $wcvar$ based on weight culprit variable, turns out to be very successful when it is combined with the restart mechanism.

Results obtained with the generic CP Optimizer solver

The generic CP Optimizer solver (IBM ILOG) was used as a reference on instances of type SatA, SatB, and SatF. It could not be used on other instances due to memory overflow.

Tab.1 shows that CP Optimizer using a depth-first strategy (CP_Optimizer) gives much lower quality results than specific algorithms. As our specific algorithms use the same depth-first strategy, these results prove the positive impact of CBJ and of specific variable ordering heuristics.

CP Optimizer using a restart strategy (CP_Optimizer_Rs) is more efficient. It allows better solutions to be produced for instances of type SatB. Nevertheless, it is far from the best solution for instances of type SatA, and SatF. To obtain better results for instances of type SatB with specific algorithms, it would be valuable to implement other restart mechanisms based on the learning of the reduction of the search tree size. This type of learning, proposed in (Refalo 2004), consists in learning the impact of each variable assignment on the size of the search tree.

Conclusion

For this new application domain, the mathematical model we proposed allows the objective of optimisation of the telecommunication payload validation plans to be expressed. However this model induces a great number of variables (many thousands) and of constraints (many hundreds of thousands). Its solving using generic tools may be impossible due to memories problem when generating models.

Only specifically implemented algorithms allow all the instances to be solved. In these algorithms, CBJ and restart mechanisms are valuable. For the most complex instances, the new adaptive variable ordering heuristic with $wcvar$ is efficient when combined with restart. For one simpler instances type, implemented restarts are not as efficient as CP Optimizer when using a restart mode. It would be interesting to develop adaptive restart mechanisms based on the learning of the reduction of the search tree size (Refalo 2004).

In another way, this optimisation problem could be solved using local search algorithms such as min conflict (Minton et al. 1990), tabu search (Glover and Laguna 1993), or simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983).

References

Bessière, C., and Régin, J. 1996. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems. In *Proc. of CP'96*, 61–75.

Bitner, J., and Reingold, E. 1975. Backtrack Programming Techniques. *Communications of the ACM* 18(11):651–656.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting Systematic Search by Weighting Constraints. In *Proc. of ECAI'04*, 146–150.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann Publishers Inc.

Epstein, L.; Levin, A.; and Van Stee, R. 2007. Multi-dimensional Packing with Conflicts. In *Proc. of FCT'07*, 288–299.

Gaschnig, J. G. 1979. *Performance measurement and analysis of certain search algorithms*. Ph.D. Dissertation.

Glover, F., and Laguna, M. 1993. Tabu Search. In *Modern Heuristic Techniques for Combinatorial Problems*. 70–141.

Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proc. of AAAI'98*, 431–437.

Grimes, D. 2008. A Study of Adaptive Restarting Strategies for Solving Constraint Satisfaction Problems. In *Proc. of AICS'08*.

Haralick, R. M., and Elliott, G. L. 1980. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence Journal* 14(3):263–313.

IBM. ILOG. IBM ILOG OPL Development Studio. http://www-01.ibm.com/software/integration/optimization/opl-dev-studio/.

Karoui, W.; Huguet, M.-J.; Lopez, P.; and Naanaa, W. 2007. YIELDS: A Yet Improved Limited Discrepancy Search for CSPs. In *Proc. of CPAIOR'07*.

Kellerer, H.; Pferschy, U.; and Pisinger, D. 2004. *Knapsack Problems*. Springer.

Kirkpatrick, S.; Gelatt, C.; and Vecchi, M. 1983. Optimization by Simulated Annealing. *Science* 220(4598):671–680.

Lecoutre, C.; Saïs, L.; Tabary, S.; and Vidal, V. 2009. Reasoning from Last Conflict(s) in Constraint Programming. *Artificial Intelligence Journal* 173:1592–1614.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1990. Solving large-scale constraint satisfaction and

scheduling problems using a heuristic repair method. In *Proc. of AAAI'90*, 17–24.

Nemhauser, G., and Wolsey, L. 1988. *Integer and Combinatorial Optimization*. John Wiley & Sons.

Prosser, P. 1993. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence* 9:268–299.

Refalo, P. 2004. Impact-Based Search Strategies for Constraint Programming. In *Proc. of CP'04*, 557–571.

Sabin, D., and Freuder, E. C. 1994. Contradicting conventional wisdom in constraint satisfaction. In *Proc. of ECAI'94*, 125–129.

Schiex, T., and Verfaillie, G. 1994. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools* 3(2):187–207.

Walsh, T. 1999. Search in a Small World. In *Proc. of IJCAI'99*, 1172–1177.