ONDERZOEKSRAPPORT NR 9220

A Synthesizing Algorithm and Interval Arithmetic as a Basis for Integer Linear Programming

by

Nico VANDAELE

D/1992/2376/26

A SYNTHESIZING ALGORITHM AND INTERVAL ARITHMETIC AS A BASIS FOR INTEGER LINEAR PROGRAMMING*

NICO VANDAELE

*

- In cooperation with CIBRE, Center for Interdisciplinary Business Research, De Beriotstraat 32, 3000 Leuven, 016/28 37 02.
- FKFO, Just-in-time project

ABSTRACT

In the area of integer programming, it is still very intriguing to explore new ways of thinking. Here we propose a fairly unknown approach composed of elements in the domain of artificial intelligence and combinatorics. From the latter, we retain an as yet not fully explored technique called 'preprocessing'. The paper presents the general reasoning by means of a representative problem. A more theoretical approach is then taken to state the synthesizing algorithm formally, which will then be applied to integer linear programming (ILP). It will turn out that the synthesizing algorithm is not very suitable for solving ILP problems. Then the basics of the algorithm are retained but are supplemented by another procedure, called REFINE. In combination with some established methods, it will lead to a suitable method for solving ILP. For purposes of comparison, some examples were taken from the literature and solved with four different methods. At least for these problems, the method presents interesting results. Another example concludes our expose. A primer on interval arithmetic is provided together with written out examples in the appendix .

1. INTRODUCTION

The problem considered is the constraint satisfaction problem (Mackworth,1990). This implies a set of variables $X_1,X_2,...,X_n$ and a number of constraints on subsets of these variables. The n-dimensional points which comply with all constraints simultaneously constitute the set of feasible solutions. The constrained optimization problem then deals with the optimization of a criterion function over this set of points. Under these general definitions resort some very wellknown problems like mathematical programming problems, combinatorics, a.o. Subsequently a method is presented to synthesize all the constraints in one synthesizing constraint which corresponds to the feasible region of the whole problem. Please note that this paper only considers discrete problems. Basically the method uses an iterated reduction of possibilities through constraint propagation, which means that constraints are used to gradually reduce the different universes in order to obtain the final feasible set (Davis,1987).

The usability of constraint propagation is twofold. On the one hand it is used to preprocess other satisfaction and optimization methods and on the other hand, occasionally small problems can be solved in the preprocessing phase or by combining preprocessing with some further enumeration technique (Guignard and Spielberg,1977). Nevertheless, the main purpose is to prepare the formulation of a problem in order to give it then to a more sophisticated algorithm. Up til now, a method, which in the sequel will be called the 'REFINE' procedure, has been used to preprocess continuous satisfaction and optimization problems (Lodwick, 1989). To the best of our knowledge, the opportunities stemming from the use of the REFINE procedure for general integer (linear) programming have not yet been investigated. The closest research area consists of the papers discussing propagation in (0-1) programs (Crowder, Johnson and Padberg, 1983; Johnson, Kostreva and Suhl, 1985; Guignard and Spielberg, 1981; Zionts, 1972). Ultimately, Guignard and Spielberg (1977) tried to generalize the ideas from O-1 problems to mixed integer problems.

2. INTRODUCTORY EXAMPLE

Let's take the map of Belgium which consists of nine provinces and ask ourselves if it is possible to colour the map with three colours while avoiding confusion caused by two adjacent provinces having the same colour. It must be noted that we are presently only concerned with checking the solution and not by the related optimization problem of minimizing the number of colours needed to colour the map of Belgium. In figure 1 the provinces have been numbered. These numbers are used to construct a network in which the nodes (numbers) represent the provinces and the edges the adjacency relation : '... is adjacent to ...'. Note that not all of the nodes are connected, so this is an instance of the general graph colouring problem. Of course there's nothing special about this problem and it has been described entirely in literature. For instance Papadimitriou and Steiglitz (1982) classify it as NP-complete. The reason why three colours are proposed is the following : in Belgium there is no point where more than three provinces meet each other. This gives us an indication of the lower bound on the number of colours that have to be used to colour the map. It is clear that only the relative order is important and not the absolute value of the colours. If one feasible order is found then, given three colours, immediately five other feasible orderings can be obtained (3.2.1=6).

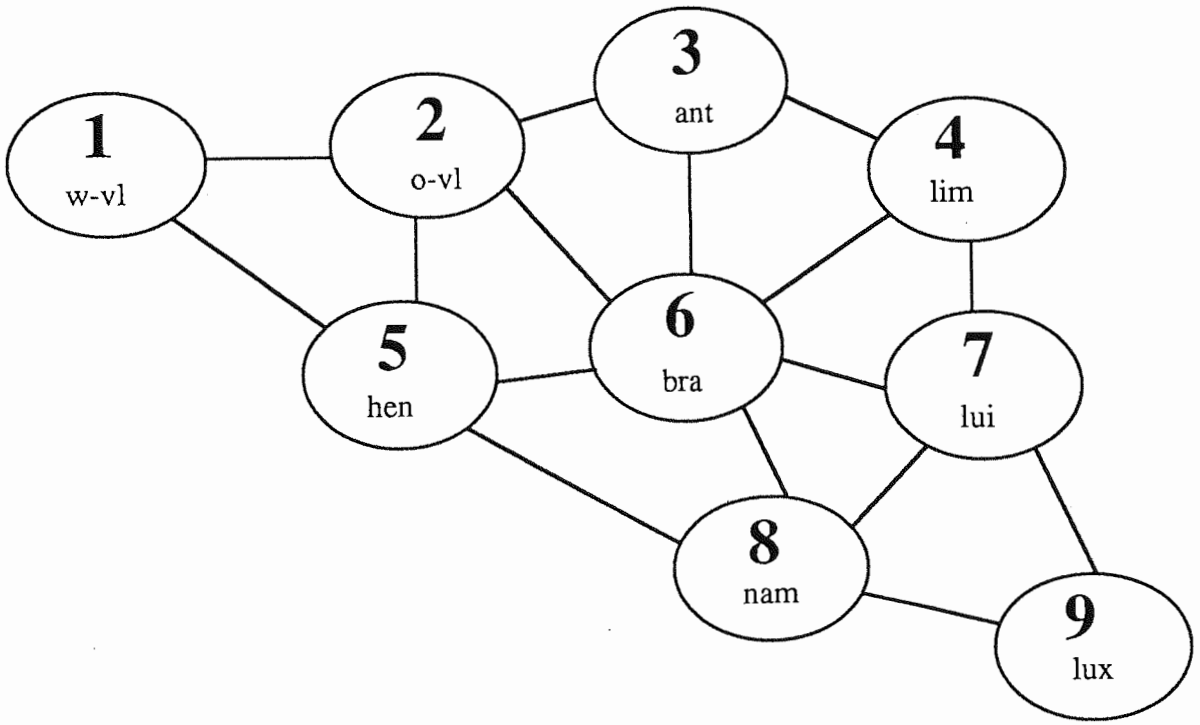


Figure 1 : map of Belgium

Many approaches can be taken to solve this problem. The most simple but expensive in terms of computational time and memory is explicit enumeration. With three colours there are 3^9=19,683 possibilities to consider! Another attempt is a branch and bound scheme using backtracking. The B&B approach can be refined by taking into account special problem features in order to obtain more efficient algorithms, but to illustrate another point of view, consider the following.

The solution strategy which will be developped is more generic. It means that, at least theoretically, the steps of this algorithm in their purest form can be used to tackle every integer problem, subject to a set of simultaneous constraints. Sometimes it will be impossible to use this algorithm because the technical operations needed to perform an individual step are nonexisting. This failure is then due to technical considerations and not to the logic of the algorithm itself. An example of such an impossible operation will occur when this algorithm is used to tackle integer linear programming.
The algorithm is used for the 'Belgian' map colouring problem and all the steps are visualized in figure 2.

Step 1

Every node has a set of possible colours containing green, blue and red. This will be noted as N_1={g,b,r} and so for every node N_i where i=1..9. These are called the unary nodes, corresponding to the unary constraints, meaning that initially, the colour must be one out of the three available colours.

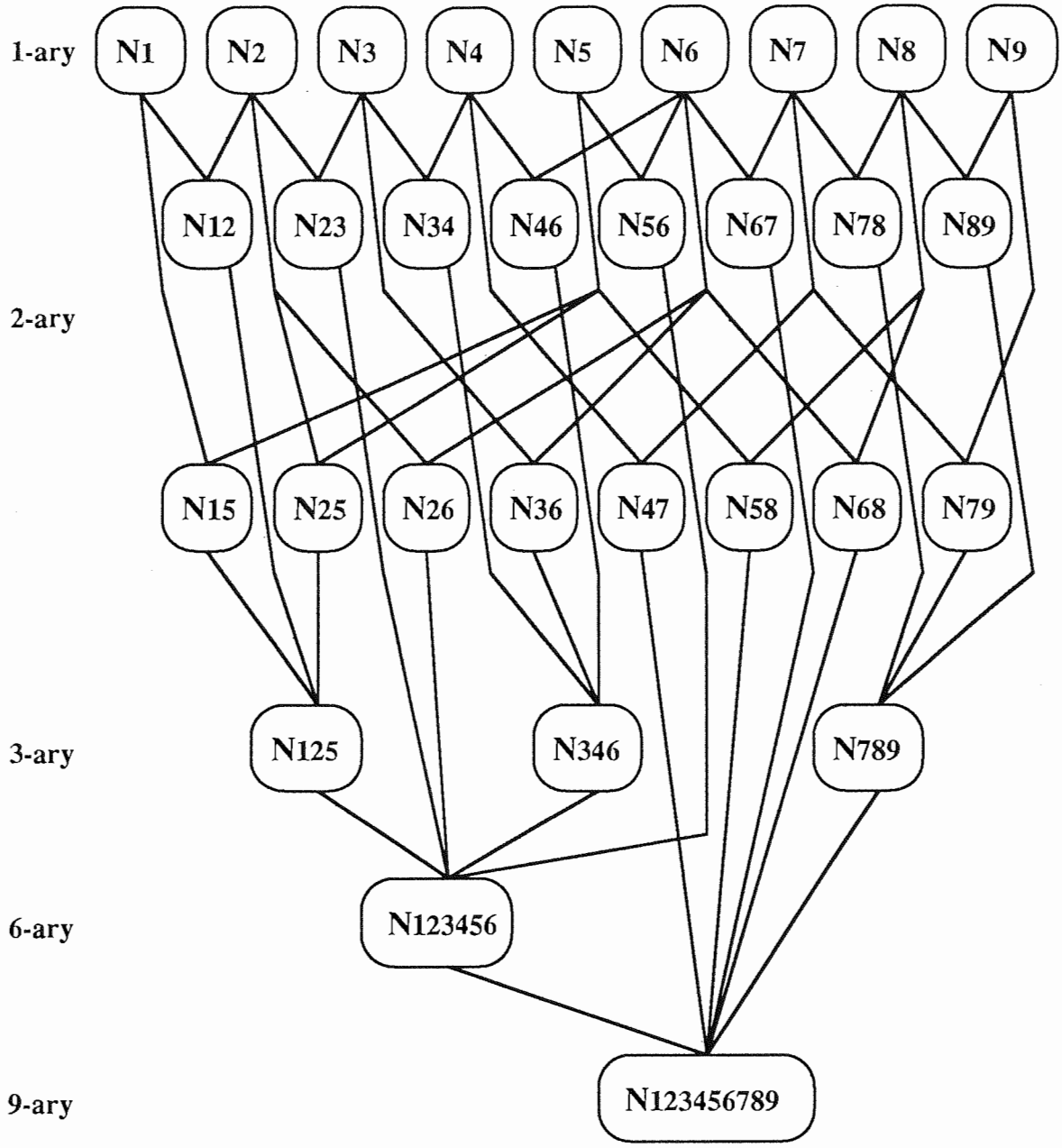


Figure 2 : constraint network for the map of Belgium

Step 2

At this level binary constraints are considered. This concept may not be confused with the one of a binary variable which means a variable which has only two possible values, mostly one and zero. Here binary means consisting of two variables. So it can be extended to n-ary constraints meaning constraints with n variables. In the graph colouring problem all the constraints are binary and of the type :'... has not the same colour as...'. The binary node N_{12} is constructed as follows:

- N_1 and N_2 are combined into N_{12}={gg,gb,gr,bg,bb,br,rg,rb,rr}.
- The constraint between the two nodes is imposed.
- This yields N_{12}={gb,gr,bg,br,rg,rb}.

Because the constraint is the same between every two connected nodes it is easy to obtain the other binary nodes :

N_{12} N_{23} N_{34} N_{46} N_{56} N_{67} N_{78} N_{89}
N_{15} N_{25} N_{36} N_{47} N_{58} N_{68} N_{79}
N_{26}

All these nodes have the same set of possible outcomes, namely the one stated above.

Step 3

Although all the 'real' constraints are dealt with, the final solution is not yet obtained. Ultimately the final solution will be established when all of the constraints are met simultaneously. By now the notion of constraint has been elaborated. A constraint can not only stand for a single constraint but also for a set of constraints which must be met simultaneously. In this way the whole problem can be understood as one constraint composed of the set of all problem constraints. It is obvious that this 'synthesizing constraint' will define all feasible solutions.

Applying the same reasoning procedure, the binary nodes are combined into higher order nodes until the final nine-ary node is obtained :

- N_{125} is a combination of N_{12} and N_5. So, N_{125}={
  gbg,gbb,gbr, grg,grb,grr,
  bgg,bgb,bgr, brg,brb,brr,
  rgg,rgb,rgr, rbg,rbb,rbr}
- Node N_{15} must be imposed, which means that the first and the third colour must be different : gbg,grg,bgb,brb,rgr,rbr are discarded.
- Node N_{25} must also be imposed : gbb,grr,bgg,brr,rgg,rbb are discarded.
- Thus N_{125}={gbr,grb,bgr,brg,rgb,rbg}.

Two other nodes are constructed in a similar fashion : N_{346} and N_{789}. Their set of feasible solutions is the same as the one of N_{125}.

Step 6

Step 4 and step 5 are neglected because in this particular case the steps can be skipped without any risk of missing the solution. If the two intermediate steps are executed, the result will undoubtedly be the same. The reason for this will subsequently be clarified. The six-ary node to be constructed is N_{123456} which is a combination of N_{125} and N_{346}.

N_{123456}={

GBgbRr impose N_{56}, so discard.
GBgrRb impose N_{26}, so discard.
GBbgRr impose N_{56} or N_{23}, so discard.
GBbrRg impose N_{23}, so discard.
GBrgRb impose N_{26}, so discard.
GBrbRg satisfies N_{23}, N_{26} and N_{56}.}

Note that it is only necessary to impose the constraints which connect the 'subnetworks' 125 and 346. The other constraints have already been taken into account implicitly. This can be done for all the members of N_{125} and this leads to N_{123456}={gbrbrg,grbrbg,bgrgrb,brgrgb,rgbgbr,rbgbgr}

Step 9

Again step 7 and 8 are skipped in order to speed things up. The nine-ary node is constructed through a combination of N_{123456} and N_{789} as follows :

GBRBRGgbr impose N_{67}, so discard.
GBRBRGgrb impose N_{67} or N_{58}, so discard.
GBRBRGbgr impose N_{47} or N_{68}, so discard.
GBRBRGbrg impose N_{47} or N_{58}, so discard.
GBRBRGrgb impose N_{68}, so discard.
GBRBRGrbg satisfies N_{47}, N_{67}, N_{58} and N_{68}.

An analogue result can be obtained for the other members of N_{123456}.
Finally $N_{123456789}$ becomes {

GBR BRG RBG
GRB RBG BRG
BGR GRB RGB
BRG RGB GRB
RGB GBR BGR
RBG BGR GBR}

This node of level n=9 is the final node in this example. It's called the node corresponding to the synthesizing constraint. This synthesizing constraint deals with all the constraints of lower levels simultaneously and by definition states the entire set of feasible solutions, which in this case consists of six elements.

Remarks

1. The alert reader has noticed that the six feasible solutions are in fact permutations of each other in the sense that only a certain sequence of colours is important and not the absolute value of the colour. Consequently, in node $N_{123456789}$ it is sufficient to keep track of one sequence. This feature can probably be incorporated into the lower nodes also. This has not been investigated because it lies outside of the scope of this paper.

2. Other refinements of the algorithm for this particular problem are of course possible. However we would like to emphasize that the basic algorithm stays the same whatever the application may be.

3. THEORETICAL PREREQUISITES

The basic definitions and the development of the algorithm are based on the paper of Freuder (1978). The notions which are not relevant for the case at hand are left out. We refer the interested reader to Freuder (1978) for the complete description. Please take care of the fact that some terms are defined completely different to the ones one is used to.

A. Basic terminology

We start with a set of variables $X_1,...,X_n$ the values of which range over their own universes $U_1,...,U_n$, respectively. For the algorithm itself it is assumed that the U_i are discrete and finite. $I=\{1,2,...n\}$ is defined as the set of indices. A lot of definitions use a subset J of I as index ($J \subseteq I$). The indexed set of variables $\{X_j\}_{j\in J}$ is denoted by X_J. A value a_i in U_i is called an instantiation of X_i. An instantiation of a set of variables X_J, denoted by a_J is an indexed set of values $\{a_j\}_{j\in J}$.

A constraint on X_J, denoted by C_J, is a set of instantiations of X_J. Such an instantiation is similar to the representation of an ordered |J|-tuple (|J| stands for cardinality of J). In working with this algorithm the 'indexed' set notation has been found more useful, especially to represent the nodes. So, in this way, given a_J, "$a_j \in a_J$" denotes the instantiation of X_j contained in a_J. Note that C_J can contain more than one 'mathematical' constraint. For instance 'x+y=6' and 'x not equal to y' are both simultaneously represented by C_{xy}. See the n-queens problem for an example.

A constraint expression of order n is a conjunction of constraints $C = \wedge_{J\in 2^I-\{\}} C_J$, one constraint for each subset J of I (except the empty set). It represents the logical conjunction of the relations expressed by the C_J and therefore the fact that the constraints must hold simultaneously. In most of the cases there is not a constraint given for every $J \subseteq I$. Nevertheless, it can be assumed that such a constraint exists : a 'non-constraint' can always be specified which represents the set of all combinations of elements from the universes of the variables in X_J. It actually represents a synthesis of other constraints, which does not impose additional constraining elements.

An instantiation a_J satisfies a constraint C_J if $a_J \in C_J$. The instantiation a_J satisfies a constraint C_H, $H \subseteq J$, if the set $\{a_j \in a_J\}_{j \in H}$, which is called a_J restricted to H, is a member of C_H. An instantiation a_J, where $|J|=k$, k-satisfies a 'constraint expression of order n ($n \geq k$)' if a_J satisfies the constraints C_H for all $H \subseteq J$. If an instantiation a_I n-satisfies the constraint expression of order n, then a_I satisfies the expression.
A constraint expression C is k-satisfiable if for all subsets J of I of cardinality k, there exists an a_J such that a_J k-satisfies C (one is enough). If C of order n is n-satisfiable it is said to be satisfiable (there exists at least one feasible solution). It is clear that a constraint expression defines in fact another constraint : the set of all instantiations a_I which satisfy the constraint expression. This is different from ordinary n-ary constraints which of course can appear also. The main purpose of the algorithm will be to synthesize the order n constraint (which is an n-ary relation) defined on X_I by the constraint expression. In that way it is possible to determine explicitly the set of instantiations a_I which simultaneously satisfy all the given constraints. If such an instantiation is obtained, it is called a solution of the constraint expression. These can be zero (no solution, infeasible set), one or more. In the case of more than one solution, some additional problems can be handled which include optimization.

A constraint network of order k in n variables, $k \leq n$, is a set of constraints called nodes, N_J, for each $J \subseteq I$, $|J| \leq k$, where a link is said to exist between N_J and N_H if $H \subseteq J$ and $|H| = |J|-1$. When two nodes are linked they are called neighbours. A full constraint network is then a constraint network of order n in n variables. A node N_J corresponds to a given constraint C_J if $N_J = C_J$ meaning that each instantiation of the first implies an instantiation of the other and vice versa. In this way a full constraint network in n variables corresponds to a constraint expression of order n if each node N_J in the network corresponds to the constraint C_J in the (synthesizing) expression. The order of a node N_J, or a constraint C_J, is the cardinality of J. So it must be noted that a link between nodes has nothing to do with a constraint; it only refers to a lower (higher) order node which is the representation of a constraint. The advantage of this is a complete correspondance between nodes and constraints so that every definition involving the term constraint can be reformulated in terms of nodes, which is an easier and more visual representation (The step to

programming is also simplified.) For instance, it is possible to speak about an instantiation satisfying a node, a synthesizing node, etc.

B. The issue of constraint propagation

The local propagation of a constraint N_J to a neighbouring constraint N_H consists of removing from N_H all a_H which do not satisfy N_J. A global propagation of a constraint N_J through a neighbouring constraint N_H consists of firstly a local propagation from N_J to N_H; then, if anything was removed from N_H during this local propagation, globally propagate N_H through all its neighbours except N_J. The propagation of a constraint N_J is the global propagation of N_J through all its neighbours.

A constraint network is relaxed if it is possible to propagate every constraint N_J in the network without causing any change in the network. The relaxation of a constraint network is the network obtained by propagating all nodes of the network.

C. Statement of the synthesis algorithm

Given $C=\wedge_{J\in 2^I-\{\}} C_J$.

Step 1 : k=1

Construct a constraint network with nodes N_J corresponding to the constraints C_J in the given constraint expression, for all $J \subseteq I$ of cardinality one.

Step k+1, $2\leq k\leq n-1$

For all $J \subseteq I$ of cardinality k+1:
Add the node N_J to the network corresponding to the given constraint C_J. Link N_J to all N_H such that H is a cardinality k subset of J. Locally propagate to N_J from each of its neighbours. Propagate N_J.

In other words, this algorithm runs in n steps and the result is a full constraint network where N_I corresponds to C. A graphical representation can be found in figure 3.

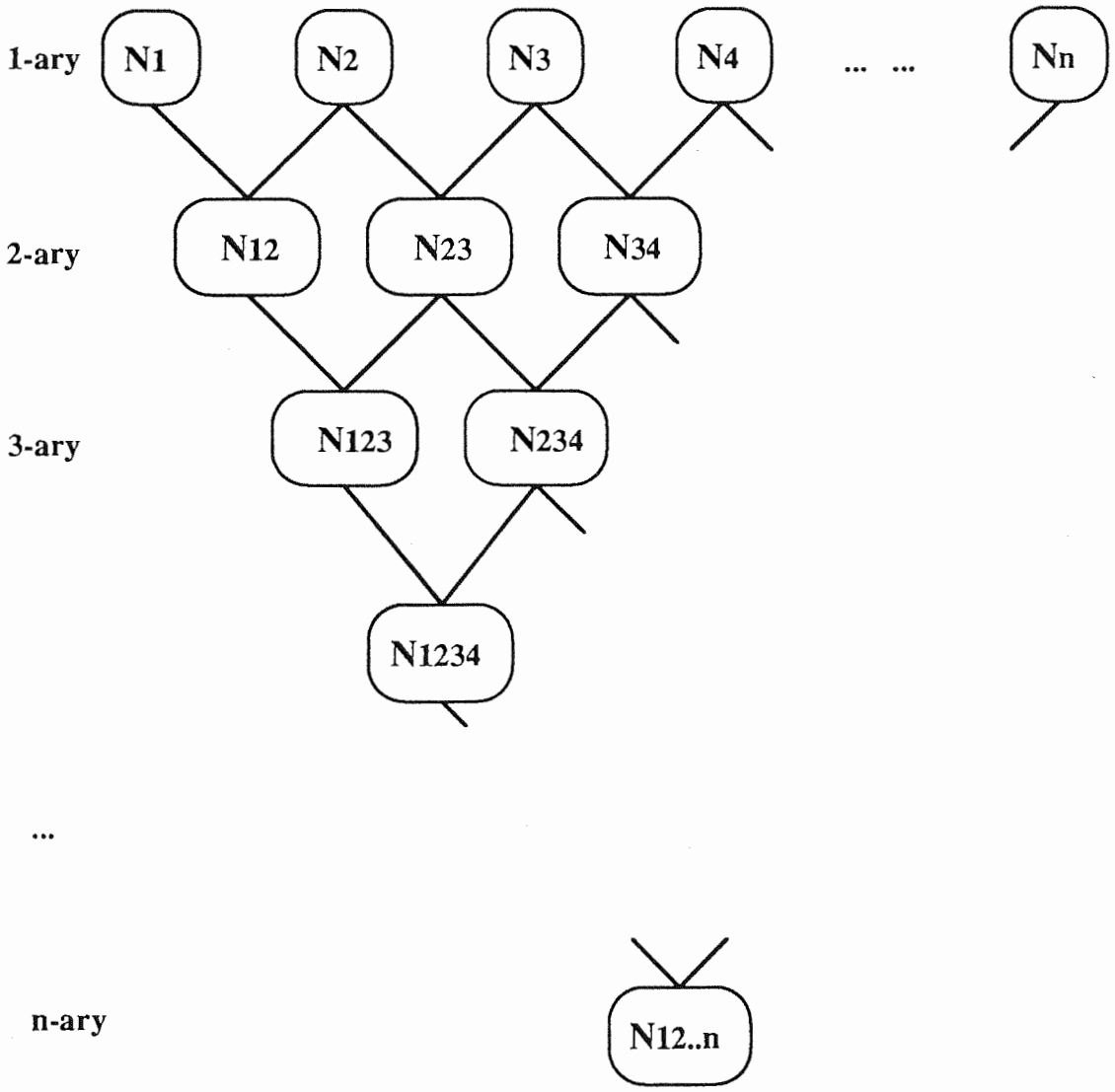


Figure 3 : representation of the general synthesizing constraint network

Already some general observations concerning the algorithm can be stated, which will be used later to tackle integer linear programming.

Observation 1

The network obtained by executing this algorithm is the relaxation of the network corresponding to the synthesizing constraint C. An alert reader may argue that this also could be obtained by simply starting from the order n network and propagating each node. By proceeding by the steps outlined above it is possible to eliminate some instantiations at earlier stages so as to reduce the effects of combinatorial explosion. Note that elimination at a lower level is generally simpler than at a higher level and limits thus also the number of higher order instantiations. So every node is propagated as soon as it is added. Another opportunity to speed up is to add the most constraining node first at a given level, for instance the node with the smallest C_J. The purpose being that by propagating this node, instantiations of other nodes are eliminated so that the construction of future nodes is simplified. This is certainly the case for non-constraints. To construct an N_J from N_H and N_{J-H}, it is preferable to take the node for which $|N_J|x|N_{J-H}|$ is the smallest.

Observation 2

Provisions should be made for an early termination of the algorithm which occurs as soon as a node becomes empty. An empty node means that the constraint corresponding to the node has an empty solution set, and thus the total problem. Also the propagation can be simplified by recognizing non-constraints or by using complementary nodes. Sometimes additional links can enable the direct propagation of a node N_J to some of the nodes corresponding to subsets of J.

Observation 3

It is redundant to include all non-constraints. It is only necessary to have one path up to the n-ary node for every real constraint. This is a correct approach because only the real constraints have an effect on the global solution. On the other hand the non-constraints can be beneficial for the pruning process, which is the act of discarding infeasible instantiations.

Every node N_J groups all the constraints contributing to it, even the ones not originally given by C_J. At the end every member of every N_J is part of some solution of the constraint expression.

D. Three final notions

The reason why the synthesis algorithm can be used for preprocessing or as a full-fledged integer solution method stems from three powerful features : consistency, completeness and compatibility.

Consistency

A constraint network of order k or higher, in n variables, is k-consistent if for any set X_H of k-1 variables, any instantiation a_H of X_H which (k-1)-satisfies N_H, and any choice of a kth variable, X_i; there exists an instantiation of X_i which combines with a_H to k-satisfy N_J, where J is the union of H and {i}. N_J is here considered as a synthesizing constraint corresponding to the partial problem symbolized by N_J in k variables. In other words no partial (consistent) solution of level k-1 together with an instantiation of an arbitrary kth variable may cause an inconsistency at the respective node of level k. The reason for this is the propagation of all the nodes N_J at creation time. Some familiar consistencies are special cases of k-consistency (Freuder,1978): 1-consistency is the same as node consistency. When all nodes are checked for their respective unary constraints then, by definition, the network is 1-consistent. 2-consistency is attained by introducing and propagating all binary constraints. In a network where a link is representing a binary constraint, 2-consistency is equivalent to arc-consistency. In the same kind of network, path-consistency corresponds to 3-consistency. Consistency can intuitively be understood by 'not violating' another constraint which is contained in N_J but not in N_H.

Furthermore, k steps of the synthesis algorithm produce a network that is j-consistent for all $j \leq k$. After k steps of the algorithm, therefore, backtracking can be executed on the remaining values knowing that a single backtracking step will never have to go more backwards than level k.

Compatibility

A node N_J of order k is k-compatible with a constraint expression C if all members of N_J k-satisfy C. A constraint network of order k or higher is k-compatible with C if all nodes of order k are k-compatible with C. If a full constraint network of order n is n-compatible with a constraint expression C of order n it is said to be compatible with C. Intuitively this can be seen as the fact that there is no member of N_J which does not k-satisfy C, although there may be others outside N_J which also k-satisfy C.

Completeness

A node N_J of order k is k-complete for C if any instantiation a_J which k-satisfies C is a member of N_J. A network is k-complete for C if every node of order k is k-complete. An n-complete full constraint network of order n is said to be complete. Again intuitively, node N_J must contain all instantiations a_J which k-satisfies C. Outside this set there are none left.

While compatibility can be thought of as a 'sufficiency' feature for C, completeness is then some kind of 'necessary' feature for C. All of the instantiations of N_J k-satisfy C (compatibility) and they are all contained in N_J (completeness).

It can be proven that k steps of the synthesis algorithm achieve k-compatibility and k-completeness to C. When using this algorithm as a preprocessor, it is possible to choose an order k node and use its members as alternative paths through the first k levels of a search tree (e.g. B&B), only branching on the remaining n-k levels. This is applied in the ILP example. Due to the consistency feature a backtrack will never occur in the first k levels. When the synthesis algorithm has been brought to completion, the order n node constitutes the set of feasible solutions (synthesizing constraint) and no further search is required. Proofs can be found in Freuder (1978).

What is really important, is the following:

- The algorithm can be used to determine the set of all solutions to an integer problem subject to a set of simultaneous constraints.
- Another, possibly more practical use, is to run the algorithm for k steps as a preprocessor and then use the more classical solution methods.

4. INTEGER LINEAR PROGRAMMING

The integer linear programming problems considered here are not of the mixed integer type. Also the objective function value must be integer. Note that an objective function with rational coefficients, can always be transformed to one with integer coefficients just by rescaling the variables. The integer linear program (ILP) can be solved theoretically by the synthesis algorithm, but practically technical problems prohibit this. First an example of the original synthesis algorithm is used. Knowing the cause of the failure to solve the problem, an alternative is developped. Then this alternative will be combined with already existing solution procedures for ILP which seems a worthwhile approach.

A. Synthesising algorithm for ILP

Let's take a very simple example to illustrate the algorithm. Here an optimisation problem is chosen to show how the feasibility method is integrated in an optimization method. A graphical representation can be found in figure 4.

Optimize z
Subject to $x+y-z=0$
$x-y\geq 0$
$x\in[1,10]$
$y\in[3,8]$
$z\in[2,7]$

The constraint expression C equals $(x+y-z=0) \wedge (x-y\geq 0) \wedge (x\in[1,10]) \wedge (y\in[3,8]) \wedge (z\in[2,7])$.

Step 1

The level one nodes are straightforward :

$N_1=[1,10]$, $N_2=[3,8]$, $N_3=[2,7]$

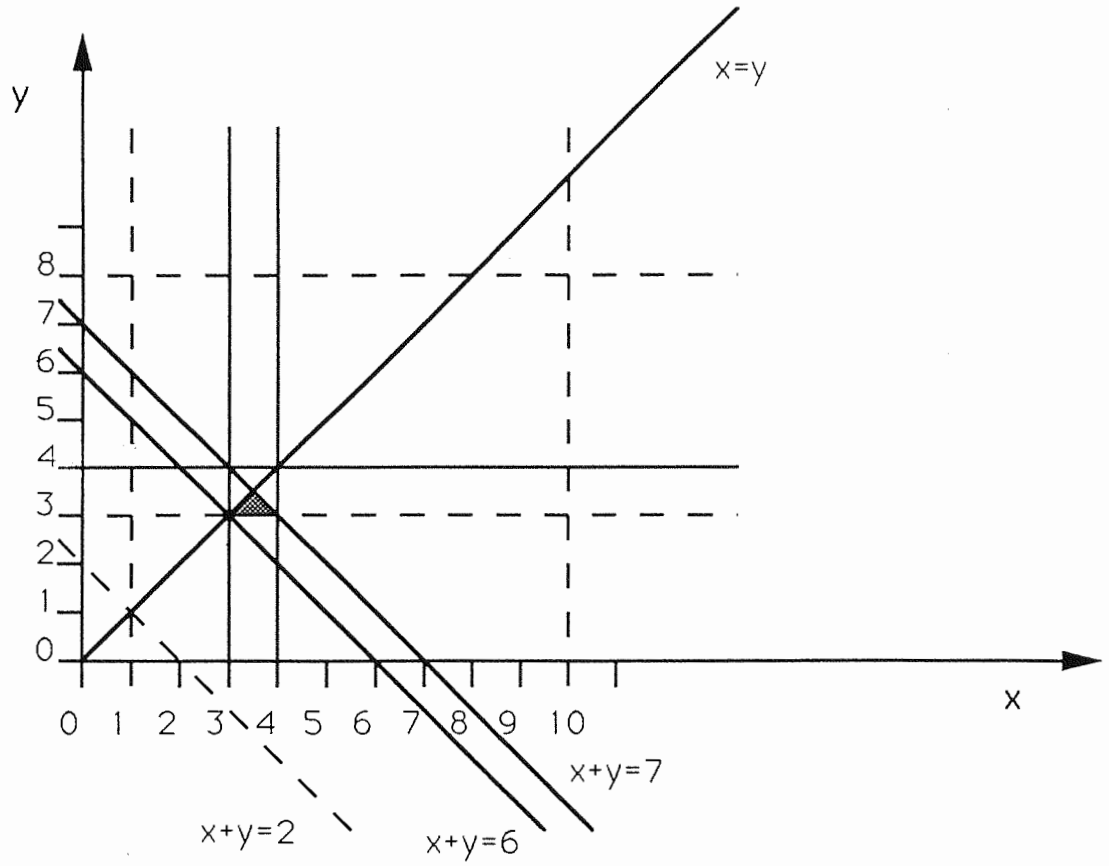


Figure 4 : graphical representation of the example problem

Step 2

To construct node N_{12} we start from the cross product of both linked nodes. Next the constraint is imposed on the set (which is : propagate N_1 and N_2 to N_{12}). If the notation is a 1 for an instantiation which belongs to the node, and a zero for an instantiation not belonging to the node then the node N_{12} can be represented by the following matrix:

x\y	3	4	5	6	7	8
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	1	0	0	0	0	0
4	1	1	0	0	0	0
5	1	1	1	0	0	0
6	1	1	1	1	0	0
7	1	1	1	1	1	0
8	1	1	1	1	1	1
9	1	1	1	1	1	1
10	1	1	1	1	1	1

The instantiations 1 and 2 for N_1 are impossible under all circumstances. Propagation of N_{12} leads to the pruning of the instantiations 1 and 2 for x in N_1. This is the only 'real' binary constraint. Now it is possible to construct the node N_{23}, corresponding to the non-constraint, as the cross product of N_2 and N_3, with all instantiations being possible ([3,8]X[2,7]). But the theoretical discussion stated that one path from each real constraint to the synthesizing constraint is enough. This option is chosen here.

Step 3

The node corresponding to the ternary constraint is added. The result if N_{12} is propagated and the constraint C_{123} imposed, is listed below:

3	3	6	8	3	11	10	3	13
4	3	7		4	12		4	14
	4	8		5	13		5	15
5	3	8		6	14		6	16
	4	9		7	15		7	17
	5	10		8	16		8	18
6	3	9	9	3	12			
	4	10		4	13			
	5	11		5	14			
	6	12		6	15			
7	3	10		7	16			
	4	11		8	17			
	5	12						
	6	13						
	7	14						

If the node N_3 is propagated then, of all the 3-tuples, only (336) and (437) are retained, because z must be $\in$ [2,7]. So N_{123}={336,437}. Now N_{123} is propagated. Therefore N_{123} is globally propagated through all its neighbours.

First N_{123} is globally propagated through N_{12}. Thus N_{123} is first locally propagated to N_{12}, which remains as {33,43}. Because something has been removed from N_{12}, this node must globally propagated. So N_1 becomes {3,4} and N_2={3}.

Second, N_{123} is globally propagated to N_3. This node becomes {6,7}.

The constraint network is now relaxed, and the feasibility problem is completely solved. The constraint network is complete, which means that N_{123} contains all the possible feasible members, none is left out. The constraint network is also compatible, which means that every member in N_{123} satisfies the constraint expression. By definition of the algorithm, the constraint network is 3-consistent. To be complete, all members of N_{123} are solutions of the constraint expression. The result is shown in figure 5.

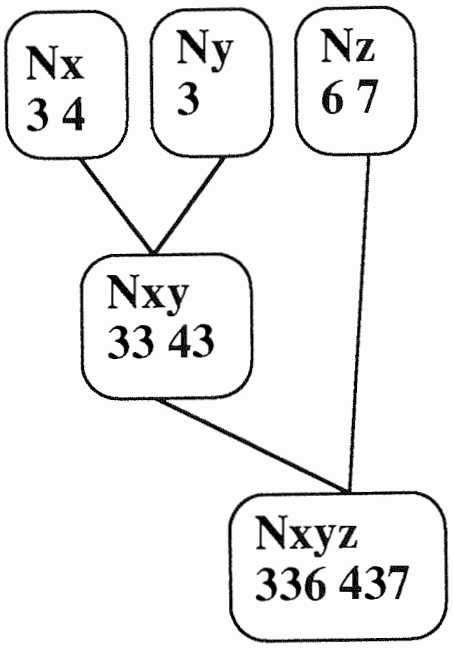


Figure 5 : Synthesizing solution for example problem

If we consider the optimization problem, the following is proposed. The (integer) objective function is replaced by a new integer variable while the constraint is added to the others. So the example problem could stem from :

Optimize	x+y
subject to	x-y≥0

To continue our example, if it were a maximum (minimum) problem then the maximum (minimum) of N_3 is taken and propagated. The reason for propagation is that there is always a possibility of alternative solutions.

So the solution is :

MAX	x = 4	MIN	x = 3
	y = 3		y = 3
	z = 7		z = 6

Of course in this simple example the solution can be derived directly from N_{123}. The reason why it is done here in detail, is primarly to show how the synthesizing algorithm works. Secondly, in more complex problems it will be impossible te 'read' the solution from the synthesizing node, especially when the node is represented in an implicit manner e.g. a calculation in the form of a formula.

Evaluation

It is obvious that the basic control structure of the method is extremely simple and in most cases the structure of the network will be known in advance. It will always end up with the solution(s) so there are no problems of convergence.
The algorithm is fast and the steps are easy to calculate but the memory considerations are impostant due to the higher order nodes. Additionally severe technical problems occur when solving sets of more than one real constraint, like in node N_{123}. Here it is dealt with by means of an enumeration technique. This is only possible for small problems. For large problems other methods are necessary to 'solve' a set of (dis)equations which consist of integer variables.

B. The refine procedure

In the previous part the biggest problem stemmed from the fact that a set of (dis)equations cannot be solved in an efficient manner when dealing with integer valued variables. So this case has to be avoided. Therefore the technique of tightening bounds is used, which is technically better known under the name 'REFINE' procedure. This refine procedure has up to now only been used for continuous variables. Based on interval arithmetic this procedure can now be used to our advantage for the case with integer variables.

The comparison of this method with the synthesizing algorithm can be visualized in the following manner. In figure 6 the unary nodes are all present and so is every real constraint. The real constraints are only linked to their respective unary constraints. Note that the node $N_{12..n}$ is not a synthesizing node but a node only reflecting one real constraint. If more real constraints of a given level exist then this leads to different nodes. Consequently, the strong features of the synthesizing algorithm disappear.

The refine procedure is defined as follows :

Let C_k be a constraint on X ($=[X_1,...,X_n]$) corresponding to the k-th constraint and let S_j be the current interval for X_j.

REFINE$(C_k,X_j) = \{a_j \in S_j \mid \exists\ (a_i \in S_i,\ i=1,...,n,\ i \neq j) \wedge (a_1,...,a_j,...,a_n) \in C_k\}$

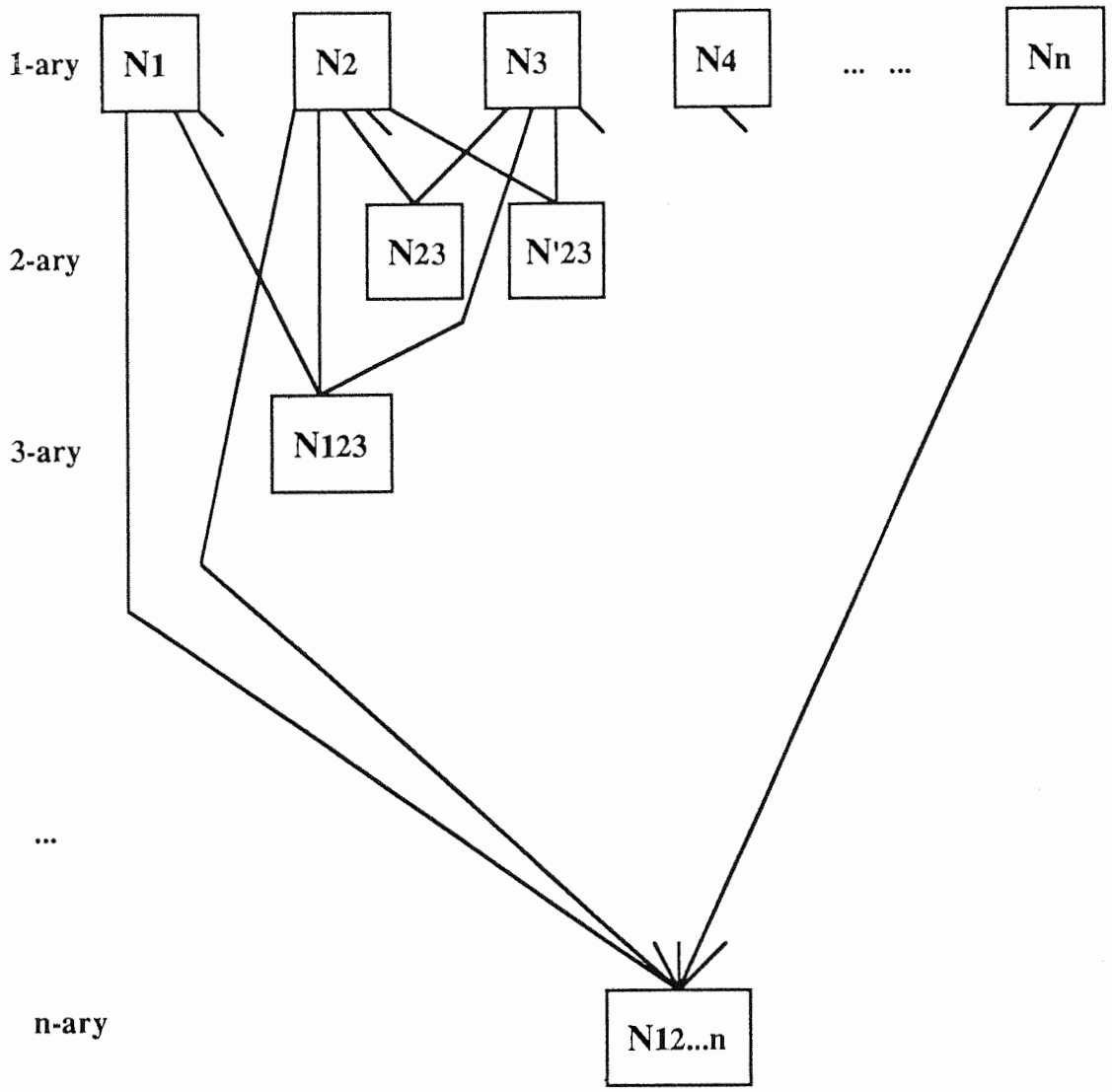


Figure 6 : representation of the general refine network

It looks as if this refinement of the interval S_j is very difficult and time consuming. Fortunately, it is not. Especially for linear constraints this refine procedure can be very easily performed using interval arithmetic. In appendix A we give a survey of the important concepts of interval analysis to which reference is made in this paper. It must be noted that interval arithmetic can also handle some nonlinear relations. Its very power lies in the fact that for calculations with intervals only the upper and lower bounds in combination with some additional conditions have to be taken into account. Another advantage is that once the tightening relations are determined, they remain unchanged during the whole of the solution process. The number of tightening relations of one constraint is equal to the number of variables appearing in the constraint. These numbers summed over all constraints give the total number of tightening relations (sum of the levels of all real constraints, excluding the unary

constraints).
For the example cited above this approach yields :

$x+y-z=0$

1x	$[\max(l_x,l_z-u_y),\min(u_x,u_z-l_y)]$
1y	$[\max(l_y,l_z-u_x),\min(u_y,u_z-l_x)]$
1z	$[\max(l_z,l_x+l_y),\min(u_z,u_x+u_y)]$

$x-y\geq 0$

2x	$[\max(l_x,l_y),u_x]$
2y	$[l_y,\min(u_y,u_x)]$

These five tightening relations never change during execution. When these relations are repetitively executed, the intervals of all variables are reduced. In the case of continuous variables as well as in the case of integer variables, both with unity coefficients, this process always terminates (cf. relaxation in the synthesizing algorithm). The refine process quiesces. In the case of general coefficients and continuous variables this process can go into an 'endless' loop, even for small problems. Take for example the following problem :

$x \in [0,100]$ and
$y \in [0,100]$ and
$x=y$
$2x=y$
The sequence of tightening is : $x \in [0,50]$, $y \in [0,50]$, $x \in [0,25]$, $y \in [0,25]$, ... If one uses a very accurate computer, then it it takes a long time for the computer to figure out the solution $x=y=0$. So in this case stopping criteria must be designed. We consider the latter problem to be far less relevant when dealing with integer variables and general coefficients. Later it will become clear that as soon as a variable is not integer, its fractional part can be truncated. This speeds up the overall process and offers a good protection against loops.

But let's continue to solve the example of which the network is presented in figure 7.

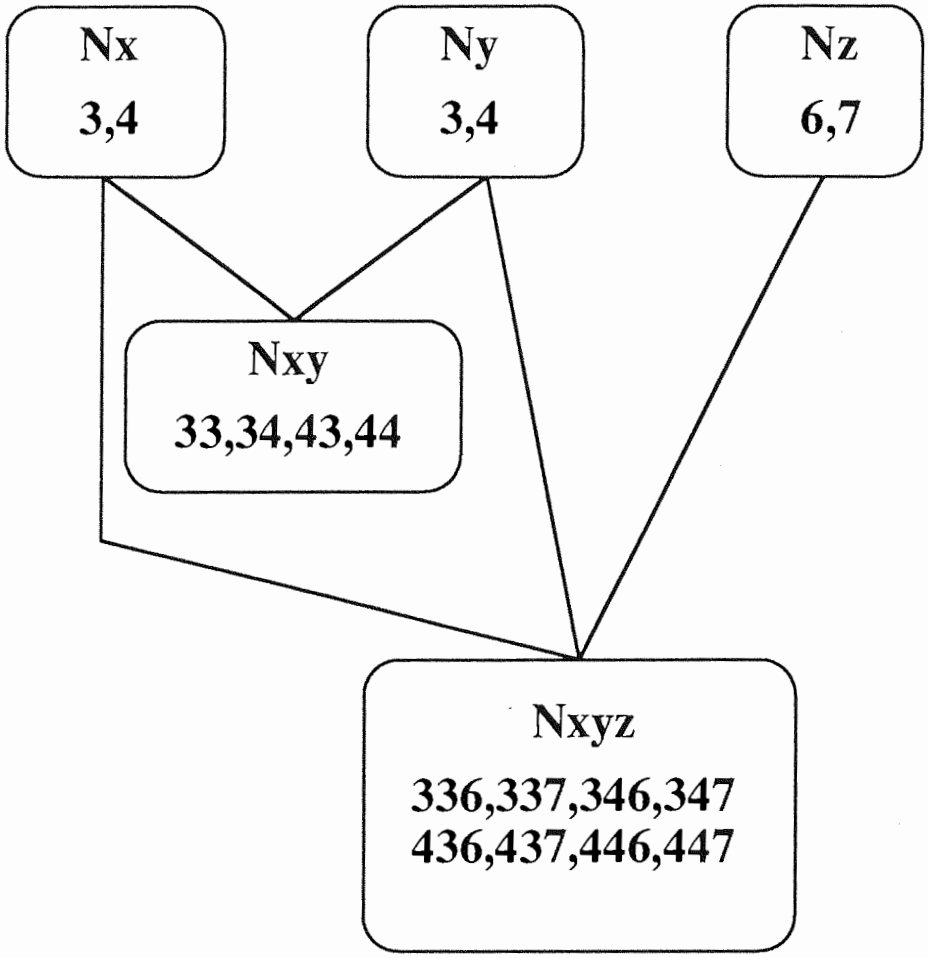


Figure 7 : refine network of the example

Pass 1

1x [max(1,2-8),min(10,7-3)] = [1,4]
1y [max(3,2-4),min(8,7-1)] = [3,6]
1z [max(2,1+3),min(7,4+6)] = [4,7]

2x [max(1,3),4] = [3,4]
2y [3,min(6,4)] = [3,4]

Pass 2

1x [max(3,4-4),min(4,7-3)] = [3,4]
1y [max(3,4-4),min(4,7-3)] = [3,4]
1z [max(4,3+3),min(7,4+4)] = [6,7]

After this there will be no further tightening. The system quiesces. But what does this solution mean? It's the projection of the hull of the feasible region derived from the synthesizing constraint on the axes (see figure 8). If we make the cross product of these solution intervals, we obtain the following result :

$$CH=\{336,337,346,347,436,437,446,447\} \quad \text{cardinality} = 2^3$$

Also the projection of the hull of the feasible region of the individual constraints on their respective axes can be determined :

$$NH_{12}=\{33,34,43,44\}$$
$$NH_{123}=\{336,337,346,347,436,437,446,447\}.$$

In other words, the true feasible regions are smaller then the regions obtained but are completely covered (note : the exact feasible regions are $x \in \{3,4\}$, $y \in \{3\}$, $z \in \{6,7\}$ and the feasible sets are here $N_{12}=\{33,43\}$, $N_{123}=\{336,437\}$ and $C=\{336,437\}$. All of them are covered). It must be stressed that by the refinement procedure some infeasible instantiations are not removed.

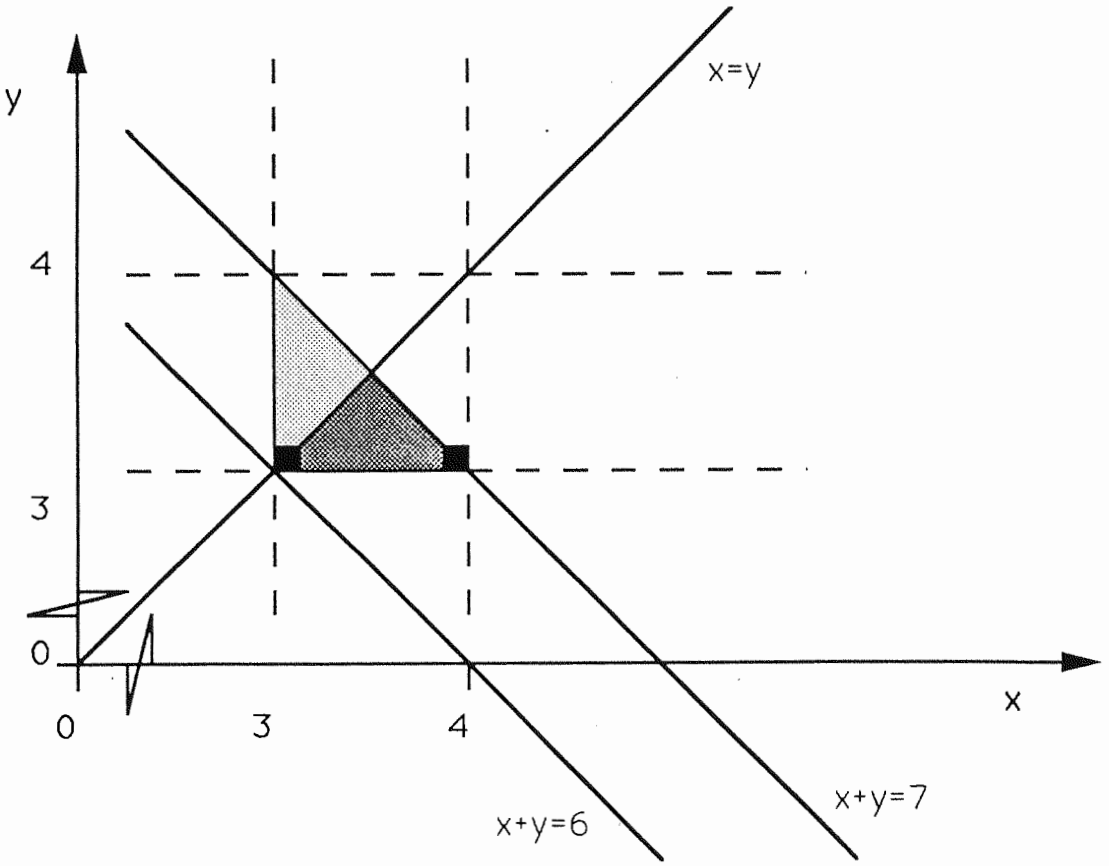


Figure 8 : regions after the application of refine

C. Turn to a possible formulation for an algorithm

The 1-compatibility of every unary node with the constraint expression still applies, so the constraint network of order 1 is 1-compatible with the constraint expression. The instantiations of the variables satisify their own starting interval. Other compatibilties cannot be determined because the nodes haven't been constructed yet.

The 1-completeness of the unary nodes for C is not guaranteed, e.g. the instantiation 1 of node N_1 1-satisfies C but is not a member anymore of N_1 after refinement. This is of course because completeness is defined in synthesizing terms. However, it is safe to state that every instantiation a_i of a unary node N_i is a candidate to satisfy C (in the sense of satisfying C through a feasible n-tuple), although it will turn out that some of the $a_i \in N_i$ will never satisfy C, which means that there will never be any combination of this instantiation a_i with other ones to form an instantiation a_I which will satisfy C. This is the case with the instantiation 4 of N_2 in the above example.

In consistency terms, 1-consistency or node consistency is present. 2-consistency is, again, not guaranteed. Consequently, it is possible to have an instantiation which is 1-consistent, but which certainly not n-satisfies C. Note that in the above example the members of the instantiation 337 of node N_{123} are 1-consistent, but 337 does not 3-satisfy C, although e.g. 33 satisfies the constraint C_{12}. 337 does not even satisfy the constraint C_{123} itself. In short, 1-compatibility is guaranteed by the refine procedure, so it can be neglected from now on. Additionally, a new definition is necessary to go on.

An instantiation a_i of a unary node N_i is k-retainable for a real constraint node N_J ($J \subseteq I$) if it is possible to construct one feasible $|J|$-tuple with a_i which satisfies N_J. If an instantiation a_i of a unary node N_i is n-retainable for the synthesizing constraint C, then a_i is retainable for C. The instantiation 1 of x is not 2-retainable for N_{12}. Also the instantiation 4 of N_2 is not n-retainable for C. As can be noticed k-retainability for one real constraint is removed by the refine procedure, by definition of refine. So, only retainability for C can remain after the application of refine. If such an instantiation a_i (not retainable for C) is detected, this instantiation can

be removed. Of course only lower or upper bounds can be removed by a propagation using refine. In the above example this leads to :

NH_{12}={33,43}
NH_{123}={336,337,436,437}=CH

It is possible that an instantiation a_J of a real constraint node N_J does not satisfy N_J although all members a_j of a_J are k-retainable for N_J. Consequently, it is also possible that instantiation a_I of a synthesizing constraint C does not satisfy C although all members a_j of a_I are retainable for C. In other words, if an instantiation a_J of a real constraint N_J satisfies N_J, then all $a_i$of a_J are k-retainable for N_J. The reverse however is not true. In contrast with retainability, this can occur at the individual real constraint level after the application of REFINE. Take the instantiation 337 of N_{123} as an example. All members of 337 are 3-retainable and still 337 does not satisfy constraint C_{123}. The same is true for 436. 34 on the other hand does not satisfy constraint C_{12} but its member 4 was not 2-retainable for N_{12}. After removal of 4 out of N_2, 3 of N_2 is 2-retainable for N_{12} and both 33 and 43 satisfy N_{12}. REFINE does not guarantee that an instantiation satisfies both a real constraint and the synthesizing constraint.

In addition to these difficulties, our optimization problem has not yet been solved. If one states that the maximum of z is 7, because it is the upper bound of the interval, one only had a lucky strike. This way of thinking cannot be generalized. So, based on the obtained results, I suggest three possible ways. Let's call this point 'the branching point', because we need to decide which one of three options to choose for branching : firstly instantiation on a general variable, secondly instantiation on the objective function variable and thirdly a LP-run.

a. Instantiation

If one interval is small, then this can be an opportunity to take each of the values as the starting values for a branching scheme. This branching will not overlook any valid candidate for a feasible n-tuple. This is true because only non-retainable instantiations are removed by REFINE. Because this is the first level of the branching process and the retainability for C is not guaranteed by the REFINE procedure, it is

possible that an entire branch can be pruned at some moment in time. In the example, branching on N_2 will lead to a branch of y=4 which cannot contain any n-tuple satisfying C. Like we already mentioned, when this happens and it involves a bound of the interval, refine can be activated to perform an update. Also if a variable is instantiated, the result of this can first be checked with refine. In case of a maximum problem for instance, it seems logic to start with the upper bound and some subsequent lower values of the variable provided this variable has a positive objective function coefficient. When appropriate, neighbouring values can be gathered in an interval. When the system quiesces the decision point is reached again. In our example it will look like :

Branch to y=4

Pass 1

1x	[3,3]=3
1y	[4,4]=4
1z	[7,7]=7
2x	[4,3] => infeasible

Just using refine leads to the conclusion that instantiating y on 4 is a branch which is to be pruned completely. Because y=4 is an upper bound, $y \in [3,3]=3$ is propagated using refine :

Pass 1

1x	[3,4]
1y	[3,3]=3
1z	[6,7]
2x	[3,4]
2y	[3,3]=3

Nothing has changed to the original intervals, after the execution of refine. Note that here also intervals or a combination of intervals and individual values could be instantiated. The decision now is equivalent to

the earlier one, and therefore again the same three options are available to us.

b. Instantiation of the objective function

This is technically the same as the previous option, but this variable is mostly not bounded by a small interval. Nevertheless, sometimes it can be advantageous to instantiate for instance some values of this variable and see what happens. The assumption is of course that the objective function is integer, which is here a starting assumption. When dealing with a maximum problem, values smaller than or equal to the upper bound are taken. (For minimization problems, it is just the other way around. For ease of statement of the algorithm, it is assumed that a minimum problem is transformed into a maximum problem). Of course this option is only useful if the interval of the objective is already 'good' (for instance after one full LP run). For the above example we have in case of maximization, the following:

Pass 1

1x	[3,4]
1y	[3,4]
1z	[7,7]=7

2x and 2y are not influenced by z

Nothing has changed. But suppose it was already established (see 1.) that y=3. Then it becomes

1x	[4,4]=4
1y	[3,3]=3
1z	[7,7]=7

It is guaranteed that this is the only integer solution of the maximization problem. Thus, given y=3 and z=7 then x equals 4. If x were still defined as an interval with a length larger than zero (≥ 1), then alternative solutions would exist. If the minimum was chosen, then the solution 336 would be obtained. In general however, after propagation the decision

point is reached again.

c. LP-run

In this option an LP run with bounds is suggested. It is known that imposing bounds on the LP variables does not harm very much the speed of the LP run. This is mostly the use of bound tightening in literature, but to my opinion more effective use can be made of the tightening technique. Suppose we have a maximum problem. If the LP solution is feasible, then a candidate for a general lower bound (GLB) on the objective is found. On the other hand, if the LP solution is not feasible, then at least a local upper bound (LUB) for the objective is known. If this LUB has a fractional part then the LUB is replaced by $\lfloor LUB \rfloor$, the largest possible integer smaller than LUB. If GUB=GLB then the problem is solved, but reservations for alternative solutions must be incorporated. Additionally, if a general lower bound is established, every branch with a local upper bound lower than GLB can be pruned. The construction of a GUB in fact more technical and will become clear when the algorithm is formally stated. Afterwards the decision point has been reached again. Note that the subsequent LP-runs only differ in their bounds, so they can be executed very fastly.

In the example the LP run is executed with result :

1a. max z

LP model : max z
subject to : $x+y-z=0$
$x-y \geq 0$
$x \in [3,4], y \in [3,4], z \in [6,7]$

The LP solutions are :
$z = 7$
$x = 4, 3.9, ..., 3.6, 3.5$
$y = 3, 3.1, ..., 3.4, 3.5$

These are indeed all alternative solutions. z=7 is now a LUB (here also a GUB). Because of the feasibility of x=4 together with y=3, the general

lower bound is 7. In this case only one solution remains :

$$x = 4 \quad y = 3 \quad z = 7$$

b. min z

The model is the same as above except for the objective function of course. The LP run gives : x = 3, y = 3 and z = 6. The local (here also the general) lower bound is 6 and because of integrality, also the general upper bound. The solution procedure stops.

The example used here is very simple. It has only been used to highlight some features of the refine procedure embedded in a broader branch and bound scheme. An attentive reader could remark that using unity coefficients in an ILP model is very beneficial for a method using LP, because the probability of occurence of fractional results is here somehow lower than in a problem with general coefficients. Therefore some problems with general coefficients have been exploited. It has already been pointed out that the refine process in case of integrality conditions is speed up by removing fractional parts from the bounds as soon as they occur. For instance, if an interval of an integer ends up with [3.5,8.2] then this interval can be rewritten as [4,8]. This result must be propagated again, in order to try to cut off other values of other variables. Before turning to the illustrations and a comparison with other well-known methods for ILP, some notions about the skeleton algorithm we have in mind. The algorithm is stated for a maximization problem, for a minimization problem transformation to a maximization problem is appropriate.

d. Skeleton algorithm

The basic control structure in the algorithm is breadth-first. This means that, once a branching variable is chosen, it must be completely exploited. Nevertheless, some branches can be gathered dynamically with the intention to speed up the search. So initially, this breadth-first control structure does not harm the efficiency of the algorithm.

As a matter of initialisation, an empty node is constructed. A strictly ordered queue Q is now formed : Q={1}. For reasons of clarity, every

node has a local lower and upper bound (LLB and LUB respectively) and there is all the time only one general lower and upper bound (GLB and GUB). U_0 and L_0 are the upper and lower value of the objective function. O is used as the objective function value of a feasible solution and O_{LP} for the objective function value associated with an LP-run.

1. PROPAGATION PHASE

- propagate the first node of Q until the system quiesces. Fill in and call it the current node.
- if solution is infeasible
  then remove current node from Q
  else
  {if it is the first propagation
  then set GUB=LUB=U_0 and GLB=LLB=L_0
  else set LUB=U_0 and LLB=L_0
  if LUB<GLB then remove current node from Q.
  else
  {begin
  if solution is feasible
  then O=U_0=L_0
  if O≥GLB
  then - GLB=O
  - move current node to the back of the branching level
  - remove all nodes which LUB<GLB
  else [neither feasible nor infeasible]
  if it was the first propagation
  then take in the next step option three except if there is a valid reason to choose option 1 or 2 (go to the chosen option immediately).
  else put the node at the end of the branching level.
  end}}
- go the decision point.

2. THE DECISION POINT

- if Q is empty then STOP. Infeasible problem solution.
- if GUB=GLB
  then - one optimal solution is found
  - if alternative solutions are of no interest
    then STOP and give solutions
    else if for all nodes GUB=GLB
      then STOP and give solutions
      else take current unexhausted level
        if it is an objective level
        then - remove from this level all nodes concerning lower objective instantiations
        else - gather the rest of the level in an interval
        - put it in front of the Q
        - propagate.
- if the branching level is not exhausted then propagate.
- if the branching level is exhausted
  then if the next node is of the same branching level
    then help=maximum of the LUB of the branching level
  if the next node is of a higher level
    then help=LUB of the next node
  if help<GUB then GUB=help.
- if GUB=GLB then the same as above.
- go to the branching point.

3. THE BRANCHING POINT

- take the first node of Q which is not a solution.

Option 1

- if a variable has a small interval
  then - instantiate on some individual values and put the rest into an interval (exhaustive and in descending (ascending) order for positive (negative) objective coefficients)
  - remove parent node (remember intervals)
  - add the child nodes in front of Q in descending (ascending) order (recall intervals)
  - propagate.

Option 2

- if the objective is 'good' in spite of a large interval
  then - instantiate on some individual values and put the rest into an interval (exhaustive and in descending (ascending) order for positive (negative) objective coefficients)
  - remove parent node (remember intervals)
  - add child nodes in front of Q in descending order (recall intervals)
  - propagate.

Option 3

- remove parent node (remember intervals)
- add one child node with the LP result.
- if the solution is feasible [=integer]
  then if O_{LP}<GLB
  then - prune current node (forget intervals)
  else - GLB=O_{LP}
  - remove all nodes with LUB<GLB
  - remove parent
  - add new node in front (recall intervals)
  - propagate [GLB,LUB].

- if solution is infeasible [not integer or empty]
  if not integer
  then $LUB_{LP}=\lfloor O_{LP} \rfloor$
    if $LUB_{LP}<GLB$
    then - prune current node (forget intervals)
    else - remove parent node
      - add new node to front of Q (recall intervals)
      - propagate $[LLB,LUB_{LP}]$.
  else [empty]
  then prune current node (forget intervals)

- go to the decision point

This is very roughly the logic of the skeleton algorithm and it is nothing more than a skeleton. The branching point must still be formalized and some other characteristics must be represented by parameters. For instance :

The choice of the variable to branche on
Some useful heuristics must be evaluated : both the variable with the smallest interval and the most constrained variable are favourable candidates. Probably it is possible to obtain more complicated evaluations using a combination of various heuristics.

The way the branching has to be performed
The question is whether the number of instantiations should obey a certain heuristic. For instance the division of the interval among golden ratio logic is worth looking at. Another possibility could be : if the size of the interval is smaller than 5, all values will be considered separately; if the size is larger than 5, the upper 5 will be instantiated and the rest of the interval wille be divided into four parts, etc. With interval arithmetic it is possible to use both integers ànd intervals in the same branching level. The way of propagating is technically the same.

The characterization of a 'good' objective function
This feature will not easily be established. At least it is possible to treat the objective function variable just like any other variable. But to my opinion, some characteristics of this 'special' variable can be exploited.

For instance if a feasible solution is found after propagation or after a LP-run, this can give valuable information about which values of the objective function variable can be neglected (based on general lower bound reasonings). Another point can be made by saying that it is useful to take the objective function variable to obtain as fast as possible a feasible solution in order to be able to limit the intervals of all variables and to prune other branches.

The formalization of the underlying B&B process
These things will merely consider the computer implementation and therefore will not be discussed here.

Although all this features are not yet incorporated, it is already possible to use the skeleton interactively and opportunistly. This will become clear in the other examples of ILP. Three principles lie behind the skeleton algorithm :

1 related to cutting planes

If REFINE encounters non-integer numbers then the fractions are truncated. The same way the objective function values can be truncated after a LP run.

2 related to implicit enumeration

The total instantiaton of a small interval is basically traditional branching. Here, instantiation of the objective function or other variables is a hybrid form : some individual values will be instantiated and the rest will be left in an interval. This does not present a problem at all because every real (and by the way integer) number is an interval of length 0. Bounding is used in four ways :
- The LP runs can give general lower or local upper bounds.
- The intervals in itself contain bounds on all variables.
- Refine can give local lower and upper bounds and/or a general lower bound.
- The 'controlled' breadth first logic, based on the exhaustive and ordered branching strategy, gives always general upper bounds.

3 related to opportunistic methods

The decision point evaluation and subsequent choices are more or less opportunistic. When the algorithm is to be implemented these points have to be formalized.

Now the basics of the algorithm should be clear enough to follow the reasoning behind the solution procedure applied to four examples. The steps are written out in appendix. Here only the results are presented. The problems are all four pure ILP problems. The first is taken from Dirickx (1987), the second from Van Winckel (1990), the third from Wagner (1975) and the fourth is the example already used above. The latter is the integer form of a continuous problem proposed by Davis (1987). The choice has been based on two criteria : firstly on the fact that different authors are invoked in order to have more variety and secondly, the availability of other solution methods as to save some work.

Example 1

Min z
S.T. $3x+y-z=0$
$3x+2y\geq 6$
$5x-4y\leq 5$
$2x-y\geq 1$

Example 2

Max z
S.T. $7x+9y-z=0$
$-1/3x+y\leq 2$
$x+1/7y\leq 5$

Example 3

Max v
S.T. $3x+3y+13z-v=0$
$-3x+6y+7z\leq 8$
$6x-3y+7z\leq 8$
$x,y,z \in [0,5]$

Example 4

Max/Min z

S.T. $x+y-z=0$

$x-y \geq 0$

$x \in [1,10]$

$y \in [3,8]$

$z \in [2,7]$

The comparison has been made in function of the number of LP-runs that where needed during the solution process. This measure can be criticized, but at least it gives an indication of the possible power of the method. For 0-1 problems, it has already been demonstrated that the reduction of the number of LP-runs is promising (Guignard and Spielberg, 1977). The other methods are the cutting plane method of Gomory, the Land & Doig algorithm and the algorithm of Dakin. The problems were already solved by the respective authors using one or more methods.

Problem	Gomory	Land & Doig	Dakin	Skeleton
1	4	6	5	1
2	3	6	5	1
3	-	13	9	1 or 2
4	1	1	1	1

According to these four examples, the skeleton algorithm seems to work well. Without special effort, it was possible to solve these examples (by hand) with considerable less LP-runs. In fact what happened is that the LP-runs have been replaced by propagations. Of course the question remains whether these steps are more efficient in terms of execution time and memory considerations than the LP-runs. Also it is not known how this replacement will evolve when larger problems are encountered. To answer these questions, a computer implementation is necessary. The algorithm should be programmed in order to be able to tackle larger problems, so that the possible advantages of the refine procedure can be further investigated. A prerequisite for its implementation is the formalization of the parameters discussed above.

5. ANOTHER APPLICATION OF THE SYNTHESIZING ALGORITHM : THE N-QUEENS PROBLEM DEMYSTIFIED?

It is very impressive to see how some of the recently available software packages solve combinatorical problems in an amazing speed. Examples found in literature include warehouse location (Van Hentenryck and Carillon, 1988; Van Hentenryck, 1989), the car sequencing problem (Dincbas, Simonis and Van Hentenryck, 1988), the n-queens problem (Van Hentenryck, 1989; Van Hentenryck and Dincbas, 1986), the graph colouring problem (Van Hentenryck, 1989; Dincbas, Simonis and Van Hentenryck, 1990), the cutting stock problem (Van Hentenryck, 1989) and disjunctive project planning (Dincbas, Simonis and Van Hentenryck, 1990) among probably others. The packages we have seen demonstrated are CHARME (Bull), PECOS (E2S) and CHIP (Dincbas et alii, 1988). All of these packages are programmed in a language like Prolog, LISP or LE-LISP. The striking aspect was that all of them used the n-queens problem as the ultimate example to show off the speed of their package. In this paragraph, it is not the intention to tell how the packages really solve the n-queens problem, but an attempt is made to show that this problem can be solved with the synthesizing algorithm in a polynomial time. The pecularities of the individual packages can, according to my opinion, be explained by further refinements of 'a synthesizing algorithm based' procedure.

This problem is not an optimization problem, it is a pure constraint satisfaction problem. Therefore it can be solved completely by the synthesizing algorithm. I have chosen to write the solution procedure out in full, because it is a good illustration of how the synthesizing algorithm really works.

A. Statement of the problem

In a chess game, the queen is a very powerfull piece. She can strike horizontally, vertically and diagonally. Our chessboard is a 5x5 board. The problem is putting on this board five queens, which are unable to strike each other. I can assure you that finding all feasible solutions (and having the certainty of having all of them) is not that simple. Here a branch and bound approach proves to be valuable. So is the synthesizing

algorithm. Almost in all available mathematical formulations the problem is transformed to a one dimensional vector of five elements $[x_1,...,x_5]$ where x_i denotes the row of the ith column. The vector must satisfy :

1. $1 \leq x_i \leq 5$ $(1 \leq i \leq 5)$;
2. $x_i \neq x_j$ $(1 \leq i < j \leq 5)$;
3. $x_i \neq x_j + (j-i)$ $(1 \leq i < j \leq 5)$;
4. $x_i \neq x_j - (j-i)$ $(1 \leq i < j \leq 5)$;

Note that the last three constraints are of the binary type and the first of the unary type. So, every binary node will refer to three constraints simultaneously. A quick calculation shows that there are 5 unary constraints and 10 binary ones (5 into groups of 2).

B. The solution procedure

Step 1

This step is straightforward. All individual values of the x_i must take one of the values of $\{1,2,3,4,5\} = N_i$, $i=1,...,5$.

Step 2

Here the nodes of level two are constructed. Let's take N_{12} as an example. The possible instantiations derived from N_1 and N_2 are represented by a 5x5 matrix, where all the entries are initially 1. A '0' should stand for an instantiation which does not satisfy the constraint. Imposing the three constraints on N_{12} leads to a matrix R_{12} :

$$R_{12} = \begin{matrix} 00111 \\ 00011 \\ 10001 \\ 11000 \\ 11100 \end{matrix} = R_{23} = R_{34} = R_{45}$$

And so for the others :

$$R_{13} = \begin{matrix} 01011 \\ 10101 \\ 01010 \\ 10101 \\ 11010 \end{matrix} = R_{24} = R_{35}$$

$$R_{14} = \begin{matrix} 01101 \\ 10110 \\ 11011 \\ 01101 \\ 10110 \end{matrix} = R_{25}$$

$$R_{15} = \begin{matrix} 01110 \\ 10111 \\ 11011 \\ 11101 \\ 01110 \end{matrix}$$

It can be seen that the outlook of the matrices depends on the number of columns between the variables. This is what could be expected from chess reality. Now the algorithm wants to locally propagate the binary nodes. This can be done but nothing will change. Formally this can be computed by taking the sum of the individual rows (or columns). If this sum is equal to zero, the value corresponding to the row(column)number can be removed from the respective unary node. This is not the case right now. Also if N_{12} is propagated, nothing can be removed.

Step 3

Because all real constraints have been incorporated, the new nodes are non-constraints. Let's take node N_{123}. It is constructed using N_{12}, N_{23} and N_{13} :

N_{12}	N_{23}	N_{13}	N_{123}	
13	10001	01011	00001	135
14	11000	01011	01000	142
15	11100	01011	01000	152
24	11000	10101	10000	241
25	11100	10101	10100	251 and 253
31	00111	01010	00010	314
35	11100	01010	01000	352
41	00111	10101	00101	413 and 415
42	00011	10101	00001	425
51	00111	11010	00010	514
52	00011	11010	00010	524
53	10001	11010	10000	531

Some clarification is needed. In the first column the node N_{12} is listed. Take matrix element 13. If 1 is fixed in the first variable, then 3 is a possible value for the second variable. If node N_{23} is added, it is shown in de second column that if 3 is fixed for the second variable, then 1 and 5 are possible for the third variable. If node N_{13} is added, with 1 fixed for the first variable, then 2, 4 and 5 are allowed for the third one. Combining these conditions leads to only one instantiation which satisfies N_{123} : 135. This can easily be performed by logical 'and'-operations on the second and third column. The latter clarifies what is meant by the local propagation of the neighbours to the new node. Note that only two instantiations of N_{12} end up with two alternatives. Still N_{123} has to be propagated :
to N_{12} : no change in R_{12}
to N_{23} : no change in R_{23}
to N_{13} : some change in R_{13} :
entries (1,4), (2,5), (4,1), (5,2) must be removed from R_{13}. This can also be done by some binary operations, which are beyond the scope of this paper, but can be found in Mackworth (1977).

This procedure can be repeated for :

N_{124}	N_{125}	N_{234}	N_{345}	N_{235}
132	132 134	135	same	same
143 145	142 143	142	as	as
152	153 154	152	N_{123}	N_{124}
241 243	243 245	241	and	
251 254	251 253 254	251 253	N_{234}	
312 314 315	312 315	314		
351 352 354	351 354	352		
412 415	412 413 415	413 415		
423 425	421 423	425		
514	512 513	514		
521 523	523 524	524		
534	532 534	531		

When all these nodes have been propagated, the new binary nodes are represented by the following R-matrices. N_{123} causes R_{13} to change :

$$R_{13}= \begin{matrix} 01001 \\ 10100 \\ 01010 \\ 00101 \\ 10010 \end{matrix}$$

N_{124} causes no changes
N_{125} causes R_{25} to change :

$$R_{25}= \begin{matrix} 01101 \\ 10110 \\ 01010 \\ 01101 \\ 10110 \end{matrix}$$

N_{234} causes R_{24} to change :

R_{24}= 01001
10100
01010
00101
10010

N_{345} causes R_{35} to change :

R_{35}= 01001
10100
01010
00101
10010

N_{235} causes no changes.

This covers the whole range of binary nodes. This is one way to proceed. Another one which minimizes the number of nodes, is the following. Because it is only necessary to have one path between each real constraint and the synthesizing constraint, only N_{123} is constructed. The other 3-ary nodes are not constructed. Consequently, only this node is propagated and causes changes in R_{13}.

Step 4

Following the first option, the node N_{1234} is constructed out of N_{123}, N_{124} and N_{234}

N_{123}	N_{124}	N_{234}
135	1352	1352
142	1423 1425	1425
152	1522	-
241	2411 2413	2413
251	2511 2514	2514
253	2531 2534	2531

314	3142 3144 3145	3142
352	3521 3522 3524	3524
413	4132 4135	4135
415	4152 4155	4152
425	4253 4255	4253
514	5144	-
524	5241 5243	5241
531	5314	5314

The last column is the result of the propagation of the three ternary nodes to N_{1234}. Propagating N_{1234} also causes changes in the respective ternary nodes. It summarizes to :

* 152 514 are removed from N_{123}. This leads to a change in R_{12} : entries 15 and 51 are removed.

* 145 152 241 314 315 351 352 425 514 523 are removed from N_{124}. By this way 13 31 35 53 are removed from R_{14}.

* 251 415 are removed from N_{234}. Changes to R_{34} : 15 51 are withdrawn.

In a similar fashion, N_{1235} and N_{2345} are constructed. They are listed below :

N_{1235}	N_{2345}
1354	1352
1423	1425
2415	2413
2534	2531
3145	3142
3521	3524
4132	4135
4251	4253
5243	5241
5312	5314

When these two nodes are propagated no changes in other nodes occur.

Step 5

The final synthesizing node N_{12345} is now constructed. Using the three 4-ary nodes it gives :

N_{1234}	N_{1235}	N_{2345}
1352	13522 13524	13524
1425	14252 14253	14253
2413	24133 24135	24135
2514	25141 25143 25144	-
2531	25311 25313 25314	25314
3142	31422 31425	31425
3524	35241 35244	35241
4135	41352 41353 41355	41352
4152	41522 41523 41525	-
4253	42531 42533	42531
5241	52413 52414	52413
5314	53142 53144	53142

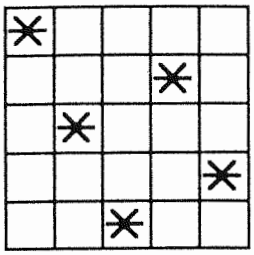
The third column is again the result of the propagation of all three 4-ary nodes. The propagation of N_{12345} is not necessary, because the solution can de derived from the synthesizing node. If this propagation is done, then some values of 4-ary, 3-ary and binary nodes are removed. The final results can be visualized in figure 9. As can be seen, we end up with the ten feasible solutions of this problem. Note the power of the horse : it is able to strike the queen without being in danger.

Let's now take again the turn to the other option proposed under step 3. Here only node N_{123} is constructed and propagated. It is now possible to move immediately to step 4, by constructing N_{1234} out of N_{123} and N_4. The following nodes must be propagated to N_{1234} : N_{14}, N_{24} and N_{34} :

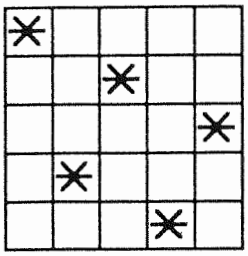
N_{123}	N_{14}	N_{24}	N_{34}	N_{1234}
135	01101	01010	11100	1352
142	01101	10101	00011	1425
152	01101	11010	00011	-
241	10110	10101	00111	2413
251	10110	11010	00111	2514
253	10110	11010	10001	2531
314	11011	01011	11000	3142
352	11011	11010	00011	3524
413	01101	01011	10001	4135
415	01101	01011	11100	4152
425	01101	10101	11100	4253
514	10110	01011	11000	-
524	10110	10101	11000	5241
531	10110	01010	00111	5314

Now N_{12345} is constructed :

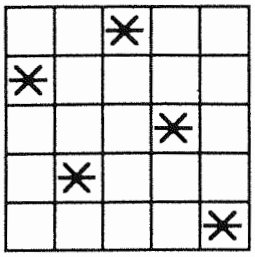
N_{1234}	N_{15}	N_{25}	N_{35}	N_{45}	N_{12345}
1352	01110	11011	11010	00011	13524
1425	01110	01101	10101	11100	14253
2413	10111	01101	01011	10001	24135
2514	10111	10110	01011	11000	-
2531	10111	10110	00111	00111	25314
3142	11011	01101	00011	00011	31425
3524	11011	10110	11000	11000	35241
4135	11101	01101	11100	11100	41352
4152	11101	01101	00011	00011	-
4253	11101	10110	10001	10001	42531
5241	01110	10110	00111	00111	52413
5314	01110	11011	11000	11000	53142



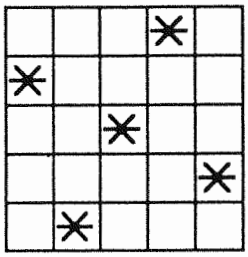
13524



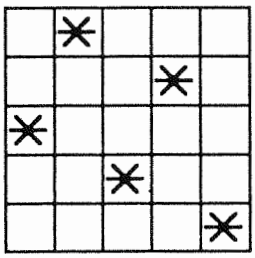
14253



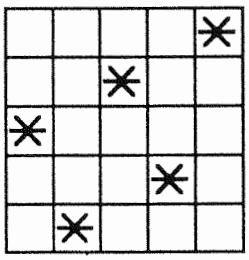
24135



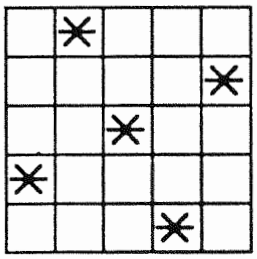
25314



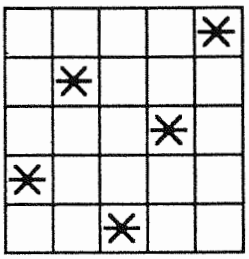
31425



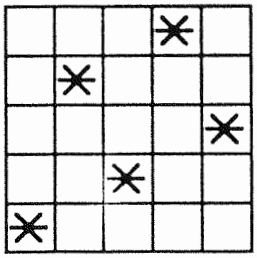
35241



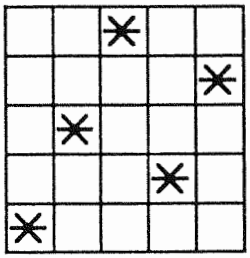
41352



42531



52413



53142

Figure 9 : final results of the 5-queens problem

In the same way, the ten feasible solutions are obtained, illustrating that the solution can be obtained with a minimal network.

C. Time complexity

Without loosing ourselves into details, some results are presented concerning the time complexity of the algorithm.

Like already stated, the n-queens problem is characterized by unary and binary constraints only. So, if one could find an algorithm that preserves consistency only using binary nodes, then the synthesizing node is linked directly to the binary nodes. Such an algorithm exists and an efficient one is developped by Mackworth (1977). He called this algorithm 'path consistency'. In another paper Mackworth and Freuder (1985) analyse the time complexity of his earlier developped algorithms. For path consistency he obtained a worst case complexity of $O(n^3)$, where n is the number of variables (unary nodes). Therefore the following time complexity results reported by Van Hentenryck on the n-queens problem (1989) are not that amazing :

N	sec
8	0.7
16	1
32	4.2
64	14.6
96	35.8

The fact that the above results are better than cubic, is probably explained by the fact that $O(n^3)$is a worst case bound and by additional features of the specific implementation. Using the n-queens problem as a proof for the excellent behaviour of the different packages is not valid if it is used to sustain the claim that it is appropriate for solving general integer programming or combinatorical problems. However from the supplier's point of view, it is undoubtedly a very good marketing argument. To elaborate on this issue somewhat further, let's conclude by the following :

CHIP tackles also resource constraint project planning (Dincbas, Simonis and Van Hentenryck, 1990). But if one knows that precedence relations are binary by definition and that the resources in the example used are only available in quantity one, so that at any given moment in time the resource can only be occupied by one activity (binary disjunctive between all consuming activities), then further investigation with other problems is necessary in this matter to validate the packages.

6. CONCLUSION

This paper intended to highlight the opportunities of a constraint satisfaction method named 'synthesizing algorithm'. Useful concepts were partially extended to be able to deal with general integer programming. A skeleton for a solution algorithm was developped. Further research topics lie ahead, such as :

- the implementation of the algorithm, for which some elements must still be formalized;
- extensive testing should be conducted;
- if succesfull, it should be further investigated whether the procedure can be adjusted for mixed integer and/or 0/1 problems;
- because interval arithmetic is still valid if the linear assumption is dropped, elaborations in this direction may be possible.

But as things are at the moment, these things still remain, I admit, probably wishfull thinking.

7. REFERENCES

ALEFELD G., HERZBERGER J. (1983), *Introduction to interval computations*, Academic Press.

CROWDER H., JOHNSON L., PADBERG M. (1983), *Solving large scale zero-one linear programming problems*, Operations Research 31, 803-834.

DAVIS E. (1987), *Constraint propagation with interval labels*, Artificial Intelligence 32,281-331.

DINCBAS M., SIMONIS H., VAN HENTENRYCK P. (1988), *Solving the car sequencing problem in constraint logic programming*, In European Conference on Artificial Intelligence (ECAI-88), Munich, W. Germany, 290-295.

DINCBAS M., SIMONIS H., VAN HENTENRYCK P. (1990), *Solving large combinatorial problems in logic programming*, Journal of Logic Programming 8,75-93.

DINCBAS M., SIMONIS H., VAN HENTENRYCK P., AGGOUN A., GRAF T., BERTHIER F. (1988), *The constraint logic programming language CHIP*, Proceedings of the International Conference on Fifth Generation Computer Systems 1988, 693-702.

DIRICKX Y.M.I, BAAS S.M., DORHOUT B. (1987), *Operationele research*, Academic Service.

FREUDER E.C. (1978), *Synthesizing constraint expressions*, Comm. ACM 21,958-966.

GUIGNARD M., SPIELBERG K. (1977), *Reduction methods for state enumeration integer programming*, Annals of Discrete Mathematics 1, 273-285.

GUIGNARD M., SPIELBERG K. (1981), *Logical reduction methods in*

zero-one programming, Operations Research 29, 49-74.

JOHNSON L., KOSTREVA M., SUHL H. (1985), *Solving 0-1 integer programming problems arising from large scale planning models*, Operations Research 33, 803-819.

LODWICK W. (1989), *Constraint propagation, relational arithmetic in AI systems and mathematical programs*, Annals of Operations Research 21, 143-148.

MACKWORTH A.K. (1977), *Consistency in networks of relations*, Artificial Intelligence 8,99-118.

MACKWORTH A.K. (1990), *Constraint satisfaction*, In Encyclopedia of Artificial Intelligence, S.C. Shapiro (ed.), John Wiley & Sons, 205-211.

MACKWORTH A.K., FREUDER E.C. (1985), *The complexity of some polynomial network consistency algorithms for constraint satisfaction problems*, Artificial Intelligence 25,65-74.

MOORE R.E. (1979), *Methods and applications of interval analysis*, SIAM publications, Philadelphia.

PAPADIMITRIOU C., STEIGLITZ (1982), *Combinatorial optimization: algorithms and complexity*, Prentice Hall.

VAN HENTENRYCK P. (1989), *A logic language for combinatorial optimization*, Annals of Operations Research 21,247-274.

VAN HENTENRYCK P., CARILLON J-P. (1989), *Generality versus specificity: an experience with AI and OR techniques,* In American Association for Artificial Intelligence (AAAI-86), St. Paul, MI, 660-664.

VAN HENTENRYCK P., DINCBAS M. (1986), *Domains in logic programming*, In AAAI-86, Philadelphia, PA, 759-765.

VAN WINCKEL F. (1990), *Lineaire programmatie en aanverwante methoden*, Acco Leuven.

WAGNER H.M. (1973), *Principles of operations research*, Prentice Hall.

ZIONTS S. (1972), *Generalized implicit enumeration using bounds on variables for solving linear programs with zero-one variables*, Naval Research Logistics quarterly 19, 165-181.

APPENDIX A

A primer on interval arithmetic

The basic notion of this kind of arithmetic is an interval, considered as an extension of a real number. An interval will be represented by a pair of real numbers, its endpoints. The arithmetic developped for this new kind of 'numbers', is relatively new but has had already a lot of applications : finite computer precision computations and data uncertainty problems among others. A real number is simply an interval with the two bounds identical, so all real arithmetic is a special case of interval arithmetic.

There are not so many books that deal with it. Two very good references are Moore (1979) and Alefeld and Herzberger (1983). The first one contains an extended bibliography. For purpose of this paper only the relevant concepts for bound tightening are presented for which I have heavily relied on Moore (1979).

1. Basics

An interval is a closed bounded set of 'real' numbers $[a,b]=\{x:a\leq x\leq b\}$. This can be considered as a number represented by an ordered pair of the endpoints, which are real numbers. In the same way a rational number is represented by a/b (an ordered pair of integers) or a complex number by a+ib (an ordered pair of real numbers). So, conceptually a set of real numbers is replaced by another number. X is an interval. Its bounds are $\underline{X}$ and $\underline{\underline{X}}$. Formally, $X=[\underline{X},\underline{\underline{X}}]$.

An n-dimensional vector, is in this context an ordered n-tuple of intervals $(X_1,X_2,...,X_n)$. Interval vectors are also denoted by capitals. For instance, if X is a two-dimensional interval vector. Then $X=(X_1,X_2)$ where $X_1=[\underline{X}_1,\underline{\underline{X}}_1]$ and $X_2=[\underline{X}_2,\underline{\underline{X}}_2]$. This is graphically a two dimensional rectangle of all points (x_1,x_2) such that $\underline{X}_1\leq x_1\leq\underline{\underline{X}}_1$ and $\underline{X}_2\leq x_2\leq\underline{\underline{X}}_2$

If the real number x is in the interval X, it is denoted as $x \in X$. Similarly,

if $x=(x_1,x_2,...,x_n)$ is a real vector and $X=(X_1,X_2,...,X_n)$ an interval vector then $x \in X$ is defined as $x_i \in X_i$ for i=1,2,...,n. The equality relation between two intervals holds if their bounds are equal.

The intersection of two intervals X and Y is empty if either $\underline{X}>\overline{Y}$ or $\underline{Y}>\overline{X}$. Otherwise, the intersection is again an interval

$$X \cap Y = [\max(\underline{X},\underline{Y}),\min(\overline{X},\overline{Y})].$$

If two intervals X and Y have a nonempty intersection, their union, $X \cup Y = [\min(\underline{X},\underline{Y}),\max(\overline{X},\overline{Y})]$, is also an interval. Other definitions can be given but are not relevant for this paper.

2. Interval arithmetic

Since intervals can be treated as numbers, operations can be defined. Like X+Y=Z, where $\underline{Z}=\underline{X}+\underline{Y}$ and $\overline{Z}=\overline{X}+\overline{Y}$. This consists of the set $X+Y=\{x+y:x \in X,y \in Y\}$ which is clearly an interval. The negative of an interval X is $-X=-[\underline{X},\overline{X}]=[-\overline{X},-\underline{X}]=\{-x:x \in X\}$. The difference of two intervals : $Z=Y-X=Y+(-X)=\{y-x:x \in X,y \in Y\}$, where $\underline{Z}=\underline{Y}-\overline{X}$ and $\overline{Z}=\overline{Y}-\underline{X}$. Also a reciprocal $1/X = \{1/x:x \in X\}$ can be defined. Also rules for multiplication and division can be formulated. Introducing functions is the next step. From these preliminaries the whole theory is built up.

3. Interval arithmetic applied in the refine procedure

The refine procedure can take advantage of the difference in structure between equalities and inequalities. When an inequality is tranformed into an equality using slack variables, it leads only to an enlargement of the refine procedure, without any additional benefits.

A. Take the linear equality :

$$\Sigma_{i\in PC} c_i x_i - \Sigma_{i\in NC} c_i x_i = c$$

where $PC=\{i:c_i$ has a + sign$\}$
$NC=\{i:c_i$ has a - sign$\}$
c_i is infact c_i without the sign
$x_i \in [l_i,u_i]$

then refine on x_j :

if $j \in$ **PC then**

$$x_j \in [\max(l_j, 1/c_j(c+\Sigma_{i\in NC} c_i l_i - \Sigma_{i\in PC\setminus\{j\}} c_i u_i)),$$
$$\min(u_j, 1/c_j(c+\Sigma_{i\in NC} c_i u_i - \Sigma_{i\in PC\setminus\{j\}} c_i l_i))]$$

if $j \in$ **NC then**

$$x_j \in [\max(l_j, 1/c_j(-c+\Sigma_{i\in PC} c_i l_i - \Sigma_{i\in NC\setminus\{j\}} c_i u_i)),$$
$$\min(u_j, 1/c_j(-c+\Sigma_{i\in PC} c_i u_i - \Sigma_{i\in NC\setminus\{j\}} c_i l_i))]$$

B. On the other hand the linear inequality :

if $\Sigma_{i\in PC} c_i x_i - \Sigma_{i\in NC} c_i x_i \leq c$

where $PC=\{i:c_i$ has a + sign$\}$
$NC=\{i:c_i$ has a - sign$\}$
c_i is infact c_i without the sign
$x_i \in [l_i,u_i]$

then refine on x_j :

if $j \in$ **PC then**

$$x_j \in [l_j, \min(u_j, 1/c_j(c+\Sigma_{i\in NC} c_i u_i - \Sigma_{i\in PC\setminus\{j\}} c_i l_i))]$$

if $j \in$ **NC then**

$$x_j \in [\max(l_j, 1/c_j(-c+\Sigma_{i\in NC\setminus\{j\}} c_i u_i - \Sigma_{i\in PC} c_i l_i)), u_j]$$

or if $\Sigma_{i\in PC} c_i x_i - \Sigma_{i\in NC} c_i x_i \geq c$

where $PC=\{i:c_i$ has a + sign$\}$
$NC=\{i:c_i$ has a - sign$\}$
c_i is infact c_i without the sign
$x_i \in [l_i,u_i]$

then refine on x_j :
if $j \in$ PC then
$x_j \in [\max(l_j, 1/c_j(c+\Sigma_{i\in NC} c_i l_i - \Sigma_{i\in PC\setminus\{j\}} c_i u_i)), u_j]$

if $j \in$ NC then
$x_j \in [l_j, \min(u_j, 1/c_j(-c+\Sigma_{i\in PC} c_i u_i - \Sigma_{i\in NC\setminus\{j\}} c_i l_i))]$

These rules can very easily be derived. They can be applied to every variable of every constraint. These refinement relations remain unchanged during the whole solution procedure.

APPENDIX B1

Solution of the problem :

min	**z**
s.t.	**$3x+y-z=0$**
	$3x+2y\geq 6$
	$5x-4y\leq 5$
	$2x-y\geq 1$
	$x,y\geq 0$ and integer

1. The refinement rules

C_1 $x : [\max\{l_x, 1/3(0+l_z-u_y)\}, \min\{u_x, 1/3(0+u_z-l_y)\}]$
$y : [\max\{l_y, 0+l_z-3u_x\}, \min\{u_y, 0+u_z-3l_x\}]$
$z : [\max\{l_z, 3l_x+l_y-0\}, \min\{u_z, 3u_x+u_y-0\}]$

C_2 $x : [\max\{l_x, 1/3(6-2u_y)\}, u_x]$
$y : [\max\{l_y, 1/2(6-3u_x)\}, u_y]$

C_3 $x : [l_x, \min\{u_x, 1/5(5+4u_y)\}]$
$y : [\max\{l_y, 1/4(-5-5l_x)\}, u_y] = [l_y, u_y]$ = no change

C_4 $x : [\max\{l_x, 1/2(1+l_y)\}, u_x]$
$y : [l_y, \min\{u_y, 2u_x-1\}]$

Now the refine procedure is applied for two passes. To simplify the notation, the reference to each of the constraints will be done with a number and to the variable with a letter.

Pass 1		Pass 2	
1x	$[0,\infty]$	1x	$[1,\infty]$
1y	$[0,\infty]$	1y	$[0,\infty]$
1z	$[0,\infty]$	1z	$[3,\infty]$
2x	$[0,\infty]$	2x	$[1,\infty]$
2y	$[0,\infty]$	2y	$[0,\infty]$

3x	[0,∞]		3x	[1,∞]
3y	[0,∞]		3y	[0,∞]
4x	[0.5,∞]=> [1,∞]	4x	[1,∞]	
4y	[0,∞]		4y	[0,∞]

No change anymore. Q={1}. LLB=GLB=3, LUB=GUB=∞

Now we arrived at the branching point. Option 1 and 2 are here out of question. So option 3 is taken. Q={1A}. An LP with bounds is run and it comes up with :

x = 1.14
y = 1.28
z = 4.71

The LP result is infeasible. LLB_{LP}=5. Q={2}. Propagation of [5,∞].

Pass 1

1x	[1,∞]
1y	[0,∞]
1z	[5,∞]
2x	[1,∞]
2y	[0,∞]
3x	[1,∞]
3y	[0,∞]
4x	[1,∞]
4y	[0,∞]

It quiesces. LLB=5, LUB=∞. The level is exhausted. Help=5 and so GLB becomes 5. At this decision point option 2 is tried. Q={3,4,5,6,7}. Back to propagation :

Pass 1

1x	[1,5/3]=> x=1
1y	[2,2]=> y=2
1z	[5,5]=5
2x	[1,1]=1
2y	[2,2]=2
3x	[1,1]=1
3y	[2,2]=2
4x	[1.5,1]=> infeasible

So this instantiation leads to an infeasibility. The initial choice of z=5 was not valid. Q={4,5,6,7}. The result is that we are again at the decision point. z is now instantiated at 6 :

Pass 1

1x	[1,2]
1y	[0,3]
1z	[6,6]=6
2x	[1,2]
2y	[0,3]
3x	[1,2]
3y	[0,3]
4x	[1,2]
4y	[0,3]

LLB=6, LUB=6. Q={5,6,7,4}. Here again the decision point turns up. The next node corresponds to the instantiation of z to 7.

Pass 1		Pass 2	
1x	[1,7/3]=> [1,2]	1x	[4/3,2]=2
1y	[1,4]	1y	[1,1]=1
z	[7,7]=7	1z	[7,7]=7
2x	[1,2]	2x	[2,2]=2
2y	[1,4]	2y	[1,1]=1
3x	[1,2]	3x	[2,1]=>infeasibl
e			
3y	[1,4]		
4x	[1,2]		
4y	[1,3]		

Due to the infeasibility, Q becomes {6,7,4}. The decision point is reached again. Arrived at the decision point again, z is instantiated to 8 and propagated :

Pass 1

1x	[1,8/3]=>[1,2]
1y	[2,5]
1z	[8,8]=8
2x	[1,2]
2y	[2,5]
3x	[1,2]
3y	[2,5]
4x	[3/2,2]=>[2,2]=2
4y	[2,3]

Pass 2

1x	[2,2]=2
1y	[2,2]=2
1z	[8,8]=8
2x	[2,2]=2
2y	[2,2]=2
3x	[2,2]=2
3y	[2,2]=2
4x	[2,2]=2
4y	[2,2]=2

This system quiesces definitely. LLB=8 and LUB=8. The GUB becomes now 8. Q={7,4,6}. The propagation of z=7 is in fact not necessary because we know with certainty that the LLB for the node will be larger than or equal to 9 which is larger than the GUB. So node 7 is removed from Q. Q={4,6}. The branching level is exhausted. Help=6 so GLB=6. The branching point is now reached. Option 1 is taken. 4 is removed from Q and 8 and 9 are added. Q={8,9,6}. First branching on x=1 and a second one on x=2, both will be sequentially propagated :

Pass a1

x	[1,1]=1
1y	[3,3]=3
z	[6,6]=6

...
Infeasibility on 4 :

4x	[2,1]=> infeasible

Q becomes {9,6}. The branching level is not exhausted, so the next node is propagated.

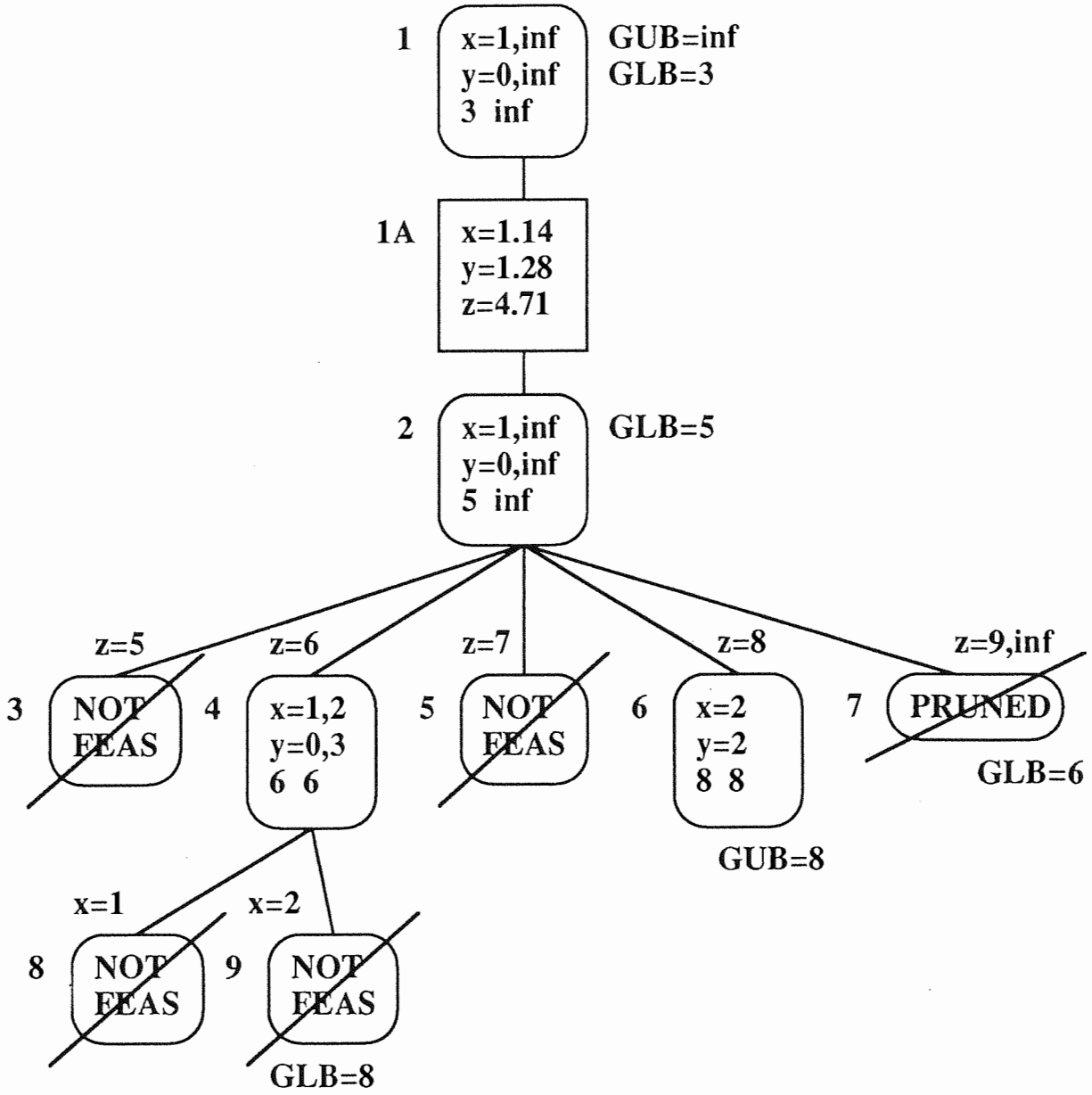
Pass b1

x	[2,2]=2
y	[0,0]=0
z	[6,6]=6

Infeasibility on 3 :

3x	[2,1]=> infeasible

Q becomes {6}. Now the branching level is exhausted. Help is equal to 8, so GLB=8. Because GLB=GUB and 6 is the only node with this characteristic, the procedure stops and gives the solution.


1
x=1,inf
y=0,inf
3 inf
GUB=inf
GLB=3
1A
x=1.14
y=1.28
z=4.71
2
x=1,inf
y=0,inf
5 inf
GLB=5
z=5
z=6
z=7
z=8
z=9,inf
3
NOT FEAS
4
x=1,2
y=0,3
6 6
5
NOT FEAS
6
x=2
y=2
8 8
GUB=8
7
PRUNED
GLB=6
x=1
x=2
8
NOT FEAS
9
NOT FEAS
GLB=8

APPENDIX B2

Solution of the problem :

max	**z**
s.t.	**$7x+9y-z=0$**
	$-1/3x+y\leq 2$
	$x+1/7y\leq 5$
	$x,y\geq 0$ and integer

The relations for this example :

1x	$[\max\{l_x,1/7(0+l_z-9u_y)\},\min\{u_x,1/7(0+u_z-9l_y)\}]$
1y	$[\max\{l_y,1/9(0+l_z-7u_x)\},\min\{u_y,1/9(0+u_z-7l_x)\}]$
1z	$[\max\{l_z,7l_x+9l_y)\},\min\{u_z,7u_x+9u_y)\}]$
2x	$[\max\{l_x,3(-2-l_y)\},u_x]=[l_x,u_x]$
2y	$[l_y,\min\{u_y,2+1/3u_x)\}]$
3x	$[l_x,\min\{u_x,5-1/7l_y)\}]$
3y	$[l_y,\min\{u_y,7(5-l_x)\}]$

As a first step, refine is executed :

Pass 1

1x	$[0,\infty]$
1y	$[0,\infty]$
1z	$[0,\infty]$
2x	$[0,\infty]$
2y	$[0,\infty]$
3x	$[0,5]$
3y	$[0,35]$

Pass 2

1x	[0,5]
1y	[0,35]
1z	[0,350]
2x	-
2y	[0,11/3]=[0,3]
3x	[0,5]
3y	[0,3]

Pass 3

1x	[0,5]
1y	[0,3]
1z	[0,62]
2x	-
2y	[0,3]
3x	[0,5]
3y	[0,3]

Here the system quiesces. Q={1}. Bounds are set : GUB=LUB=62 and GLB=LLB=0. Option 3 is taken and an LP is solved. The solution is :
x=4+4/7
y=3
z=59

Q={1A}. This solution is not completely integer. A LUB_{LP} is known : 59. The objective function is integer, so no fraction can be removed. However the new interval for z is now [0,59]. Q={2}. This can be propagated :

1x	[0,5]
1y	[0,3]
1z	[0,59]

These intervals won't change anymore, because the only constraint relating z to the other two is the first one. If this relation undergoes no change, then it is not necessary to look at the other constraints which involve only other variables. The reason is that the system for these given values already quiesced in the former pass. Note that LUB=59 and LLB=0. The decision point is reached. Next help is set to 59, thus GUB becomes 59.

At the branching point option 1 with regard to y or x can be taken. Let's take x. Q={3,4}. Node 3 corresponds to x=5. We now take the upper bound of the interval, because it is a maximum problem. Propagation is done :

Pass 1

1x	[5,5]=5
1y	[0,24/9]=[0,2]
1z	[35,59]
2x	-
2y	[0,2]
3x	[5,5]=5
3y	[0,0]=0

Pass 2

1x	[5,5]=5
1y	[0,0]=0
1z	[35,35]=35

For the same reason as above, the intervals won't change anymore. LLB=35=LUB. This solution (O=35) is feasible but we are not sure if it is optimal. So we derive a general lower bound on the optimal value

equal to 35. Q={4,3}. Let's now propagate the interval [0,4].

Pass 1

1x	[0,4]
1y	[0,3]
1z	[0,55]
2x	-
2y	[0,3]
3x	[0,4]
3y	[0,3]

Pass 2

1x	[0,4]
1y	[0,3]
1z	[0,55]

Nothing will change anymore. LUB=55 and LLB=0. Q={3,4}. The branching level is exhausted and the next node is of the same branching level. Help= 55, so GUB=55. The first node in Q is 3, but this can not be branched any further, so node 4 is taken. Let's now take option 2. Q={5,6,3}. Propagate node 5 which stands for z=55. The propagation :

Pass 1

1x	[4,4]=4
1y	[3,3]=3
1z	[55,55]=55
2x	-
2y	[3,3]=3
3x	[4,4]=4
3y	[3,3]=3

Pass 2 will not change anything. LLB=LUB=55 and also GLB=55. Q={6,5,3}. Node 3 can be pruned. Q={6,5}. GUB=GLB so we end up with an optimal feasible solution. It can be seen that not all nodes have their GUB=GLB. The current branching level is the level corresponding to the objective function, so all lower objective levels can be removed. Node 6 is pruned. Q={5}. This is the only node where GUB=GLB. Stop. The solution :

x=4
y=3
z=55

As a matter of control, $z \in [0,54]$. LUB≤54, which is lower than GLB of 55. This node would be pruned anyway.

Note that if above we did not instantiated x but y (for reason of the smallest interval) the solution went on like this :

Q={3,4}. Instantiate y=3 and propagate :

Pass 1

1x	[0,4]
1y	[3,3]=3
1z	[27,55]
2x	-
2y	[3,3]=3
3x	[0,4]
3y	[3,3]=3

Pass 2

1x	[0,4]
1y	[3,3]=3
1z	[27,55]

It quiesces. LUB=55 and LLB=27. Q={4,3}. Now the alternatives of y ∈ [0,2] must still be investigated. Propagation of this interval :

Pass 1

1x	[0,5]
1y	[0,2]
1z	[0,53]

LLB=0 and LUB=53. Q={3,4}. The branching level is exhausted with not all the nodes infeasible. Help=55 and so becomes GUB=55. The branching point is reached. Q={5,6,4}. If an instantiation on x=4 is made then we will end up with :

Pass 1

1x	[4,4]=4
1y	[3,3)=3
1z	[55,55]=55

This will not lead to an infeasibility. Now a feasible solution is obtained. LLB=LUB=55. GLB=55. Q={6,5,4}. Node 4 can be pruned. Q={6,5}. GUB=GLB so one optimal solution is found. Alternatives are of interest and not all nodes have GUB=GLB. Current unexhausted level is taken and being not the objective level, the rest is gathered in an interval. In this particular case node 6 is replaced by the same node. Propagation leads to:

Pass 1:

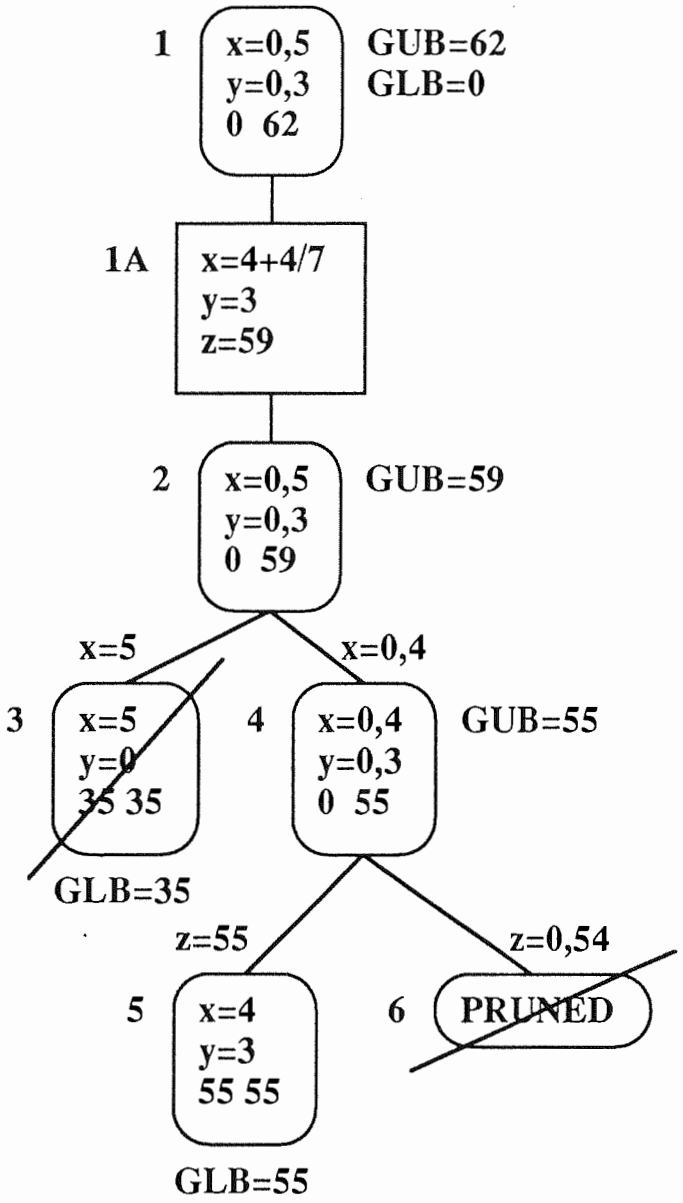
1x	[0,3]
1y	[3,3]=3
1z	[27,48]
2x	-
2y	[3,3]=3
3x	[0,3]
3y	[3,3]=3

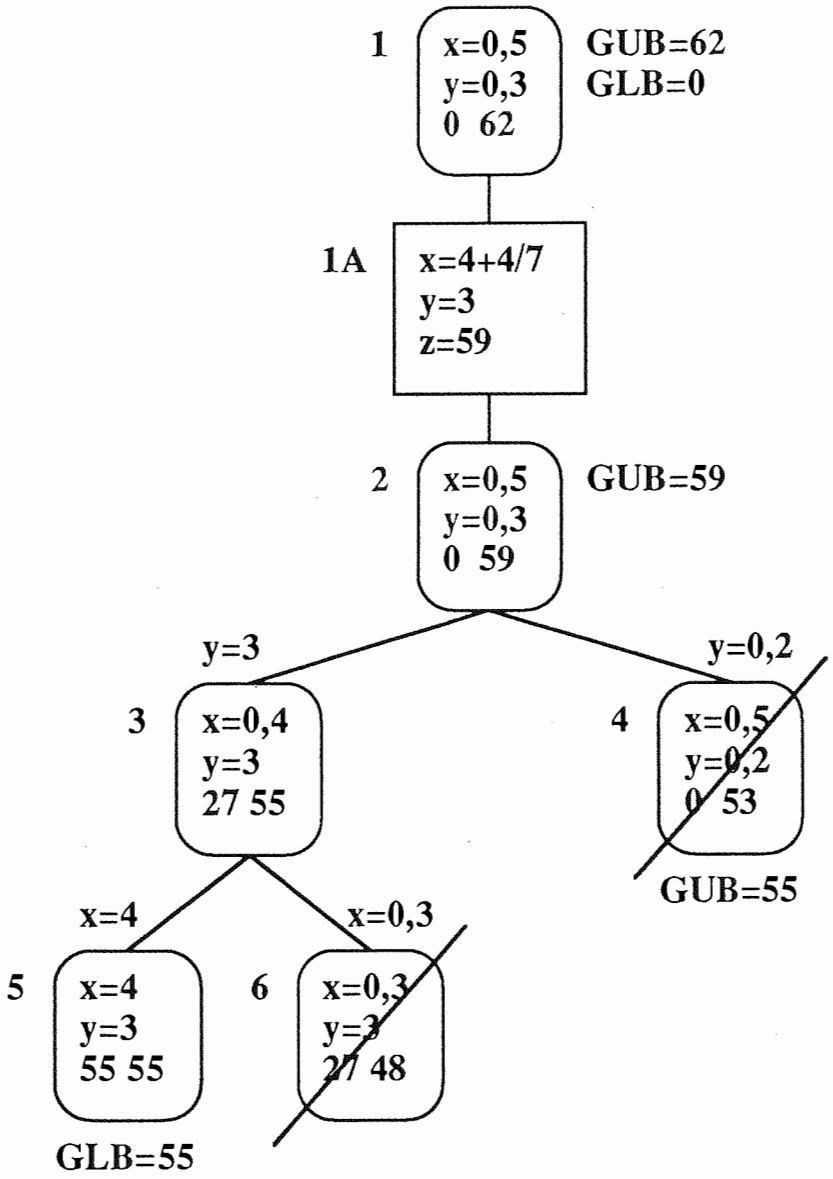
Pass 2

1x	[0,3]
1y	[3,3]=3
1z	[27,48]

It quiesces. LLB=27 and LUB=48. This LUB=48 is smaller than GLB=55, so Q={5}. All nodes have now GUB=GLB. Stop.

Note that in both cases only one LP was necessary to solve the problem.


1
x=0,5
y=0,3
0 62
GUB=62
GLB=0
1A
x=4+4/7
y=3
z=59
2
x=0,5
y=0,3
0 59
GUB=59
x=5
x=0,4
3
x=5
y=0
35 35
GLB=35
4
x=0,4
y=0,3
0 55
GUB=55
z=55
z=0,54
5
x=4
y=3
55 55
GLB=55
6
PRUNED


1
x=0,5
y=0,3
0 62
GUB=62
GLB=0
1A
x=4+4/7
y=3
z=59
2
x=0,5
y=0,3
0 59
GUB=59
y=3
y=0,2
3
x=0,4
y=3
27 55
4
x=0,5
y=0,2
0 53
GUB=55
x=4
x=0,3
5
x=4
y=3
55 55
6
x=0,3
y=3
27 48
GLB=55

APPENDIX B3

Solution of the problem :

max	v
s.t.	3x+3y+13z-v=0
	-3x+6y+7z≤8
	6x-3y+7z≤8
	x,y,z≥0 and integer

The rules for this problem are :

1x
$[\max\{l_x, 1/3(0+l_v-3u_y-13u_z)\}, \min\{u_x, 1/3(0+u_v-3l_y-13l_z)\}]$
1y
$[\max\{l_y, 1/3(0+l_v-3u_x-13u_z)\}, \min\{u_y, 1/3(0+u_v-3l_x-13l_z)\}]$
1z
$[\max\{l_z, 1/13(0+l_v-3u_x-3u_y)\}, \min\{u_z, 1/13(0+u_v-3l_x-3l_y)\}]$
1v
$[\max\{lv, 3l_x+3l_y+13l_z)\}, \min\{u_v, 3u_x+3u_y+13u_z)\}]$

2x $[\max\{l_x, 1/3(-8-6l_y-7l_z)\}, u_x] = [l_x, u_x]$
2y $[l_y, \min\{u_y, 1/6(8+3u_x-7l_z)\}]$
2z $[l_z, \min\{u_z, 1/7(8+3u_x-6l_y)\}]$

3x $[l_x, \min\{u_x, 1/6(8+3u_y-7l_z)\}]$
3y $[\max\{l_y, 1/3(-8-6l_x-7l_z)\}, u_y] = [l_y, u_y]$
3z $[l_z, \min\{u_z, 1/7(8+3u_y-6l_x)\}]$

$x,y,z \in [0,5]$ and $v \in [0,\infty]$

The first propagation phase :

1x	[0,5]
1y	[0,5]
1z	[0,5]
1v	[0,95]

2x	[0,5]
2y	[0,23/6]=[0,3]
3z	[0,23/7]=[0,3]
3x	[0,17/6]=[0,2]
3y	[0,3]
3z	[0,17/7]=[0,2]

Pass 2

1x	[0,2]
1y	[0,3]
1z	[0,2]
1v	[0,41]
2x	[0,2]
2y	[0,14/6]=[0,2]
2z	[0,2]
3x	[0,14/6]=[0,2]
3y	[0,2]
3z	[0,2]

Pass 3

1x	[0,2]
1y	[0,2]
1z	[0,2]
1v	[0,38]
2x	[0,2]
2y	[0,2]
2z	[0,2]
3x	[0,2]
3y	[0,2]
3z	[0,2]

It queiesces. LLB=GLB=0 and LUB=GUB=38. Q={1}. The next step is the usual LP run. Q={1A}. This ends up with :
x=2
y=2
z=2/7
v=15+5/7

This makes it possible to construct a local upper bound of 15. Q={2}. Consequently the interval of v is set to [0,15]. Propagation :

Pass 1

1x	[0,2]
1y	[0,2]
1z	[0,15/13]=[0,1]
1v	[0,15]
2x	[0,2]
2y	[0,2]
2z	[0,1]
3x	[0,2]
3y	[0,2]
3z	[0,1]

Pass 2

1x	[0,2]
1y	[0,2]
1z	[0,1]
1v	[0,15]

The system quiesces here. LLB=0 and LUB=15. The branching level is exhausted, so help=15 and by this way GUB=15. Just like in all other cases three options are open. Let's now run a second LP run. Q={2A}.

The results are :
x=0
y=0
z=1
v=13

GLB=13. Q={3}. The propagation of [13,15] is now necessary. The result is :
x=0
y=0
z=1
v=13

The propagation is left out for the unconvinced reader. LUB=LLB=13. The branching level is again exhausted with not all infeasibilities, so help=13 and GUB=13. GLB=GUB and an optimal solution is found. There are no more candidates in Q, so stop.

Instead of running a second LP run, something else could be done. An instantiation on z looks promising, because there are only two values in the interval. Branching is chosen. Q={3,4}. Propagation : let's instantiate z to 0 :

Pass 1

1x	[0,2]
1y	[0,2]
1z	[0,0]=0
1v	[0,12]

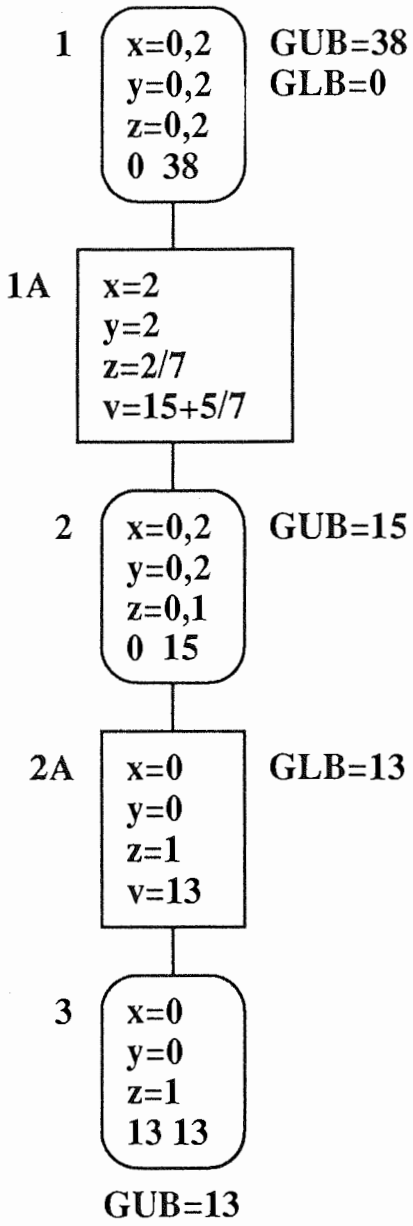
This does not change anymore. LLB=0 and LUB=12. Q={4,3}. If the other instantiation is propagated we get :

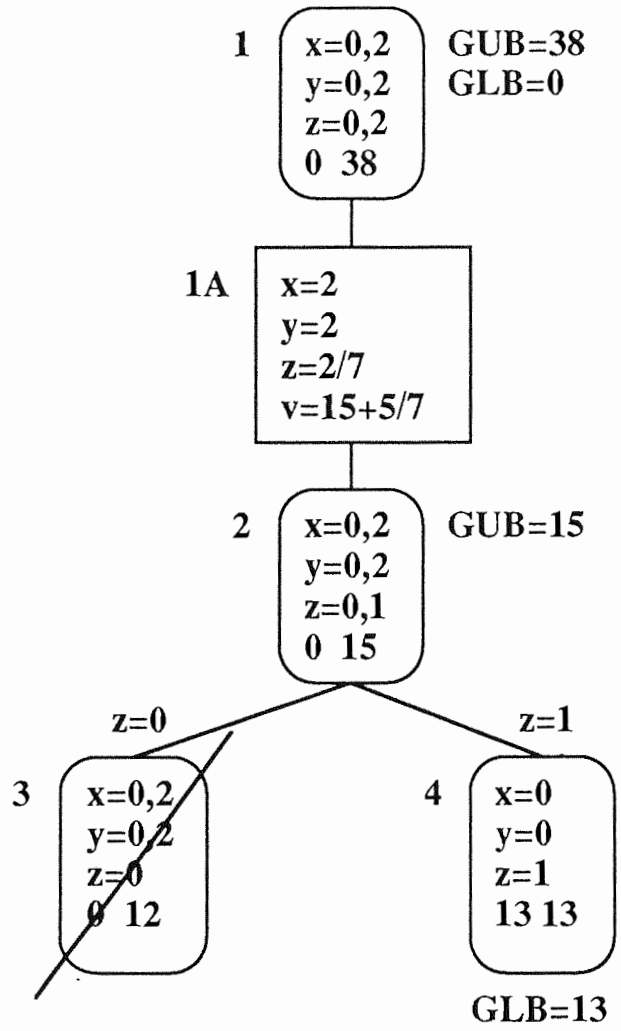
Pass 1

1x	[0,2/3]=>[0,0]=0
1y	[0,2/3]=>[0,0]=0

1z	[1,15/13]=>[1,1]=1
1v	[13,13]=13

This result will not lead to an infeasibility when the rest of the propagation is done. LLB=LUB=13. Because of feasibility, GLB=13. Q={3,4}. LUB of node 3 is smaller than GLB, so Q={4}. The branching level is exhausted with not all nodes infeasible. Help=13. GUB=13. GUB=GLB and an optimal solution is obtained. It is the only member of Q, so stop.


1
x=0,2
y=0,2
z=0,2
0 38
GUB=38
GLB=0
1A
x=2
y=2
z=2/7
v=15+5/7
2
x=0,2
y=0,2
z=0,1
0 15
GUB=15
2A
x=0
y=0
z=1
v=13
GLB=13
3
x=0
y=0
z=1
13 13
GUB=13


1
x=0,2
y=0,2
z=0,2
0 38
GUB=38
GLB=0
1A
x=2
y=2
z=2/7
v=15+5/7
2
x=0,2
y=0,2
z=0,1
0 15
GUB=15
z=0
z=1
3
x=0,2
y=0,2
z=0
0 12
4
x=0
y=0
z=1
13 13
GLB=13