The Convex Hull Heuristic for Nonlinear 0-1 Programming Problems with Linear Constraints

Monique Guignard and Aykut Ahlatcioglu

University of Pennsylvania

Abstract. The Convex Hull Heuristic (CHH) is a heuristic for 0-1 integer programming problems with a nonlinear objective function and linear constraints. It is a matheuristic in two ways: it is based on the mathematical programming algorithm called simplicial decomposition [1], and at each iteration, it calls a linear integer programming solver. Its purpose is to produce quickly near optimal solutions for convex and nonconvex problems. It is multi-start: following each restart, it repeatedly solves 0-1 programming problems with the original constraints and with a linear objective that depends on the previous iterations. We have tested it on a number of hard quadratic 0-1 optimization problems and present numerical results for quadratic assignment problems (QAP), generalized quadratic assignment problems (GQAP), cross-dock door assignment problems (CDAP), quadratic knapsack problems (QKP) and quadratic knapsack problems with a cardinality constraint (E-kQKP). We compare solution quality and solution times with results from the literature, when available.

Keywords: nonlinear 0-1 integer programming, simplicial decomposition, quadratic 0-1 programs with linear constraints, primal relaxation, convex hull relaxation, CHR, convex hull heuristic, CHH

1 Introduction

The Convex Hull Heuristic (CHH) method is a fast matheuristic that can be applied to nonlinear 0-1 optimization problems with linear constraints. It can be used indifferently for convex or nonconvex objective functions (in the case of minimization), as long as problems with the same constraint set but with a linear objective function are easy to solve. It is in general a multi-start heuristic, but for particularly well-behaved problem types, such as quadratic 0-1 knapsack problems, it can be single start. It always terminates after a finite number of iterations.

The CHH algorithm is loosely based on simplicial decomposition (SD) [1], with the feasible region equal to the convex hull of all 0-1 feasible solutions. This convex hull, constant since the constraints always remain the same, is a polyhedron that is constructed progressively as more and more of its 0-1 extreme points are generated. Each iteration of CHH requires one continuous NLP with one linear constraint and a number of variables (the weights in the convex

hull) that increases through the iterations, and one 0-1 MIP with all original constraints and a linear objective function that changes with the iterations. In addition, to speed up the generation of 0-1 (extreme) points, one can make use of the solution pool of CPLEX and choose the incumbent with best MINLP objective value.

Our initial use of simplicial decomposition was in computing bounds for 0-1 NLIP's with linear constraints. Those bounds were computed over the intersection of the convex hull of the integer points satisfying a subset of constraints, called the "kept constraints" thereafter, with the polyhedron of the continuous solutions to the other constraints. Introduced in [3], this primal relaxation, equivalent to Lagrangean relaxation for linear problems, yields different bounds for nonlinear problems. This relaxation works in the primal space and considers separately the kept constraints and the other constraints, like in Lagrangean relaxation for integer linear problems. The real step forward was discovered independently by V. Albornoz [4] and A. Ahlatcioglu as reported in [5]. It allows the construction of a Convex Hull Relaxation where all constraints are kept. This relaxation was shown to be efficient for convex 0-1 problems in [6], but most hard quadratic 0-1 problems in the literature, such as QAP's, usually have nonconvex objective functions, limiting the usefulness of the approach.

There is however one important feature of simplicial decomposition when the feasible region is the convex hull of all 0-1 feasible points: even for problems with a nonconvex objective function, it will generate one new 0-1 feasible point at each iteration, more if one uses CPLEX to solve the subproblems. Even in the nonconvex case, it is possible to use the convex hull approach, not to get valid bounds, but in a heuristic fashion, as a generator of 0-1 feasible solutions. By carefully designing the initialization phase, one can generate good, most of the time excellent feasible solutions, and not infrequently optimal ones, for instances where the optimum is known.

In Section 2, we describe the algorithm. In Section 4, we present a sample of results from our extensive computational testing for a variety of problem types. Finally, in Section 5, we discuss possible extensions and conclusions.

2 The Convex Hull Heuristic Method

In [5], a relaxation method called convex hull relaxation (CHR) was introduced for computing bounds tighter than the continuous lower bound, for convex mixed integer nonlinear programming (MINLP) with linear constraints. By applying CHR to such problems, good integer feasible solutions are generated, as well as a lower bound on the optimal value. Suppose the nonlinear integer program (NLIP) is

$$\text{Minimize} \quad f(x) \tag{1}$$

subject to

$$Ax \leq b \tag{2}$$

$$x \in B \tag{3}$$

with B the set of all 0-1 n-vectors, A and b the constraint matrix and the right hand side vector, respectively, of a set of linear constraints in x. Let H represent the convex hull of the 0-1 feasible solutions of (NLIP). This set will play an important role in what follows. Define the convex hull relaxation (CHR) of NLIP to be the problem of minimizing $f(x)$ over the convex hull H of the 0-1 feasible solutions of (NLIP). We will apply SD to problem (CHR). Following [1], we define the Convex Hull Subproblem (CHS) at the k-th iteration of SD as that of minimizing over H the original objective function $f(x)$ linearized at a feasible point $x(k)$ of (CHR). Note that (CHS) is a linear program, then it is equivalent to problem (IPS), the "integer programming subproblem", which is (CHS) restricted to its 0-1 feasible solutions. Indeed, let $y(k)$ be the optimal solution of (CHS), it depends on $x(k)$ because the linearized function

$$H(x) = f(x(k)) + \bigtriangledown f(x(k)).(x - x(k)) \tag{4}$$

depends on the linearization point $x(k)$. This objective function being linear, this is equivalent to solving (IPS), which is a well-defined (0-1) linear program. To summarize, at each iteration, one solves a (0-1) linear program with a different linearized function H(x), while the constraint set remains the same. The solution to (IPS), $y(k)$, is an extreme point of the convex hull, unless $x(k-1)$ is optimal for the convex hull relaxation (CHR). In the former case, the new extreme point is used to expand the search area at the next iteration. Each iteration produces a new vertex $y(k)$ of the integer convex hull and then solves the NLP over the convex hull of $x(0)$, $y(1)$,...,$y(k)$. Each time the algorithm adds a new point $y(k)$, it goes from a line segment to a triangle to a quadrangle etc. Then the convex hull of $x(0)$, $y(1)$,..., y(k), grows, and approximates better the integer convex hull. The area searched gets increasingly larger, and if the objective is convex, SD will at some point have found enough points for the convex hull of $x(0)$, $y(1)$,..., y(k), to contain the global optimum. In the worst case, it will need to generate all extreme points of the integer convex hull. If $f(x)$ is convex, the algorithm will converge after, say, p iterations to a point $x(p)$ in the convex hull of the set of 0-1 points x feasible for (NLIP). This happens when $f(x(p\text{-}1)) = f(x(p))$, i.e., the latest point $x(p)$ has not improved the NLIP optimum, or the gradient is 0 at $x(p)$. $f(x(p))$ is a valid lower bound on the integer optimum, because $f(x(p))$ is equal to the minimum of $f(x)$ over H. A by-product of CHR is that it always produces a number of feasible integer solutions, $y(1)$, ..., $y(p)$, we can compute $f(y(1))$, $f(y(2))$, ..., $f(y(p))$, and keep the best, called y^*, as the best integer feasible solution found. Whether the problem is convex or not, CHR works as a primal heuristic. In the convex case (min convex or max concave), in addition to producing a, usually very good, 0-1 feasible solution, CHR also produces a valid bound on the optimum, $f(x(p))$.

3 Displaying CHR on a figure

The figure is meant to give a feeling for what the iterations look like. These iterations lead to at least a local minimum solution. The algorithm goes as

follows; x(0) is the initial linearization point. Solve the linear integer program (IPS) and get y(1). Solve the original NLP over the convex hull of $x(0)$ and $y(1)$, get x(1). Solve the new linear integer program (IPS) and get $y(2)$. Solve the original NLIP over the convex hull of $x(0)$, $y(1)$ and $y(2)$, and get $x(2)$, which is at least a local optimal solution. In the given example, the CHR heuristic has generated two integer feasible solutions $y(1)$ and $y(2)$.

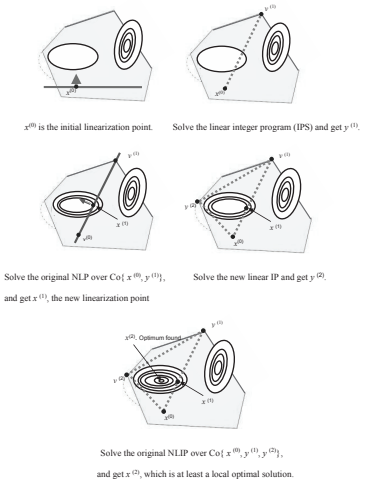

Fig. 1. Illustration of iterations.

If the NLIP in this example had had a convex function (or a concave function in a max problem), CHR would have produced both a valid bound on the optimum, $f(x(p))$, and a feasible integer solution y^*, which is the best among the feasible solutions found, $y(1)$, ... , $y(p)$. If the problem is not convex, then a valid bound cannot be guaranteed, but one still has a heuristic for generating feasible solutions $y(1)$, ... , $y(p)$, and the best solution found y^* is kept.

3.1 Implementation of The Convex Hull Heuristic

The implementation of CHR requires a starting strategy, doing restarts if necessary from different initial linearization points, keeping a balance between optimality and runtime, and using CPLEX solution pool.

The initial iteration of SD as applied to 0-1 nonlinear problems with linear constraints requires a first linearization point. After testing extensively different approaches, it became clear that choosing directly the initial linearization point was more difficult and less effective than solving LP's over the linear constraints, tilting the gradient of the linear objective function to cover a wide area of the

Table 1. GQAP instances. instance = instance. nbval = number of 0-1 variables. found = Optimum found?. percnt = optimality percentage error. TM = Time for MIP. TN = Time for NLP. TT = total time in seconds.

instance	nbval	found	percnt	TM	TN	TT
20-15-35	300	y	0	5	2	7
30-06-95	180	y	0	11	6	17
30-07-75	210	y	0	17	4	21
30-08-55	240	y	0	3	2	5
30-10-65	300	y	0	17	5	23
30-20-35	600	y	0	16	6	22
30-20-95	600	n	0.28	219	2	221
35-15-55	525	y	0	14	8	22
35-15-95	525	y	0	334	13	347
40-10-65	400	y	0	18	12	29
50-10-65	500	y	0	100	5	105
50-10-75	500	y	0	29	7	36

feasible region, and for each direction solve a max and a min problem. This gave a variety of initial linearization points, and we used the same 8 choices (times 2 for min and max) for all multi-start runs. Another option in our code concerns running times of the linear integer subproblems. Without imposing a time limit, within a run, it often happened that one or more of the linear MIPS took many times as long as the average of the others. If we had been looking for a valid lower bound, we would have had to let these problems run to optimality, but in a nonconvex heuristic mode, we decided to impose a limit for each rum of no more than a few times the average observed for similar instances of similar sizes. Additionally, we made use of the solution pool of CPLEX to look at the incumbents of each BB run. We made CPLEX compute their quadratic objective function values at basically no additional cost, and we kept the best overall solution. For most instances, the best solution found came from a solution pool. Finally, contrary to the general feeling concerning SD, we were very surprised to see that the number of SD iterations never exceeded 100, and most of the time was in the 20-30 range. So even though we had implemented RSD (Restricted Simplicial Decomposition), [2], what was used was really the original von Hohenbalken's SD. This means that the algorithm always converged in a relatively small number of iterations. In other words, it was very fast!

4 Computational results

We tested the CHH heuristic on a number of difficult quadratic 0-1 problems with linear constraints: GQAP (generalized quadratic assignment problem), QAP (quadratic assignment problem), CDAP (cross-dock door assignment problem), E-kQKP (quadratic knapsack problem with a cardinality constraint), and QKP (quadratic knapsack problem). We will present the results separately for each problem type. An important point is that for some problems, well known heuris-

tics are specific to the problem structure, while CHH is essentially a general purpose heuristic, which needs very little effort to convert to another problem type. All implementations were done in GAMS, on either a Thinkpad laptop T431S, or a unix workstation with similar characteristics.

4.1 GQAP

The generalized quadratic assignment problem is the quadratic equivalent of the generalized assignment problem. It was introduced by Lee and Ma [7]. We are using data from [9]. The instances range from 180 to 600 0-1 variables. Times are in seconds on the Thinkpad. Notice that for these instances, we did not put a limit on the time spent on the MIP subproblems.

4.2 CDAP

The cross-dock door assignment problem (CDAP) is the problem of assigning incoming trucks to unloading doors and outgoing trucks to loading doors in a rectangular cross-dock. After being sorted according to destination, unloaded goods have to be transported, often manually, in carts, from incoming doors on one side to loading doors on the other, and the distance from door to door affects labor costs. The assignment problem has to be solved before the beginning of a shift, taking into account the trucks that are already there, and it needs to be solved again quickly whenever trucks arrive leave, unloaded or loaded, as the situation in the cross-dock may have changed substantially.

The CDAP is a special case of the GQAP but with many infeasible assignments: incoming trucks are normally not assigned to outbound doors, nor outgoing trucks to unloading doors. Assigning infinite costs to these assignments is one possibility, but may lead to numerical difficulties. We consider a sparse subset of feasible assignments from the start.

An early presentation [8] mentioned a preliminary implementation of CHH for the CDAP over small instances. We use here a realistic dataset generated by D. Cardoso da Silva [10]. We give results for the largest instances tested. An instance type B labeled 100 by 50 for instance is one of the largest instances, with 100 trucks incoming and 100 outgoing, and 50 doors on each side. No instance beyond 15 by 10 has yet been solved optimally. Without known optimal solutions, and for comparison purposes, We are presenting our results in parallel with those of Cardoso's. He designed a local search heuristic for the CDAP, pingpong-ing between fixing the assignments on one side and solving on the other side, and reversing the order. This yields two series of tests, that differ slightly in the way each iteration starts. For smaller problems (with up to 400 0-1 variables, Cardoso's and CHH are basically indistinguishable in solution quality and running time. For the large problems, on the average, CHH solutions were 0.5 percent worse than the better value of Cardoso's two implementations. CHH solution times, however, were much shorter, as seen in the following table (these instances were run on the same machine). This may be important if trucks are waiting to be assigned.

Table 2. CDAP instances. instance = instance. nbval = number of 0-1 vars. Card best = Cardoso best value. Card T = Cardoso time . CHH best = CHH best value. CHH T = CHH time. prcnt = optimality percentage error

instance	nbval	Card best	Card T	CHH best	CHH T	prcnt
SetB-25x10S5	500	49144	194	49904	106	+1.55
SetB-25x10S20	500	48215	85	48338	120	+026
SetB-50x10S5	1000	191773	4039	192114	178	+0.18
SetB-50x10S15	1000	188006	3363	187753	164	-0.13
SetB-50x10S30	1000	83961	1652	184532	186	+0.31
SetB-50x20S5	2000	238048	7074	240288	504	+ 0.94
SetB-50x30S20	3000	266199	2467	264971	390	+0.45
SetB-75x20x15	3000	514760	9778	513166	602	-0.31
SetB-75x30S10	4500	636697	6898	636740	2088	+0.38
SetB-75x30x15	4500	620356	7481	617984	1455	-0.11
SetB-100x20x20	4000	921746	4938	925019	360	+0.36
SetB-100x30S30	6000	1052682	5038	1057666	900	+0.47

4.3 QAP

The quadratic assignment problem (QAP) is the quadratic equivalent of the assignment problem. While the linear assignment problem is easy to solve, the QAP is probably one of the most difficult quadratic 0-1 problems. The state of the art for the Nugent data set is still 30x30, or 900 0-1 variables.

The instances are taken from QUAPLib. The instance sizes range from 12 to 36. We report on the instances for which we found the optimal solution, plus the largest instance for each category tested. Times were so small that they were not recorded, indeed the integer subproblems are actually linear assignment problems and take no time.

4.4 QKP

For the quadratic 0-1 knapsack problem, we used Pisinger at al.'s dataset [12]. We ran the first instance for each density and size up to 300 0-1 variables. For this problem type, we quickly realized that it was a waste of time to run SD multiple times. First, we could choose the origin as the initial feasible point, as in this case it is feasible. This point provided us with optimal or very close to optimal solutions every time. The results were so good that we did not try other starting points. Only for this problem type however was it clearly a good choice, indeed For all other problem types mentioned above, each of the 8x2 possible restart types was the only one to produce the best solution for at least a few instances. Actually some other starting points were eliminated along the way because they did not produce interesting feasible solutions. Time-wise, on the Thinkpad T431s, the longest time for running CHH was approximately 15 seconds. Notice that it is our only maximization problem.

Table 3. QAP instances . instance = instance . size = number of rows (and columns) . avg = average number of SD iterations per pass. found = Optimum found? . prcnt = optimality percentage error

instance	size	avg	found	prcnt
bur26c	26	6.4	Y	0
bur26d	26	6.4	Y	0
bur26e	26	6.4	Y	0
bur26f	26	6.4	Y	0
bur26h	26	6.4	N	0.1
kra32	32	43.3	N	2.9
lipa30b	30	13.5	Y	0
nug15	15	20.7	Y	0
nug16b	16	26.6	Y	0
nug17	17	21.2	Y	0
nug20	20	21.6	Y	0
nug30	30	25.3	N	0.7
ste36c	36	46.3	N	4.1
tai12a	12	17	Y	0
tai25b	25	36.4	N	0.5

5 Extensions and conclusions

The convex hull heuristic is based on simplicial decomposition, and generates feasible solutions provided by linearized versions of the original quadratic objective function. At each iteration, one solves one nonlinear continuous problem with an increasing number of variables to generate the new linearization point (this part is very fast), and a linearized 0-1 problem to generate a new extreme point of the integer convex hull of the 0-1 feasible solutions (this part takes more time, except for the QAP, because one has to solve MIP problems). The algorithm converges (i.e., stops) when SD produces the same linearization point twice in a row. It is a generic approach, that can easily be adapted to other quadratic 0-1 problems. So far, the number of iterations has always be less than one hundred. Possible directions for improvement might be in the selection of the initial linearization point, possibly adapted to the problem type. In the case of a convex minimization problem, one would also obtain a valid lower bound if every 0-1 subproblem is solved to optimality.

References

1. von Hohenbalken, B.: Simplicial decomposition in nonlinear programming algorithms. Mathematical Programming 13, 49–68 (1977)
2. Hearn D.W., Lawphongpanich S., Ventura J.A.: Finiteness in restricted simplicial decomposition. Op. Res. Letters 4, 125–130 (1985).
3. Guignard, M.: Primal relaxations for integer programming. Invited plenary session, VII CLAIO, Santiago, Chile (July 1994).Also Technical Report 94-02-01, University of Pennsylvania, OPIM Department (1994).

Table 4. QKP instances. dens = instance density. nb = number of variables. opt = optimal value. found = optimum found? . best = best value found. prcnt = optimality percentage error

dens	nb	opt	found	best	prcnt
100	200	623145	Y	same	0
100	250	3036287	Y	same	0
100	300	3539021	Y	same	0
75	200	75588	Y	same	0
75	250	1374797	Y	same	0
75	300	924906	N	924750	0.0169
50	200	427428	N	427380	0.0112
50	250	1550630	Y	same	0
50	300	194556	Y	same	0
25	200	85771	Y	same	0
25	250	704437	Y	same	0
25	300	1116454	Y	same	0

4. Albornoz, V.: Diseno de Modelos y Algoritmos de Optimizacion Robusta y su Aplicacion a la Planicacion Agregada de la Produccion. Ph.D. thesis, Universidad Catolica de Chile, Santiago, Chile (1998).
5. Ahlatcioglu,A., Guignard,M.: The convex hull relaxation for nonlinear integer programs with linear constraints. Technical report,University of Pennsylvania, The Wharton School, OPIM Department, Philadelphia, PA (2007/2009),latest revision January 2009.
6. Ahlatioglu, A., Bussieck, M., Esen, M., Guignard, M., Jagla, J.-H.,Meeraus A.: Combining QCR and CHR for convex quadratic pure 01 programming problems with linear constraints. Ann. Oper. Res. 199:33-49 (2012).
7. Lee, Chi-Guhn, Ma, Zhong. The Generalized Quadratic Assignment Problem. Research Report, Department of Mechanical and Industrial Engineering, University of Toronto (2003)
8. Guignard, M., Hahn, P.M., Pessoa, A.A., Cardoso da Silva, D.: Algorithms for the cross-dock door assignment problem. In: Proceedings of the 4th International Workshop on Model-Based Metaheuristics. pp. 1–12. Angra dos Reis, Brazil (2012)
9. Cordeau, J.-F., Gaudioso, M., Laporte, G., Moccia, L. : A memetic heuristic for the generalized assignment problem, INFORMS Journal on Computing 18, 433–443 (2006).
10. Cardoso da Silva, D. : Uma heuristica hibrida de busca em vizinhanca larga para o problema de atribuicao de portas de cross-dock. Universidade Federal Fluminense, Escola de Enghenaria, Mestrado em Engenharia de Producao. (2013).
11. Letocart, L., Plateau, G. Private communication. (2015–16).
12. Caprara, A., Pisinger, D., Toth, P.: Exact solution of the Quadratic Knapsack Problem. INFORMS Journal on Computing, 11, 125-137 (1999). Also for the datasets, see http://www.diku.dk/ pisinger/testqkp.c