The Convex Hull Heuristic for Nonlinear Integer Programming Problems with Linear Constraints and Application to Quadratic 0-1 Problems

Monique Guignard and Aykut Ahlatcioglu

Department of OID, The Wharton School,
University of Pennsylvania
Philadelphia, PA 19104,USA

Abstract. The Convex Hull Heuristic (CHH) is a heuristic for mixed-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, or SD [27], and at each iteration, one solves a mixed-integer programming problem with a linear objective function and the original constraints, and a continuous problem with a nonlinear objective function and a single linear constraint. Its purpose is to produce quickly feasible and often near optimal or optimal solutions for convex and nonconvex problems. It is usually multi-start. We have tested it on a number of hard quadratic 0-1 optimization problems and present numerical results for generalized quadratic assignment problems (GQAP), cross-dock door assignment problems (CDAP), quadratic assignment problems (QAP) and quadratic knapsack problems (QKP). We compare solution quality and solution times with results from the literature, when possible.

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

1 Introduction

The Convex Hull Relaxation, or CHR, is a special case of the primal relaxation PR introduced in [12] for nonlinear mixed-integer programming problems (see also [13] and [14]). In this primal relaxation, part of the constraints are replaced by their integer convex hull. More precisely, given the nonlinear integer problem (NLIP):

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

subject to

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

$$Cx \leq d \tag{3}$$

$$x \in B \tag{4}$$

where B is the set of points x satisfying the integrality and nonnegativity requirements, the primal relaxation with respect to the constraints $Cx \leq d$ is the problem

$$\text{Minimize} \quad f(x) \tag{5}$$

subject to

$$Cx \leq d \tag{6}$$

$$x \in Co\{y \in B \mid Ay \leq b\}. \tag{7}$$

In the linear case, this is equivalent to the Lagrangean relaxation of part of the constraints (see [11]), this equivalence is often referred to a "the geometric interpretation of Lagrangean relaxation," and it is the basis for avoiding relaxations that have the "integrality property." In the nonlinear case, the primal relaxation usually is not equivalent to Lagrangean relaxation. Contesse and Guignard ([8]) showed that this relaxation can be solved by an augmented Lagrangean method, and this was successfully implemented by Ahn in his Ph.D. dissertation ([3]). Albornoz [4] and later, independently, Ahlatcioglu [1], used this relaxation keeping all the constraints: they defined the convex hull relaxation (CHR) of the problem

$$\text{Minimize} \quad \{f(x) \mid Ax \leq b, x \in B\} \tag{8}$$

as

$$\text{Minimize} \quad \{f(x) \mid x \in Co\{y \in B \mid Ay \leq b\}\}. \tag{9}$$

(CHR) is a nonlinear problem with implicit polyhedral constraints. We will say that it is not an "explicit" model, in the sense that its constraints are not explicitly known a priori. If the objective function is at least pseudo-convex, instead of having to solve the relaxation by a complex method such as an augmented Lagrangean method, this relaxation could be solved very simply by using the algorithm of Frank and Wolfe [10] or, more efficiently, a multidimensional version of it, proposed by von Hohenbalken, called Simplicial Decomposition, or SD [27]. This is closely related to an idea of Michelon and Maculan [20] for solving directly the Lagrangean dual in the linear case, that is, solving it in the primal space without resorting to Lagrangean multipliers. What is of particular interest for this paper is that this relaxation process generates integer *feasible* points for the original problem since none of the original constraints is relaxed. In the convex case this provides a bound on the optimal value of the problem, and in the nonconvex case at least feasible integer solutions. CHR was shown to be efficient for quadratic convex 0-1 problems with linear constraints in [2]. Most hard quadratic problems in the literature as well as in practice, however, tend to have nonconvex objective functions, which limits the usefulness of the method as a bounding approach, yet suggests its use as a heuristic.

In 2008, Patriksson [22] described SD in the following way: "Simplicial decomposition is a class of methods for solving continuous problems in mathematical programming with convex feasible sets. There are two main characteristics of

the methods in this class: (i) an approximation of the original problem is constructed and solved, wherein the original feasible set is replaced by a polyhedral subset thereof, that is, an inner approximation of it which is spanned by a finite set of feasible solutions; and (ii) this inner approximation is improved (that is, enlarged) by generating a vector (or, column) in the feasible set through the solution of another approximation of the original problem wherein the original cost function is approximated (often by a linear function). In case we cannot compute bounds because the objective function is nonconvex, we can still apply the general ideas of SD by (i) constructing and solving an inner approximation of the convex hull of all integer feasible solutions by using the convex hull of all integer feasible solutions already found, and (ii) generating a new feasible extreme point by optimizing an approximation of the original objective function over the original constraint set, for instance, using its first order Taylor expansion, which amounts to solving a linear integer problem over the original constraints. We call this approach the convex hull heuristic, or CHH.

In case SD is applied to a nonlinear integer optimization problem with linear constraints, the convex hull of all extreme feasible solutions, initially unknown, but constant since we always keep all the constraints, is constructed progressively as more and more of its extreme points are generated by the algorithm. Each iteration of CHH requires solving one continuous nonlinear programming (NLP) problem with a single linear constraint and a number of variables that increases through the iterations, and one mixed-integer programming (MIP) problem with all original constraints and a linear objective function that changes with the iterations. Even for problems with a nonconvex objective function, SD will generate one new integer feasible point at each iteration, probably more if one uses a Solution-Pool-Generating-Software (or SPGS) to solve the subproblems. One big advantage of an SPGS, such as the solution pool of CPLEX, for instance, is that one can make it deliver a variety of integer feasible solutions encountered during the Branch-and-Bound phase, for a given run, in addition to the optimal solution. One option is to produce Branch-and-Bound incumbents whose original objective function value is better than the best found so far. If one did not use an SPGS, one might never discover the existence of feasible points that are better relative to the original objective function, but are suboptimal for the SD problem.

In our implementation, SD is used multiple times starting from a different point. For programming and practical reasons, it is convenient to start from an integer feasible solution, which will be considered part of the family of extreme points for that SD run. This of course already requires solving an initial linear integer optimization problem without having much information about the feasible region. After testing different possibilities, we found that if one provides the optimizer with a variety of linear objective functions, it will return a variety of integer feasible solutions. Since the initial point greatly influences the sequence of points generated in a particular SD run, and since with a nonconvex nonlinear problem, the optimal solution could end up being at any feasible corner of the convex hull, we use a set of objective functions that we maximize in one SD run

and minimize in the next SD run, to start exploring the feasible corner points all around the convex hull of integer feasible points. Our objective function patterns could for instance be of the form [0, 1, 0, 1, ...], [1, 0, 1, ...], or similar. We typically use 8 patterns first for min and then for max, so 16 SD runs per instance. But we have no strict rules about how to choose these initial linear functions, and it is basically up to the user to find the kind of patterns for linear objective functions that work well with her problem type.

After describing the general CHH algorithm, we will concentrate on difficult pure 0-1 problems from the literature, with a nonlinear (in fact, quadratic as this type is more easily available in the literature) objective function, a priori nonconvex, and linear constraints. Our versions of CHH are multi-start, heuristic, with the number of repeats up to the user (we used 16), but for particularly well-behaved problem types, such as quadratic 0-1 knapsack problems, it is single start. SD always terminates after a finite number of iterations, therefore so does CHH. Since the folklore had it that SD takes a large number of iterations to converge, and therefore needs to store a large number of integer or 0-1 extreme points, we started coding a version of SD called Restricted Simplicial Decomposition (see in particular [16]) where one would start removing the weakest extreme points every (in our codes) 100 iterations. In fact, though, we never needed more than at most 30 to 40 iterations per SD run, and often much fewer, so we could have coded the standard version of [27].

In Section 2, we describe the algorithm. In Subsection 3.1, we display the SD algorithm for a more general 2-dimensional nonlinear nonconvex problem where the polyhedron represents the convex hull of the integer feasible solutions. In Subsection 3.2, we describe our implementation of CHH. Section 4 presents a sample of results from our extensive computational testing for a variety of 0-1 quadratic nonconvex problem types with linear constraints. Finally, in Section 5, we discuss possible extensions and conclusions.

The following abbreviations :
CHR Convex Hull Relaxation
CHH convex Hull Heuristic
MINLP Mixed-Integer Non Linear Programming
MIP Mixed-Integer Programming
NLIP Non Linear Integer Programming
NLP Nonlinear Programming
SD Simplicial Decomposition
SDP Semi-Definite Programming
SPGS Solution Pool Generating Software
are introduced the first time they are encountered, and will be used thereafter throughout the paper.

2 The Convex Hull Heuristic Method

In [1], a relaxation method called Convex Hull Relaxation (CHR) was introduced for computing bounds tighter than the continuous bound, for convex mixed

integer nonlinear programming problems (or MINLPs) 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.

Given problem (NLIP):

$$\text{Minimize} \quad f(x) \tag{10}$$

subject to

$$Ax \leq b \tag{11}$$

$$x \in B \tag{12}$$

where B is the set of vectors x satisfying the integrality requirements, and A and b are 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 integer feasible solutions of (NLIP):

$$H = Co\{x | Ax \leq b, x \in B\}. \tag{13}$$

This set will play an important role in what follows. The goal is to find a vertex of H with the best objective function value. We will use simplicial decomposition (SD, [27]) multiple times within a run of CHH, with a different starting point each time. Let us describe first what happens during one of these SD steps.

We defined earlier the convex hull relaxation CHR of (NLIP) as the problem of minimizing $f(x)$ over the convex hull H of the integer feasible solutions of (NLIP). Let us now describe one iteration of SD on problem CHR.

We define the Convex Hull Subproblem (CHS) at the k-th iteration of SD at that of minimizing over H the original objective function $f(x)$ linearized at a feasible point $x(k)$ of (CHR), in other words (CHS) is the problem of minimizing $L(x)$

$$L(x) = f(x(k)) + \bigtriangledown f(x(k)) \cdot (x - x(k)) \tag{14}$$

over x in H.

Note that (CHS) is a linear program, therefore it is equivalent to problem (IPS)

$$\text{Minimize} \quad L(x) \tag{15}$$

subject to

$$Ax \leq b \tag{16}$$

$$x \in B \tag{17}$$

that is, the "integer programming subproblem", which is (CHS) restricted to its integer feasible solutions.

Indeed, let $y(k)$ be an optimal solution of (CHS), it depends on $x(k)$ because the linearized function depends on the point of linearization $x(k)$. This objective function being linear, this is equivalent to solving (IPS), which is a well-defined integer linear program, in the sense that its constraints are known explicitly. To summarize, at each iteration, one solves a linear integer program with a different linearized function $L(x)$, while the constraint set remains the same.

The solution to (IPS), $y(k)$, is an extreme point of the convex hull, unless $x(k)$ is already 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 problem over the convex hull of $x(0)$, $y(1)$, $y(2)$, ... Whenever 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 $x(0)$, $y(1)$, $y(2)$, ..., 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 $x(0)$, $y(1)$, $y(2)$, ..., to contain a 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. $f(x(p))$ 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, $y(1)$, ..., $y(p\text{-}1)$, ..., and we can compute $f(y(1))$, $f(y(2))$, ..., $f(y(p\text{-}1))$,..., 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, $f(x(p))$. In the nonconvex case, we call the algorithm CHH for Convex Hull Heuristic, and while it cannot produce a valid bound, it always produces a family of 0-1 feasible solutions.

3 The CHH heuristic is based on Simplicial Decomposition

CHH is a matheuristic based on SD that is applied multiple times to obtain feasible solutions for a nonlinear integer programming problem with linear constraints. In our case SD is started each time from a different feasible point. These SD steps are independent and can actually be run sequentially or in parallel.

We first display a full SD run for a general, not 0-1, two-dimensional example. We then discuss our implementation of CHH for 0-1 optimization problems with a nonconvex objective function and linear constraints.

3.1 Displaying CHH on a figure

The picture is meant to give a feeling for what the iterations of SD, or equivalently, of one of the runs within CHH, would look like. Since a two-dimensional picture representing the unit square would not allow us to show many iterations, we chose to show the moves produced by SD (or CHH) on a polyhedron with an arbitrary shape and a somewhat larger number of vertices. This polyhedron

is the convex hull of its extreme points, which would be feasible 0-1 extreme points in CHH. These iterations lead to at least a local minimum solution for the nonconvex problem. The algorithm goes as follows. Let $x(0)$ be the initial point of linearization. Solve the linear integer program (IPS) and get $y(1)$. Solve the original NLP problem over the convex hull of $x(0)$ and $y(1)$. The new point of linearization $x(1)$ is obtained from minimizing over the line segment $[x(0),y(1)]$. Solve the new linear IP 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 $y(2)$. If the NLIP problem in this example had a convex function (or a concave function in a max problem), CHR would produce 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, $y(1), \ldots, y(p\text{-}1)$. If the problem is not convex, then a valid bound cannot be guaranteed, but one still has a heuristic for generating feasible solutions $y(1), \ldots, y(p\text{-}1)$, and the best solution found y^* is kept.

3.2 Implementation of The Convex Hull Heuristic

We coded the algorithm in GAMS, and for coding reasons, we have two types of codes, for models with single indexed or with double indexed variables.

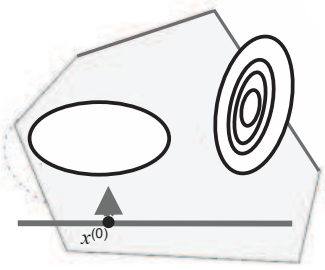
The implementation of CHR requires a starting strategy, doing restarts if necessary from different initial points of linearization, keeping a balance between optimality and runtime, and using a solution pool.

The initial iteration of SD as applied to 0-1 nonlinear problems with linear constraints requires a first point of linearization. After testing extensively different approaches, it became clear that choosing directly a good initial point of linearization was very difficult. We then somehow forced the system to generate a variety of starting points by solving LP's over the linear constraints, each time with a different objective function. Changing the linear objective function each time, that is, effectively, tilting its gradient, will be an attempt to cover a wide area of the feasible region. For even more variety, we change the orientation of each objective function direction by solving the same problem once as a maximization and once in the opposite direction as a minimization problem. This gives a variety of initial points of linearization, and we use the same 8 choices (times 2, once for minimization and once for maximization) for all multi-start runs.

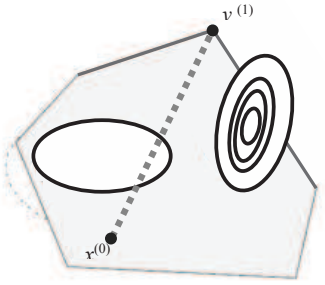
Another option in our code might be added concerning running times of the linear integer subproblems. Without imposing a time limit, within a run, it might happen that one or more of the linear MIP's takes many times as long as the average of the others. When looking for a valid lower bound, one should let these problems run to optimality, but in a nonconvex heuristic mode, one might impose a limit for each MIP run. The times reported in the paper do not use that option.

We did make use of the solution pool of our SPGS to look at the incumbents of each Branch-and-Bound (BB) run. We made it compute their quadratic

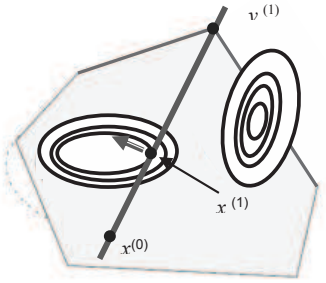
objective function value at basically no additional cost, and we kept the best overall solution. For most instances, the best feasible solution found came from a solution pool. Finally, contrary to the general feeling concerning SD, we were very surprised to see that the number of SD iterations never exceeded 100, and most of the time was in the 20-40 range. So even though we had implemented RSD (Restricted Simplicial Decomposition), [16], 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.



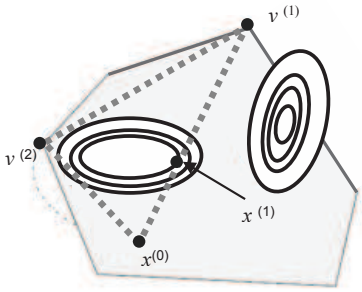
$x^{(0)}$ is the initial linearization point.



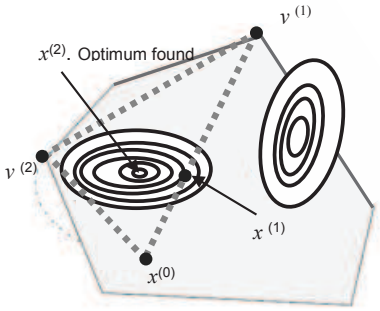
Solve the linear integer program (IPS) and get $y^{(1)}$.



Solve the original NLP over Co$\{x^{(0)}, y^{(1)}\}$, and get $x^{(1)}$, the new linearization point



Solve the new linear IP and get $y^{(2)}$.



Solve the original NLIP over Co$\{x^{(0)}, y^{(1)}, y^{(2)}\}$, and get $x^{(2)}$, which is at least a local optimal solution.

Fig. 1. Illustration of iterations

4 Computational results

We tested the CHH heuristic on a number of difficult quadratic pure 0-1 problems with linear constraints: GQAP (generalized quadratic assignment problem), CDAP (cross-dock door assignment problem), QAP (quadratic assignment problem), and QKP (quadratic 0-1 knapsack problem). While the CHH heuristic is generic and could be applied to more general problems, we decided to start implementing it on pure 0-1 quadratic problems that are of interest to a number of researchers, especially for computing bounds, with well known approaches such as the RLT method of Sherali and Adams [26], interior point methods, for instance [25], or SDP methods as used in [23], just to cite a few. We will present the results separately for each problem type. An important point is that some well known heuristics are specific to the problem structure, while CHH is essentially a general purpose heuristic, which requires little effort to convert to another problem type. All implementations were done in GAMS, on either a Thinkpad laptop T431S, core I7 running Windows 7, 8GB of memory, and a department server with similar characteristics. We used CONOPT as the NLP solver and CPLEX as the MIP solver under GAMS, and activated the solution pool feature of CPLEX. As this is the main computational burden, we recorded the time needed either separately for solving MIP problems and for solving NLP problems, or for the sum of the two. As mentioned earlier, running the heuristic on a parallel machine would allow multiple runs of SD in parallel, and with multiple servers, one could use much more parallel runs of SD than we did in our experiments where we limited them to 16.

4.1 GQAP

The generalized quadratic assignment problem is the quadratic equivalent of the generalized assignment problem. It was introduced by Lee and Ma [17]. We are using data from Cordeau et al. [9]. The instances range from 180 to 600 0-1 variables. Times are in seconds on an older department machine. Total times were all less than 40 seconds, except for three instances for which cplex runs took much longer to reach MIP optimal solutions. Only for one instance, the 30-20-95, did we not find an optimal solution, but the percentage gap was still only 0.28

Table 1. GQAP instances. instance. nbvar = number of 0-1 variables. prcnt = optimality percentage error. TM = Time for MIP. TN = Time for NLP. TT = total time in seconds. GAMS 23.7.3.

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

4.2 CDAP

The cross-dock door assignment problem (CDAP, see for instance [7] and [15]) is the problem of assigning loaded incoming trucks to unloading doors and empty outbound 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 transfer times and labor costs. The assignment problem can be solved before the beginning of a shift, taking into account the trucks that are already there, and it may need to be solved again quickly whenever trucks arrive or 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 was one possibility, but did lead to numerical difficulties, like division by 0. Instead we used the subset handling capability of GAMS to consider only the sparse subset of feasible assignments from the start. This code does not use the general 2-indexed variable code structure, but relies heavily on the use of subsets of feasible index pairs for the decision variables. Occasionally, if permitted, this would allow the user to include such possibilities as assigning a empty truck to a door on the unloading side if most of the items in an inbound truck go to the same destination and could therefore be transferred cheaply to the outbound truck if assigned to a neighboring door also on the incoming side.

An early presentation [15] mentioned a preliminary implementation of CHH for the CDAP over small instances. We use here a realistic dataset generated by D. Cardoso da Silva [7]. 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 outbound, 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 sides. This yields two series of tests, that differ slightly in the way each iteration starts. All these instances were run on the same fast workstation for both methods. Since optimal values were not known for these large instances, we compared our best values with those of Cardoso. For smaller problems (with up to 400 0-1 variables), Cardoso's heuristic and CHH are basically indistinguishable in solution quality and running time. For the large problems, on the average, CHH solutions are 0.5 percent worse than the better value of Cardoso's two implementations. The table only mentions the time needed by Cardoso's better run for each instance, so that his actual runtimes are actually roughly double those listed in the table. CHH solution times were much shorter, as seen in the following table. This may be important if trucks are waiting to be assigned. A - sign in the prcnt column means that for that instance, the CHH solution value was cheaper than the better of the two Cardoso values by that percentage gap, and conversely for a + sign.

Table 2. CDAP instances. instance . nbvar = 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. GAMS 24.4.5.

instance	nbvar	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	+0.26
SetB-50x10S5	1000	191773	4039	192114	178	+0.18
SetB-50x10S15	1000	188006	3363	187753	164	-0.13
SetB-50x10S30	1000	183961	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. It is also one of the most widely studied (see for instance [5]). The state of the art for the Nugent data set is still 30, or 900 0-1 variables. Many instances are now available with larger sizes.

The instances below are taken from QAPLib [24]. They are divided into "smaller" and "larger" sizes. The "smaller" instances range in size from 12 to 29, and the "larger" ones 30 and above. Remember that a QAP of size n has nxn 0-1 variables, e.g., a problem of size 40 has 1600 binary variables. The two tables of results present only a subset of the large number of problems available in QAPLIB. We solved all instances for which we could obtain the data in GAMS format. Times were negligible for the smaller instances because linear assignment problems take no time for such sizes. For the larger sizes, for most instances times are still very small. All runs were done on a department server running GAMS 26.1.0, using CONOPT and CPLEX.

Table 3. smaller QAP instances . instance . size = number of rows (and columns) . TT = total time in seconds . TM = Time for MIP. TN = Time for NLP. prcnt = optimality percentage error . OV = optimum . best = best found .

instance	size	TT	TM	TN	prcnt	OV	best
tai12a	12	0.81	0.74	0.07	8.4	224416	243206
nug15	15	0.71	0.58	0.12	3.3	1150	1188
nug16a	16	0.92	0.60	0.31	0.75	1610	1622
nug16b	16	0.56	0.50	0.06	0.97	1240	1253
nug17	16	0.71	0.51	0.2	1.5	1732	1758
nug18	18	0.65	0.58	0.07	1.2	1930	1954
nug20	20	0.48	0.43	0.05	1.17	2570	2600
nug21	21	0.27	0.24	0.03	1.7	2438	2480
nug22	22	0.46	0.44	0.05	2.4	3596	3684
nug24	24	0.34	0.31	0.03	4.2	3488	3634
nug25	25	0.42	0.38	0.04	0.2	3744	3752
tai25b	25	0.33	0.29	0.04	1.6	344355646	350007430
bur26a	26	0.31	0.30	0.04	0.3	5426670	5443125
bur26b	26	0.43	0.38	0.05	0.4	3817852	3832488
bur26c	26	0.32	0.29	0.03	0.1	5426795	5432612
bur26d	26	0.44	0.39	0.05	0.07	3821225	3823853
bur26e	26	0.32	0.29	0.03	0.07	5386879	5390408
bur26f	26	0.35	0.34	0.04	0.07	3782044	3784879
bur26g	26	0.26	0.24	0.03	0.6	10117172	10173352
bur26h	26	0.35	0.31	0.04	0.8	7098658	7155135
nug28	28	0.41	0.37	0.04	2.2	5166	5282

Table 4. larger QAP instances . instance . size = number of rows (and columns) .TT = total time in seconds . TM = Time for MIP. TN = Time for NLP. prcnt = optimality percentage error . OV = optimum . best = best found .

instance	size	TT	TM	TN	prcnt	OV	best
kra30a	30	0.40	0.37	0.03	3.6	88900	92070
kra30b	30	0.4	0.37	0.03	4.5	91420	95550
lipa30a	30	0.61	0.54	0.7	2.1	13178	13451
lipa30b	30	0.43	0.41	0.06	0	151426	151426
nug30	30	0.39	0.35	0.04	0.2	6124	6136
esc32d	32	0.57	0.54	0.02	35	200	270
kra32	32	0.5	0.47	0.04	2.6	88700	90970
ste36c	36	0.46	0.43	0.03	4.1	8239110	8851130
lipa40a	40	0.67	0.55	0.12	1.8	31538	32117
lipa40b	40	0.49	0.41	0.08	0	476581	476581
tho40	40	0.47	0.42	0.05	2	240516	245464
lipa50a	50	0.86	0.7	0.16	1.4	62093	62971
esc64a	64	0.83	0.81	0.02	0	116	116
esc128	128	2.88	2.86	0.02	15.6	64	74

4.4 QKP

For the quadratic 0-1 knapsack problem (see for instance [18]), we used Caprara et al.'s dataset [6] and want to thank L.Ltocart and G.Plateau [19] for making available to us the data and solution files. We ran the first instance in each density and size group, 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, first of all the origin was not feasible, and 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. While for other problems we reported separately on the time spent on MIP and on NLP subproblems, these subproblems were solved so quickly on the Thinkpad T431s that we just report that the longest "clock time" for one complete run of a dense CHH with 300 0-1 variables was approximately 15 seconds. Notice that it is our only maximization problem.

Table 5. QKP instances. dens = instance density. nb = number of variables. opt = optimal value. found = optimum found? . best = best value found. prcnt = optimality percentage error. GAMS 24.6.1.

dens	nb	opt	found	best	prcnt
100	50	114593	Y	same	0
100	100	343337	Y	same	0
100	150	571718	Y	same	0
100	200	623145	Y	same	0
100	250	3036287	Y	same	0
100	300	3539021	Y	same	0
75	50	55007	Y	same	0
75	100	354120	N	354095	0.0071
75	150	137672	Y	same	0
75	200	75588	Y	same	0
75	250	1374797	Y	same	0
75	300	924906	N	924750	0.0169
50	50	24804	Y	same	0
50	100	108439	Y	same	0
50	150	400743	Y	same	0
50	200	427428	N	427380	0.0112
50	250	1550630	Y	same	0
50	300	194556	Y	same	0
25	50	17260	Y	same	0
25	100	52998	Y	same	0
25	150	140710	N	140598	0.0796
25	200	85771	Y	same	0
25	250	704437	Y	same	0
25	300	1116454	Y	same	0

5 Extensions and conclusions

The convex hull heuristic is based on simplicial decomposition, and generates feasible solutions provided by linearized versions of the original 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 integer problem to generate a new extreme point of the integer convex hull of the integer feasible solutions (this part takes more time, except for small 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 nonlinear integer problems. So far, the number of iterations has always been 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 one solves every integer subproblem to optimality.

Computationally this paper has been concentrating on applying CHH to quadratic 0-1 problems. Future research might switch to other types of instances, with mixed-integer rather than pure 0-1 variables, and/or general nonconvex nonlinear objectives rather than quadratic.

References

1. 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 (2007–09).
2. Ahlatcioglu, A., Bussieck, M., Esen, M., Guignard, M., Jagla, J.-H., Meeraus A.: Combining QCR and CHR for convex quadratic pure 0-1 programming problems with linear constraints. Ann. Oper. Res. 199:33–49 (2012).
3. Ahn, S.: On solving some optimization problems in stochastic and integer programming with applications in finance and banking. Ph.D. dissertation, University of Pennsylvania. (1997)
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. Anstreicher, K.M., Brixius, N.W.: Solving quadratic assignment problems using convex quadratic programming relaxations. Optimization Methods and Software, 16:49–68, 2001.
6. 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
7. 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).
8. Contesse L., Guignard M.: An augmented Lagrangean relaxation for nonlinear integer programming solved by the method of multipliers, Part I, Theory and algorithms. Working Paper, OPIM Department, University of Pennsylvania. (1995, latest revision 2009).
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. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Naval Research Quarterly, 3, 95–109 (1956).
11. Geoffrion, A. M.: Lagrangean relaxation for integer programming. Mathematical Programming Studies, 2, 82–114. (1974).
12. Guignard, M.: Primal relaxations for integer programming. Invited plenary talk, VII CLAIO, Santiago, Chile (July 1994). Also Technical Report 94–02–01, University of Pennsylvania, OPIM Department (1994).
13. Guignard, M.: Lagrangean relaxation. TOP, 11(2), 151–228 (2003).
14. Guignard, M.: A new, solvable, primal relaxation for nonlinear integer programming problems with linear constraints. Optimization Online (2007).
15. 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. 1–12. Angra dos Reis, Brazil (2012).

16. Hearn D.W., Lawphongpanich S., Ventura J.A.: Finiteness in restricted simplicial decomposition. Op. Res. Letters 4, 125–130 (1985).
17. Lee, Chi-Guhn, Ma, Zhong : The Generalized Quadratic Assignment Problem. Research Report, Department of Mechanical and Industrial Engineering, University of Toronto. (2003).
18. Ltocart L., Nagih A., Plateau G.: Reoptimization in Lagrangian methods for the 01 quadratic knapsack problem. Computers and O.R., Special Issue on Knapsack Problems and Applications, 39, 12–18 (2012)
19. Ltocart L., Plateau G.: Private communication. (2015–16).
20. Michelon, P., Maculan, N.: Solving the Lagrangean dual problem in integer programming. Report 822, Departement d'Informatique et de Recherche Operationnelle, University of Montreal (1992).
21. Park J., Boyd S.: A semidefinite programming method for integer convex quadratic minimization, Optimization Letters, 12, 499-518 (2018).
22. Patriksson, M.: Simplicial Decomposition Algorithms. In: Floudas C., Pardalos P. (eds) Encyclopedia of Optimization, Springer, Boston, MA. (2008)
23. Poljak, S., Rendl, F., Wolkowicz, H.: A recipe for best semidefinite relaxation for 0-1 quadratic programming. J. Global Optimization, 7, 51–73 (1995).
24. http://anjos.mgi.polymtl.ca/qaplib/inst.html
25. Resende, M.G.C., Ramakrishnan, K.G., and Drezner. Z.: Computing Lower Bounds for the Quadratic Assignment Problem with an Interior Point Algorithm for Linear Programming. Operations Research, 43, 781–791 (1995).
26. Sherali, H.D., Adams, W.P.: A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM J. Discrete Math. 3, 411–430 (1990).
27. von Hohenbalken, B.: Simplicial decomposition in nonlinear programming algorithms. Mathematical Programming 13, 49–68 (1977)