DisjunctiveProgramming.jl: Generalized Disjunctive Programming Models and Algorithms for JuMP

Hector D. Perez*, Shivank Joshi, Ignacio E. Grossmann

Carnegie Mellon University, Pittsburgh, PA, USA
*hdperez@cmu.edu

ABSTRACT

We present a Julia package, *DisjunctiveProgramming.jl*, that extends the functionality in *JuMP.jl* to allow modeling problems via logical propositions and disjunctive constraints [7]. Logical propositions are converted into algebraic expressions by converting the Boolean expressions to Conjunctive Normal Form and then to algebraic inequalities. The package allows the user to specify the technique to reformulate the disjunctions (Big-M or Hull reformulations) into mixed-integer constraints. This allows generating models that can be solved by the various MIP solvers supported by JuMP [8]. The package supports reformulations for disjunctions containing linear, quadratic, and nonlinear constraints.

Keywords

Julia, Optimization, Mathematical programming,
Generalized disjunctive programming

1. Introduction

The modeling of systems with discrete and continuous decisions is commonly done in algebraic form with mixed-integer programming (MIP) models. This is the standard approach for setting up problems that are linear or nonlinear in their continuous variables.

A more systematic approach to modeling such systems is to use Generalized Disjunctive Programming (GDP) TODO (Chen & Grossmann, 2019; Grossmann & Trespalacios, 2013), which generalizes the Disjunctive Programming paradigm proposed by Balas [1]. GDP enables the modeling of systems from a logic-based level of abstraction that captures the fundamental rules governing such systems via algebraic constraints and logic. This formulation is useful for expressing problems in an intuitive way that matches their logical structure, without needing to translate the logical statements into mathematical form.

The models obtained via GDP can then be reformulated into the pure algebraic form best suited for the application of interest. These models group related constraints into disjuncts to keep GDP models highly organized. It is also often possible to exploit the explicit logical structure of a GDP model to provide tighter relaxations than corresponding MIP models, which may improve convergence speed and robustness for solutions via advanced solution algorithms TODO (Chen & Grossmann, 2019).

There is currently high volume of ongoing research using GDP to formulate models for a variety of applications. Due to the combinatorial nature of system design problems, the GDP paradigm is commonly applied to the optimization of design synthesis for complex sequential processes and networks [17, 22]. Other work using the GDP approach is in the area of planning and optimal control for energy systems [6, 15]. There is also ongoing research to further develop the generalized disjunctive programming approach, handling uncertainties in the chemical synthesis area [5]. These numerous applications of GDP illustrate the benefit of a robust package to abstract away some of the overhead associated with developing and testing GDP models to accelerate research.

This paper provides background information on the GDP paradigm and a review of techniques for reformulating and solving models. It then presents the package DisjunctiveProgramming.jl as an extension to `JuMP.jl` for creating models for optimization that follow this modeling paradigm. A case study demonstrates the use of the package for optimization over a chemical process superstructure.

2. Generalized Disjunctive Programming

The GDP form of modeling is an abstraction that uses both algebraic and logical constraints to capture the fundamental rules governing a system. The models obtained via GDP can then be reformulated into the mixed-integer algebraic form best suited for the application of interest. The two main reformulation strategies are the Big-M reformulation [19, 20] and the Hull reformulation [16], the latter of which yields tighter models at the expense of larger model sizes [12].

2.1 Formulation of Problem

The notation for a general decision process with continuous and discrete variables written in the Mixed-Integer Nonlinear Programming (MINLP) problem is as follows:

$$\begin{aligned} &\min f(x,y) \\ &\text{s.t. } g(x,y) \leq 0 \\ &\quad h(x,y) = 0 \\ &\quad x \in X \subset \mathbb{R}^n \\ &\quad y \in Y \subset \mathbb{Z}^m \end{aligned}$$

In the above notation, x is the set of continuous variables, y is the set of discrete variables, and f is the objective function to be minimized. g and h are functions representing the inequality and equality constraints on the problem, respectively [4].

The general notation for a disjunction in a GDP problem is

$$
\begin{aligned}
&\min f(x,y)\\
&\text{s.t. } g(x,y) \le 0\\
&\quad \bigvee_{i\in D_k} \begin{bmatrix} \Upsilon_{i,k} \\ h_{i,k}(x,z) \le 0 \end{bmatrix},\ \forall k \in K\\
&\quad \Omega(\Upsilon) = True\\
&\quad \Upsilon_{i,k} \in \{0,1\}\\
&\quad x \in X \subset \mathbb{R}^n\\
&\quad y \in Y \subset \mathbb{Z}^m
\end{aligned}
$$

Here, there are k disjunctions with i terms each, and constraint functions h, are applied only if the respective disjunct's term's indicator variable, Υ, is denoted as being active [4]. The set of logical propositions, Ω, describe logical relationships between the selections of indicator variables.

In the case of a linear objective and set of constraints, the GDP model can be written as

$$
\begin{aligned}
&\min c^T x\\
&\text{s.t. } Ax \le b\\
&\quad \bigvee_{i\in D_k} \begin{bmatrix} z_{i,k} \\ B_{i,k}x \le d_{i,k} \end{bmatrix},\ k \in K\\
&\quad \Omega(z) = True\\
&\quad z_{i,k} \in \{0,1\}
\end{aligned}
$$

In this notation, x groups discrete and continuous variables.

2.2 Linear GDP reformulation example

The simplest example of a linear GDP system is given in (1) - (3), where y_i is a Boolean indicator variable that enforces the constraints in the disjunct ($Ax \le b$ or $Cx \le d$) when $true$.

$$\begin{bmatrix} y_1 \\ Ax \le b \end{bmatrix} \vee \begin{bmatrix} y_2 \\ Cx \le d \end{bmatrix} \tag{1}$$

$$0 \le x \le U \tag{2}$$

$$y_i \in \{true, false\} \quad \forall i \in \{1,2\} \tag{3}$$

For the purpose of visualization of the following reformulations of the feasible space, the simple example of a feasible solution space in 1 will be used without loss of generality.

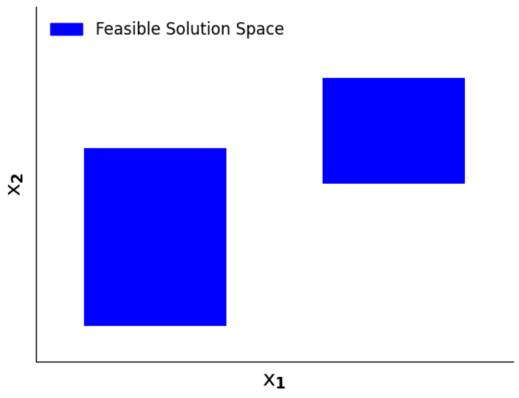

Fig. 1. Feasible solution space for example disjunction

2.2.1 *Big-M Reformulation.*

The Big-M reformulation for this problem is given by (2), (4) - (7).

$$Ax \le b + M\cdot(1-y_1) \tag{4}$$

$$Cx \le d + M\cdot(1-y_2) \tag{5}$$

$$y_1 + y_2 = 1 \tag{6}$$

$$y_i \in \{0,1\} \quad \forall i \in \{1,2\} \tag{7}$$

This reformulation can be visualized by the region encapsulated by the dashed line in 2. Note that the relaxed region is not as tight as possible around the feasible solution space. The choice of large 'M' value determines the tightness of this relaxation, and the minimal value of 'M' for the optimal relaxation can be found through interval arithmetic.

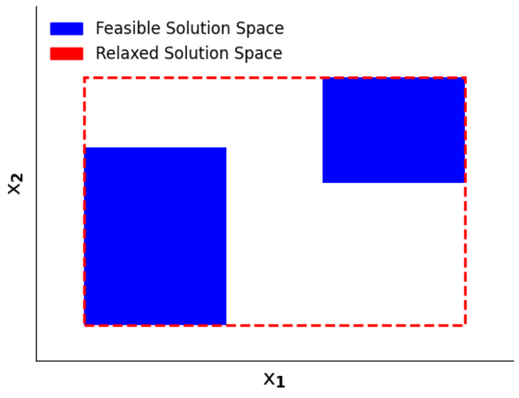

Fig. 2. Relaxed solution space using Big-M Reformulation

2.2.2 Convex Hull Reformulation. The Hull reformulation is given by (6) - (11), which requires lifting the model to a higher-dimensional space. When projected to the original space, the continuous relaxation of the model is tighter than its Big-M equivalent (Grossmann & Trespalacios, 2013).

$$Ax_1 \leq by_1 \tag{8}$$

$$Cx_2 \leq dy_2 \tag{9}$$

$$x = x_1 + x_2 \tag{10}$$

$$0 \leq x_i \leq Uy_i \quad \forall i \in \{1, 2\} \tag{11}$$

This reformulation can be visualized by the region encapsulated by the dashed line in 3. Note that this reformulation provides a tighter relaxation than the Big-M reformulation. Also note that describing the geometry of this relaxation is more complex than the Big-M relaxation, and this increases the relative size of the model.

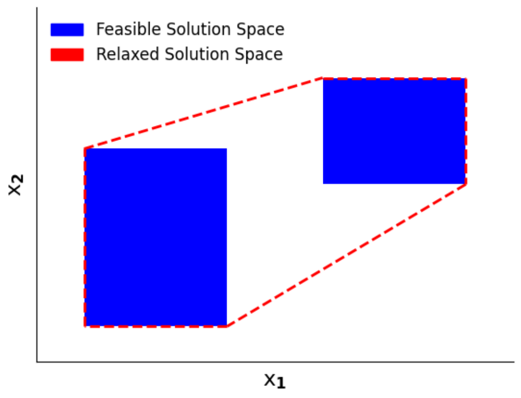

Fig. 3. Relaxed solution space using Hull Reformulation

2.3 Model Logical Constraint Reformulation

2.3.1 Propositional Logic. The logical propositions within the set of decision selection relationships, Ω, must be converted into conjunctive normal form (CNF) prior to solving a GDP model. This means that each phrase within the let of propositions must be formulated into a conjunction of disjunctions. This process can be accomplished by following the simplifying rules of propositional logic as follow:

For boolean variables A, B, and C,

$$\begin{aligned} A \leftrightarrow B &\cong (A \rightarrow B) \wedge (B \rightarrow A) \\ A \rightarrow B &\cong \neg A \vee B \\ \neg(A \vee B) &\cong \neg A \wedge \neg B \\ \neg(A \wedge B) &\cong \neg A \vee \neg B \\ (A \wedge B) \vee C &\cong (A \vee C) \wedge (B \vee C) \end{aligned}$$

More complex approaches to CNF conversion involve preserving phrase satisfiability rather than phrase equivalence, which prevents exponential size increase in phrases and yields logically consistent results. [14].

2.3.2 Constraint Programming. The requirements for constraints within Ω are also reformulated as follows:

$$\begin{aligned} \text{exactly}(n, Y) &\rightarrow n = \sum_i Y_i \\ \text{atleast}(n, Y) &\rightarrow n \leq \sum_i Y_i \\ \text{atmost}(n, Y) &\rightarrow n \geq \sum_i Y_i \end{aligned}$$

for n boolean variables Y.

2.4 Solution Techniques

2.4.1 Disjunctive branch and bound. The disjunctive branch and bound method closely mirrors the standard branch and bound approach for the solution of mixed-integer programming problems [12]. This algorithm iterates to converge to a solution by first solving a relaxed version of the problem and assigning a value, λ, to each term in each disjunction corresponding to how close the disjunct's constraints are to being satisfied. Then, the disjunct closest to being satisfied is fixed and each branch stemming from this problem is added to a queue. As long as there are problems on the queue, this algorithm iterates. If a solution is found, the problem is solved. If there is no solution found and the queue is empty, the problem is infeasible.

2.4.2 Logic-based outer approximation. Logic-based outer approximation is another algorithm which mirrors a standard technique for solving mixed integer programming problems [9]. This approach starts by identifying sub-problems of the GDP model to solve in order to encompass each set of disjunctions. Then, each subproblem is solved to find optimal solutions which serve as upper bounds. These solutions are used to locally linearize the objective and constraints of the original problem and solve the resulting problem to find a lower bound. If the lower and upper bound solutions have not converged, fix the boolean variables from the previous solution and solve this problem to find a potentially tighter upper bound solution. If the bounds have still not converged, repeat the process, linearizing the problem and resolving.

2.4.3 Hybrid cutting planes. The cutting planes method is an algorithm for tightening the relaxed solution space of a problem reformulated with Big-M before solving it by adding additional constraints which remove parts of the relaxed space that are disjoint from the actual feasible solution space. These 'cuts' to the relaxed solution space are derived from the tighter, convex-hull relaxation for the problem. This algorithm provides a middle-ground for the tradeoff between the complexity and corresponding computational expense of the Convex Hull reformulation with the less tight Big-M reformulation. [21].

3. DisjunctiveProgramming.jl

3.1 Features

3.2 Example

To illustrate the syntax in *DisjunctiveProgramming.jl* (Version 0.3.2), consider the simple superstructure optimization problem for the chemical process given in Figure 4. The GDP model seeks to maximize the product flow (F_7), while discounting for reactor (CR) and separator (CS) installation costs as given in (12), subject to the nested disjunction in (13) and the global mass balances in (14) - (15). The system variables are the flows on each stream i (F_i) and the installation costs, with their respective bounds given in (16) - (18). The fixed cost and process yield parameters are given by γ and β, respectively.

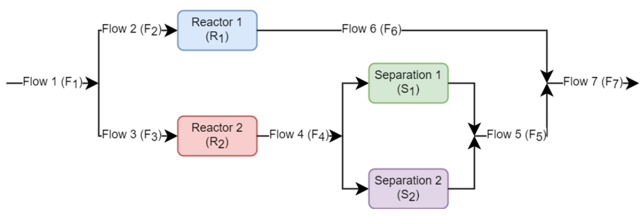

Fig. 4. Illustrative superstructure optimization problem

$$\max F_7 - CR - CS \tag{12}$$

$$\begin{bmatrix} Y_{R1} \\ F_6 = \beta_{R1} F_2 \\ F_3 = 0 \\ F_4 = 0 \\ F_5 = 0 \\ C_R = \gamma_{R1} \\ CS = 0 \end{bmatrix} \vee \begin{bmatrix} Y_{R2} \\ F_2 = 0 \\ F_6 = 0 \\ F_4 = \beta_{R2} F_3 \\ C_R = \gamma_{R2} \\ \begin{bmatrix} Y_{S1} \\ F5 = \beta_{S1} F_4 \\ C_S = \gamma_{S1} \end{bmatrix} \vee \begin{bmatrix} Y_{S2} \\ F5 = \beta_{S2} F_4 \\ C_S = \gamma_{S2} \end{bmatrix} \end{bmatrix} \tag{13}$$

$$F_1 = F_2 + F_3 \tag{14}$$

$$F_7 = F_5 + F_6 \tag{15}$$

$$0 \leq F_i \leq 10 \quad \forall i \in \{1, ..., 7\} \tag{16}$$

$$0 \leq CS \leq CS^{max} \tag{17}$$

$$CR^{min} \leq CR \leq CR^{max} \tag{18}$$

When using the Big-M reformulation in *DisjunctiveProgramming.jl*, the user can specify the Big-M value to be used, which can either be general to the disjunction or specific to each constraint expression in the disjunction. Alternatively, the user can allow the package to determine the tightest Big-M value based on the variable bounds and constraint functions using interval arithmetic. When the Convex-Hull reformulation is selected, the perspective function approximation from Furman, et al. (2020) is used for nonlinear constraints with a specified tolerance value, ϵ [10]. This is done by relying on manipulation of symbolic expressions via Symbolics.jl [11].

The above system can be modeled, reformulated via the Big-M reformulation, and optimized as follows:

```julia
using DisjunctiveProgramming, JuMP, HiGHS

# create model
m = JuMP.Model(HiGHS.Optimizer)
# add variables to model
@variable(m, 0 <= F[i = 1:7] <= 10)
@variable(m, 0 <= CS <= CSmax)
@variable(m, CRmin <= CR <= CRmax)

# add constraints to model
@constraints(m,
    begin
        F[1] == F[2] + F[3]
        F[7] == F[5] + F[6]
    end)

# define inner disjunction
@disjunction(m,
    begin
        F[5] == β[:S1]*F[4]
        CS == γ[:S1]
    end,
    begin
        F[5] == β[:S2]*F[4]
        CS == γ[:S2]
    end,
    reformulation = :big_m, # reformulation type
    name = :YS) # symbol for indicator variable

# define first outer disjunction
R1_con = @constraints(m,
    begin
        F[6] == β[:R1]*F[2]
        [i = 3:5], F[i] == 0
        CR == γ[:R1]
        CS == 0
    end)

# define second outer disjunction
R2_con = @constraints(m,
    begin
        F[6] == β[:R2]*F[3]
        CR == γ[:R2]
    end)

# add nested disjunction to model
add_disjunction!(m,
    R1_con,
    # general constraints in R2 disjunction
    (R2_con,
     m.ext[:YS]), #reformulated inner disjunction
    reformulation = :big_m, # reformulation type
    name = :YR) # symbol for indicator variable

# add logical constraints
#     1) choose only one reactor at once
#     2) select separation system only if
#        using reactor 2
choose!(1, YR[1], YR[2]; mode = :exactly)
choose!(YR[2], YS[1], YS[2]; mode = :exactly)

# add objective function and optimize
@objective(m, Max, F[7] - CS - CR)
optimize!(m)
```

4. Future Work

The following is a selection of potential future steps for work on this package:

- Upgrade package to new Non-Linear API in JuMP

- Leverage JuMP extension infrastructure: extend `JuMP.build_constraint` and `JuMP.add_constraint`
- Extend JuMP further:
  - Create a GDP Model that can be reformulated at solve time
  - Allow indexing notation when creating disjunctions
  - Leverage:
    - New boolean variable type
    - New disjunction constraint type
    - Ext dictionary
  - Use `JuMP.set_optimize_hook`
- Add additional GDP solution strategies including:
  - Disjunctive Branch-and-Bound
  - Logic-based Outer Approximation
  - Hybrid Cutting Plane methods
  - Basic Steps

5. Related Work

The popular Python package `Pyomo` is widely used for optimization development and includes an extension for generalized disjunctive programming [3, 13]. `GAMS` is a widely used optimization solver with support for GDP under the solver `GAMS EMP` [2]. Research is also being conducted to integrate modern process simulation technology, such as `Aspen`, with the GDP paradigm [18].

6. Conclusion

`DisjunctiveProgramming.jl` general recap
Usage and potential applications
Comparison with other work

7. References

[1] Egon Balas. *Disjunctive programming*. Springer, 2018.

[2] Michael R. Bussieck and Alex Meeraus. *General Algebraic Modeling System (GAMS)*, pages 137–157. Springer US, Boston, MA, 2004. doi:10.1007/978-1-4613-0215-5_8.

[3] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, third edition, 2021.

[4] Qi Chen and Ignacio Grossmann. Modern modeling paradigms using generalized disjunctive programming. *Processes*, 7(11):839, 2019. doi:10.3390/pr7110839.

[5] Ying Chen, Yixin Ye, Zhihong Yuan, Ignacio E. Grossmann, and Bingzhen Chen. Integrating stochastic programming and reliability in the optimal synthesis of chemical processes. *Computers & Chemical Engineering*, 157:107616, 2022. doi:https://doi.org/10.1016/j.compchemeng.2021.107616.

[6] Seolhee Cho and Ignacio E. Grossmann. An optimization model for expansion planning of reliable power generation systems. In Ludovic Montastruc and Stephane Negny, editors, *32nd European Symposium on Computer Aided Process Engineering*, volume 51 of *Computer Aided Chemical Engineering*, pages 841–846. Elsevier, 2022. doi:https://doi.org/10.1016/B978-0-323-95879-0.50141-7.

[7] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017. doi:10.1137/15m1020575. https://doi.org/10.1137/15M1020575.

[8] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017. doi:10.1137/15M1020575.

[9] Ignacio E. Grossmann. *Logic-based outer approximationLogic-Based Outer Approximation*, pages 1928–1931. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-74759-0_348.

[10] Kevin C. Furman, Nicolas W. Sawaya, and Ignacio E. Grossmann. A computationally useful algebraic representation of nonlinear disjunctive convex sets using the perspective function. *Computational Optimization and Applications*, 76(2):589–614, 2020. doi:10.1007/s10589-020-00176-0.

[11] Shashi Gowda, Yingbo Ma, Alessandro Cheli, Maja Gwóźzdź, Viral B. Shah, Alan Edelman, and Christopher Rackauckas. High-performance symbolic-numerics via multiple dispatch. *ACM Commun. Comput. Algebra*, 55(3):92–96, jan 2022. doi:10.1145/3511528.3511535.

[12] Ignacio E. Grossmann and Sangbum Lee. Generalized convex disjunctive programming: Nonlinear convex hull relaxation. *Computational Optimization and Applications*, 26(1):83–100, 2003. doi:10.1023/a:1025154322278.

[13] William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[14] Paul Jackson and Daniel Sheridan. Clause form conversions for boolean circuits. *Theory and Applications of Satisfiability Testing*, page 183–198, 2005. doi:10.1007/11527695_15.

[15] Donghun Kim. Generalized disjunctive programming-based, mixed integer linear mpc formulation for optimal operation of a district energy system for pv self-consumption and grid decarbonization: Field implementation. *International High Performance Buildings Conference*, 2022.

[16] Sangbum Lee and Ignacio E. Grossmann. New algorithms for nonlinear generalized disjunctive programming. *Computers & Chemical Engineering*, 24(9):2125–2141, 2000. doi:https://doi.org/10.1016/S0098-1354(00)00581-0.

[17] Fahad Matovu, Shuhaimi Mahadzir, Rasel Ahmed, and Nor Erniza Mohammad Rozali. Synthesis and optimization of multilevel refrigeration systems using generalized disjunctive programming. *Computers & Chemical Engineering*, 163:107856, 2022. doi:https://doi.org/10.1016/j.compchemeng.2022.107856.

[18] Miguel A. Navarro-Amorós, Rubén Ruiz-Femenia, and José A. Caballero. Integration of modular process simulators under the generalized disjunctive programming framework for the structural flowsheet optimization. *Computers & Chemical Engineering*, 67:13–25, 2014. doi:https://doi.org/10.1016/j.compchemeng.2014.03.014.

[19] George L. Nemhauser. *Integer and combinatorial optimization*. John Wiley and Sons, 1999.

[20] Francisco Trespalacios and Ignacio E. Grossmann. Improved big-m reformulation for generalized disjunctive programs. *Computers & Chemical Engineering*, 76:98–103, 2015. doi:https://doi.org/10.1016/j.compchemeng.2015.02.013.

[21] Francisco Trespalacios and Ignacio E. Grossmann. Cutting plane algorithm for convex generalized disjunctive programs. *INFORMS Journal on Computing*, 28(2):209–222, 2016. doi:10.1287/ijoc.2015.0669.

[22] Wenjin Zhou, Kashif Iqbal, Xiaogang Sun, Dinghui Gan, Chun Deng, José María Ponce-Ortega, and Chunmao Chen. Disjunctive programming model for the synthesis of property-based water supply network with multiple resources. *Chemical Engineering Research and Design*, 187:69–83, 2022. doi:https://doi.org/10.1016/j.cherd.2022.08.027.