GIỚI THIỆU ERX SOLVER

CÔNG CỤ THIẾT KẾ VÀ GIẢI BÀI TOÁN TỐI ƯU LP & MIP

(LINEAR & MIX-INTEGERED PROGRAMING)

- *Kiến thức được tổng hợp từ internet với sự hỗ trợ của AI – ChatGPT (mode: deep research)*
- *Python code & Excel format & Website – được thiết kế bởi Ths. Nguyễn Minh Ngọc (MSc Suppy Chain & Logistics Management)*
- *ERX Vietnam phát hành*

1. Lý thuyết tối ưu hóa lựa chọn – Khái niệm và ứng dụng thực tế

Tối ưu hóa trong Operations Research (OR) là việc tìm kiếm phương án **tốt nhất** (tối ưu) trong số các lựa chọn khả dĩ, dựa trên một mô hình toán học của bài toán thực tế. Theo TechTarget, OR là "một phương pháp phân tích để giải quyết vấn đề và ra quyết định, trong đó vấn đề được chia thành các thành phần cơ bản và giải bằng phân tích toán học theo các bước xác định" (What is Operations Research and Why is it Important?). Nói cách khác, ta xây dựng mô hình gồm **hàm mục tiêu** (đại diện cho tiêu chí cần tối ưu, ví dụ: tối đa lợi nhuận hoặc tối thiểu chi phí) và một tập các **ràng buộc** (đại diện cho giới hạn nguồn lực hoặc điều kiện phải thỏa mãn). Nhiệm vụ của tối ưu hóa là tìm **giá trị của các biến quyết định** (decision variables) sao cho hàm mục tiêu đạt cực đại hoặc cực tiểu, đồng thời thỏa mãn tất cả ràng buộc.

Ứng dụng thực tế của OR rất rộng rãi, trải khắp nhiều lĩnh vực. OR khởi nguồn từ việc tối ưu hóa tác vụ quân sự trong Thế Chiến II, sau đó mở rộng sang quản trị và kinh doanh. Ngày nay, phương pháp OR được áp dụng trong **quy hoạch đô thị, lập lịch trình, quản lý chuỗi cung ứng, vận tải, tài chính, y tế, sản xuất** và nhiều lĩnh vực khác. Chẳng hạn, OR giúp giải các bài toán về **phân bổ nguồn lực, định tuyến vận tải, quản lý tồn kho, xếp lịch nhân sự**, lựa chọn địa điểm cơ sở, hoạch định dự án,... (Operations research - Wikipedia). Mục tiêu chung là hỗ trợ tổ chức ra quyết định **định lượng** tốt hơn, tìm được phương án **hiệu quả nhất** trong bối cảnh các nguồn lực có hạn và điều kiện ràng buộc phức tạp.

Ví dụ minh họa: Một công ty sản xuất muốn tối đa hóa lợi nhuận từ hai sản phẩm A và B. Giả sử lợi nhuận thu được khi bán mỗi đơn vị A là 40 USD và mỗi đơn vị B là 30 USD. Công ty có

giới hạn về nguyên vật liệu và nhân công, chẳng hạn nguyên liệu chỉ đủ sản xuất 80 đơn vị và thời gian lao động chỉ đủ cho 50 đơn vị sản phẩm. Bài toán có thể mô hình hóa như sau: chọn lượng sản xuất *x* (đơn vị A) và *y* (đơn vị B) để **tối đa** hóa `Z = 40x + 30y` (hàm mục tiêu: tổng lợi nhuận), dưới các ràng buộc `x + y ≤ 80` (hạn chế nguyên liệu), `2x + y ≤ 100` (hạn chế nhân công, giả sử A tốn 2 giờ, B tốn 1 giờ lao động mỗi đơn vị) và `x, y ≥ 0` (không sản xuất âm). Đây là một mô hình tối ưu hóa tuyến tính đơn giản. Giải bằng phương pháp OR sẽ cho biết phương án `(x, y)` tối ưu (ví dụ: sản xuất bao nhiêu A, bao nhiêu B) để đạt lợi nhuận cao nhất trong phạm vi các nguồn lực cho phép.

2. Xây dựng mô hình LP và MIP – Phương pháp giải quyết bài toán tối ưu

Mô hình toán học cho bài toán tối ưu: Để xây dựng mô hình, ta thực hiện các bước: (1) Xác định *biến quyết định* – các đại lượng cần tìm để quyết định phương án; (2) Xác định *hàm mục tiêu* – tiêu chí cần tối ưu hóa, biểu diễn dưới dạng hàm của các biến quyết định; (3) Xác định *các ràng buộc* – tập hợp các phương trình hoặc bất đẳng thức mô tả giới hạn về tài nguyên, quy tắc vận hành hoặc quan hệ logic giữa các biến. Nếu **hàm mục tiêu và mọi ràng buộc đều là biểu thức tuyến tính** của các biến, ta được một mô hình **quy hoạch tuyến tính (Linear Programming – LP)**. LP là một trường hợp đặc biệt của tối ưu hóa, trong đó "các yêu cầu được biểu diễn bằng quan hệ tuyến tính" (Linear problem - wiki.openmod-initiative.org). Mô hình LP thường được viết dưới dạng chuẩn: **tối đa** (hoặc tối thiểu) hàm tuyến tính `c^T x` với `x` là vector biến, thỏa mãn `A x ≤ b` (các ràng buộc tuyến tính) và thường kèm `x ≥ 0`. Nếu mô hình có thêm **biến nguyên** – biến bắt buộc nhận giá trị nguyên (ví dụ 0/1 hoặc số nguyên không âm), ta có mô hình **quy hoạch nguyên hỗn hợp (Mixed Integer Programming – MIP)**. MIP cho phép biểu diễn các quyết định rời rạc, chẳng hạn biến nhị phân 0-1 đại diện cho lựa chọn "có hoặc không" (mua một máy mới, xây một kho hàng, v.v.) (Mixed-Integer Programming (MIP) – A Primer on the Basics - Gurobi Optimization). Nhờ đó, MIP có khả năng mô hình hóa nhiều bài toán tổ hợp phức tạp (ví dụ bài toán phân công, lịch trình, phủ tập hợp, v.v.) mà LP thuần túy không biểu diễn được.

Phương pháp giải bài toán LP: Các mô hình LP có thể được giải hiệu quả bằng các **thuật toán đa thức** hoặc giả đa thức. Thuật toán nổi tiếng nhất là **simplex** do George Dantzig đề xuất năm 1947. Thuật toán simplex duyệt qua các đỉnh của miền nghiệm lồi đa diện và tìm được nghiệm tối ưu của LP. Trong thực tế, simplex rất hiệu quả và giải được các bài toán quy mô lớn. Ngoài ra, còn có các phương pháp **điểm trong (interior-point)** – ví dụ thuật toán đường trung tâm – với độ phức tạp đa thức, thường được dùng trong các solver hiện đại cho LP. Kết quả là các bài toán LP với hàng **triệu biến và ràng buộc** vẫn có thể được giải trong thời gian hợp lý bằng các phần mềm tối ưu chuyên dụng.

Phương pháp giải bài toán MIP: Bài toán có biến nguyên thuộc loại **NP-khó**, nghĩa là khó giải hơn nhiều so với LP. Tuy nhiên, có những phương pháp tổng quát để tìm nghiệm **tối ưu toàn cục** cho MIP. Phổ biến nhất là phương pháp **nhánh – cận (Branch and Bound)** kết hợp với kỹ thuật mặt phẳng cắt (**cutting planes**), thường gọi chung là **nhánh – cắt (Branch and Cut)**. Ý tưởng chính: giải bài toán LP relax (coi mọi biến liên tục, bỏ qua ràng buộc nguyên) để được một cận trên/cận dưới cho giá trị tối ưu. Nếu nghiệm LP thỏa mãn tính nguyên, đó chính là nghiệm tối ưu của MIP. Nếu không, chọn một biến có giá trị phân số và phân nhánh thành hai trường hợp (nhánh) bằng cách thêm ràng buộc chặn giá trị phân số đó (ví dụ thêm `x ≤ 5` và `x ≥ 6` nếu

LP cho `x = 5.7` (Mixed-Integer Programming (MIP) – A Primer on the Basics - Gurobi Optimization)). Mỗi nhánh tạo thành một bài toán con với không gian nghiệm bị thu hẹp. Áp dụng đệ quy quá trình này sẽ tạo ra một cây tìm kiếm các trường hợp. Thuật toán loại bỏ (cắt tỉa) sớm những nhánh không thể cho nghiệm tốt hơn nghiệm tốt nhất hiện tại (dựa trên cận - do đó có tên "nhánh và cận"). Bằng cách này, ta sẽ tìm được nghiệm tốt nhất thỏa mãn tính nguyên. Các trình tối ưu hiện đại đều **giải MIP bằng thuật toán nhánh – cận kết hợp với kỹ thuật LP** (giải các bài toán LP ở từng node của cây) (Mixed-Integer Programming (MIP) – A Primer on the Basics - Gurobi). Ngoài ra, chúng còn tự động tạo thêm các *mặt phẳng cắt* (cut) – những bất đẳng thức thắt chặt miền nghiệm, cắt bỏ phần nghiệm phân số nhưng không loại bỏ nghiệm nguyên – để tăng hiệu quả. Các cải tiến như cắt Gomory, cắt mặt phẳng, cắt tạo bởi CGL,… giúp giảm kích thước cây nhánh đáng kể. Nhờ những phương pháp này, nhiều bài toán MIP thực tế với hàng trăm nghìn biến nhị phân vẫn có thể giải được đến tối ưu bằng phần mềm trong thời gian chấp nhận được, mặc dù bài toán tổng quát thuộc loại NP-khó.

Ví dụ minh họa: Để thấy sự khác biệt LP/MIP, xét bài toán chọn dự án đầu tư: Giả sử có 5 dự án (biến 0-1: chọn hay không), mỗi dự án có lợi nhuận và chi phí, với tổng ngân sách giới hạn. Nếu coi biến có thể chia nhỏ (đầu tư một phần dự án), bài toán thành LP và nghiệm tối ưu có thể cho các giá trị phân số (đầu tư một phần mỗi dự án). Nhưng vì dự án phải hoặc làm hoặc không, ta thêm ràng buộc biến nhị phân, biến bài toán thành MIP (bài toán dạng knapSack). Nghiệm LP relax có thể vi phạm tính nguyên (ví dụ "làm 0.7 dự án A"), nhưng thuật toán nhánh – cận sẽ tìm ra nghiệm nguyên tốt nhất (một tổ hợp dự án tối ưu trong ngân sách). Bài toán knapSack với 5 dự án rất nhỏ, có thể liệt kê mọi khả năng. Nhưng với 500 hoặc 5000 dự án, ta cần nhánh – cận và các kỹ thuật tối ưu hiện đại để giải hiệu quả.

3. So sánh Excel Solver (miễn phí) và CBC trên Python

Tổng quan về hai công cụ: *Excel Solver* là tiện ích giải tối ưu tích hợp trong Microsoft Excel, do Frontline Systems phát triển. Phiên bản miễn phí đi kèm Excel sử dụng các thuật toán như Simplex LP, GRG Nonlinear và Evolutionary để giải những mô hình nhỏ đến trung bình. *CBC (Coin-or Branch and Cut)* là bộ giải tối ưu mã nguồn mở cho bài toán LP/MIP, được phát triển bởi dự án COIN-OR. CBC thường được sử dụng thông qua các giao diện lập trình (ví dụ Python thông qua PuLP, OR-Tools, …) hoặc add-in như OpenSolver trên Excel. Sau đây là so sánh chi tiết:

- **Giới hạn số biến và ràng buộc:** Excel Solver bản chuẩn (miễn phí) bị giới hạn về kích thước mô hình. Cụ thể, *tối đa 200 biến quyết định* (ô có thể thay đổi) (vba - Reaching maximum number of solver variables? - Stack Overflow) và *tối đa 100 ràng buộc* (không kể các ràng buộc biến đơn giản như biến ≥ 0 hoặc biến nguyên) (Standard Excel Solver - Dealing with Problem Size Limits - Continued | solver). Nếu vượt quá, Solver sẽ báo lỗi "too many variable cells" hoặc không chạy. Ngược lại, CBC **không áp dụng giới hạn cố định** về số biến và ràng buộc – bạn có thể xây mô hình lớn tùy ý, miễn là đủ bộ nhớ máy tính (Solver Max - OpenSolver). Trên thực tế, OpenSolver (add-in Excel dùng CBC) cho phép giải các mô hình với **hàng nghìn biến và ràng buộc** mà không gặp rào cản kích thước nhân tạo (Solver Max - OpenSolver). Như vậy, về khả năng mở rộng quy mô mô hình, CBC linh hoạt và vượt trội hơn hẳn so với Excel Solver bản miễn phí.

- **Tính linh hoạt và khả năng mở rộng:** Excel Solver tích hợp trong Excel nên thuận tiện cho những ai quen làm việc với bảng tính, nhưng lại **hạn chế về tính linh hoạt**. Người dùng bị ràng buộc trong môi trường Excel, mô hình phải được biểu diễn bằng các ô tính và công thức Excel, và kích thước mô hình lớn sẽ làm bảng tính trở nên cồng kềnh, khó quản lý. Trong khi đó, CBC chạy thông qua mã Python (hoặc các giao diện khác) nên rất linh hoạt: ta có thể **tự động hóa** quá trình giải nhiều lần, tích hợp vào ứng dụng, kết hợp với các thư viện Python khác để tiền xử lý hoặc hậu xử lý dữ liệu. CBC xử lý tốt các mô hình lớn vì nó được viết bằng C++ hiệu năng cao. Thêm nữa, CBC là mã nguồn mở, có thể chạy trên các máy chủ độc lập không cần giao diện, nên có thể mở rộng quy mô tính toán (ví dụ chạy trên server, song song nhiều tiến trình) dễ dàng hơn so với việc dùng Excel trên máy cá nhân. Tóm lại, với CBC/Python ta có **môi trường phát triển linh hoạt**, phù hợp giải các bài toán tối ưu lặp đi lặp lại hoặc tích hợp vào hệ thống, còn Excel Solver chủ yếu phù hợp giải tương tác các bài toán nhỏ trong Excel.
- **Hiệu suất tính toán và độ chính xác:** Đối với các mô hình nhỏ (vài trăm biến), Excel Solver và CBC đều có thể tìm nghiệm tối ưu khá nhanh. Tuy nhiên, trên các bài toán tối ưu **phức tạp hoặc kích thước lớn**, CBC thường cho **hiệu suất cao hơn**. Lý do là CBC là một solver chuyên dụng hiện đại, sử dụng các kỹ thuật cutting-edge (presolve, cắt, nhánh – cận hiệu quả). Nghiên cứu cho thấy CBC (OpenSolver) thường cho kết quả tương đương hoặc nhanh hơn Solver của Excel trong nhiều trường hợp; ví dụ, các bài toán kiểu knapSack kích thước lớn mà Excel Solver mất hàng giờ, thì OpenSolver (dùng CBC) có thể giải **gần như tức thì** () nhờ các kỹ thuật tiền xử lý và tăng cường bài toán mà CBC áp dụng. Nhìn chung, CBC được xem là "bộ giải tối ưu hiện đại hơn so với Solver của Excel" nên cho hiệu năng cải thiện trên các bài toán khó (). Về **độ chính xác**, cả hai công cụ đều dùng số học dấu phẩy động kép và có độ chính xác cao trong tính toán. Tuy nhiên, cần lưu ý **thiết lập dung sai tối ưu (optimality tolerance)**: Excel Solver mặc định cho phép dừng khi tìm được nghiệm trong vòng 5% lân cận tối ưu thật (Tolerance = 0.05) (Standard Excel Solver - Tolerance Option for Integer Problems | solver) nhằm giảm thời gian chạy, do đó nghiệm trả về có thể **chưa phải tối ưu tuyệt đối** (chỉ gần tối ưu). Ví dụ, với Tolerance 5%, Solver có thể dừng với phương án có giá trị hàm mục tiêu kém tối ưu thật 5%. Người dùng phải chỉnh Tolerance = 0% nếu muốn tìm đúng optimum. Trong khi đó, CBC mặc định cố gắng tìm **nghiệm tối ưu toàn phần (optimal)** với khoảng cách sai số rất nhỏ (thường <0.0001). Điều này có nghĩa CBC thường tìm được nghiệm **chính xác hơn** (đúng optimum) nếu cho phép chạy đủ lâu, trong khi Solver Excel nếu người dùng không tinh chỉnh có thể dừng sớm ở nghiệm gần tối ưu. Ngoài ra, CBC nhìn chung ổn định với các mô hình **số liệu phức tạp** (ma trận hệ số thưa, hệ số rất lớn hoặc rất nhỏ), thậm chí đôi khi vượt trội một số solver thương mại trong trường hợp số học khó (Cbc and Clp performance - Optimization (Mathematical) - Julia Programming Language). Tóm lại, về độ tin cậy và chính xác, CBC cho phép đạt tới nghiệm tối ưu đúng với kiểm soát tốt về sai số, còn Excel Solver tiện dụng nhưng cần cẩn thận cấu hình để đảm bảo độ chính xác cao.
- **Lợi ích khi sử dụng CBC thay vì Excel Solver:** Sử dụng CBC (thông qua Python/PuLP hoặc các công cụ như OpenSolver) mang lại nhiều lợi ích: (1) **Giải được các bài toán lớn** vượt giới hạn của Excel Solver (ví dụ hàng nghìn biến, ràng buộc); (2) **Tốc độ và khả năng giải bài toán khó tốt hơn** nhờ thuật toán tiên tiến – như dẫn chứng, những bài toán mất hàng giờ trong Excel có thể giải nhanh bằng CBC (); (3) **Không tốn chi phí bản quyền**: CBC là miễn phí và mã nguồn mở, trong khi để vượt giới hạn 200 biến của

Excel Solver, người dùng thường phải mua bản **Premium Solver** hoặc Analytic Solver của Frontline (có phí); (4) **Tự động hóa và tích hợp dễ dàng**: Với CBC/Python, ta có thể viết script chạy nhiều lần, tích hợp vào quy trình doanh nghiệp (ví dụ đọc dữ liệu từ cơ sở dữ liệu, giải tối ưu, rồi ghi kết quả trả lại) – điều này khó thực hiện linh hoạt với môi trường Excel thuần túy; (5) **Cộng đồng và hỗ trợ**: CBC được cộng đồng COIN-OR phát triển, tài liệu mở, nhiều dự án và diễn đàn hỗ trợ, trong khi Excel Solver là công cụ đóng. Tóm lại, **CBC đem lại sức mạnh tính toán và quy mô công nghiệp** cho các bài toán tối ưu, thích hợp cho ứng dụng thực tế lớn, trong khi Excel Solver phù hợp cho bước đầu làm quen hoặc những bài toán nhỏ, đơn lẻ.

4. Các thư viện tối ưu hóa trong Python (miễn phí và thương mại)

Python hiện có hệ sinh thái phong phú hỗ trợ xây dựng và giải mô hình tối ưu hóa, từ thư viện mã nguồn mở miễn phí đến phần mềm thương mại hàng đầu. Dưới đây là một số thư viện tiêu biểu:

- **PuLP:** PuLP là một thư viện modeling *miễn phí* thuần Python, cho phép người dùng xây dựng mô hình LP/MIP bằng cách khai báo biến, hàm mục tiêu, ràng buộc một cách trực quan. PuLP sẽ tự động sinh mô hình dạng .LP hoặc .MPS và gọi solver thích hợp để giải (Third-Party Tools for Gurobi - Gurobi Optimization). Mặc định PuLP tích hợp sẵn với **CBC** (nếu cài CBC) làm solver nội, do đó dùng PuLP là có thể giải tối ưu miễn phí ngay. PuLP cũng hỗ trợ giao tiếp với các solver thương mại (như Gurobi, CPLEX) nếu người dùng có license. Ưu điểm của PuLP là cú pháp đơn giản, phù hợp cho người mới học OR dùng Python hoặc giải nhanh các bài toán quy mô vừa. Nhược điểm: hiệu suất có thể kém hơn chút so với các modeling khác khi mô hình rất lớn (do thuần Python), và nó chủ yếu hỗ trợ LP/MIP (không chuyên cho phi tuyến).
- **Google OR-Tools:** OR-Tools là bộ công cụ tối ưu hóa mã nguồn mở do Google phát triển (hoàn toàn miễn phí kể cả cho mục đích thương mại). OR-Tools cung cấp nhiều solver mạnh: **CP-SAT** (để giải MILP và các bài toán ràng buộc rời rạc bằng ràng buộc thỏa mãn – ràng buộc hỗn hợp SAT, hiện được đánh giá rất cao về hiệu năng), solver tuyến tính (sử dụng CLP/CBC), solver quy hoạch ràng buộc (CP) cổ điển, và đặc biệt nổi tiếng với **thuật toán định tuyến (Vehicle Routing)**. OR-Tools viết bằng C++ và có API cho Python, Java, .NET (cplex - What is the purpose of libraries like Pyomo and Google OR tools? - Operations Research Stack Exchange). Một điểm thuận lợi là OR-Tools **hoàn toàn miễn phí và không cần thư viện thương mại nền** – "được viết bằng C++ từ đầu, không gọi thư viện thương mại nền, do đó người dùng không cần tốn chi phí mua CPLEX hay Gurobi" (cplex - What is the purpose of libraries like Pyomo and Google OR tools? - Operations Research Stack Exchange). OR-Tools rất thích hợp cho các bài toán tối ưu tổ hợp phức tạp như lập lịch, định tuyến, tối ưu vận hành… mà Google đã dùng trong nội bộ. Với LP/MIP thông thường, OR-Tools hiện dùng CP-SAT làm mặc định (một solver kết hợp kỹ thuật ràng buộc và MILP rất mạnh), có thể giải hiệu quả nhiều bài toán mà CBC gặp khó. Thư viện này đang được cập nhật liên tục bởi Google và có cộng đồng người dùng lớn.
- **Pyomo:** Pyomo là một **ngôn ngữ mô hình hóa đại số** trong Python, tương tự như AMPL nhưng miễn phí. Pyomo cho phép biểu diễn các mô hình tối ưu hóa một cách tự nhiên bằng cách định nghĩa các tập hợp, tham số, biến, ràng buộc, hàm mục tiêu sử dụng cú

pháp Python. Ưu điểm lớn là Pyomo có thể mô hình hóa **nhiều lớp bài toán**: LP/MIP, quy hoạch phi tuyến (NLP), quy hoạch động, tối ưu đa mục tiêu, v.v. Pyomo đóng vai trò *interface* tới nhiều solver – cả open-source lẫn commercial. Nó hỗ trợ các solver như CBC, GLPK (miễn phí) và cũng có thể gọi Gurobi, CPLEX, IPOPT... nếu có cài đặt. Theo mô tả, Pyomo cho phép người dùng "định nghĩa mô hình tối ưu ở mức trừu tượng cao (biến, ràng buộc, mục tiêu) và cung cấp giao diện đến nhiều solver khác nhau, giúp cùng một mô hình có thể thử nghiệm với nhiều solver" (cplex - What is the purpose of libraries like Pyomo and Google OR tools? - Operations Research Stack Exchange). Pyomo hữu ích trong các dự án lớn, đặc biệt trong học thuật và nghiên cứu, nơi cần linh hoạt thay đổi solver và biểu diễn mô hình phức tạp. Nhược điểm: do trừu tượng cao và tổng quát, Pyomo có thể chậm hơn ở bước dựng mô hình cho những bài toán rất lớn so với các công cụ chuyên biệt.

- **Gurobi:** Gurobi Optimizer là một trong những solver tối ưu **thương mại hàng đầu thế giới** cho LP, MILP, và nhiều biến thể (quadratic, QCP…). Gurobi nổi tiếng về tốc độ giải MILP cực nhanh nhờ các thuật toán nhánh – cận song song và tối ưu cao. Gurobi cung cấp Python API rất thuận tiện, cho phép xây dựng mô hình bằng code Python (tương tự PuLP nhưng hiệu quả hơn) và gọi solver Gurobi nội bộ. Theo đánh giá, "Gurobi là solver thương mại đẳng cấp thế giới" trong lĩnh vực tối ưu toán học (What software is used in the field these days? : r/optimization - Reddit). Nhược điểm là Gurobi yêu cầu mua **license** khá đắt cho sử dụng thương mại (dù miễn phí cho mục đích học thuật/phi lợi nhuận). Trong môi trường doanh nghiệp cần giải các bài toán tối ưu lớn phức tạp trong thời gian ngắn, Gurobi thường được lựa chọn vì hiệu suất vượt trội so với các solver mã mở. Gurobi cũng hỗ trợ nhiều ngôn ngữ khác (C++, Java, .NET) ngoài Python.
- **CPLEX:** IBM ILOG CPLEX Optimizer là solver tối ưu thương mại lâu đời và uy tín, hiệu năng tương đương Gurobi trên nhiều loại bài toán. CPLEX có khả năng giải rất nhanh các bài toán LP, đặc biệt "về bài toán tuyến tính, hiếm công cụ nào sánh được với CPLEX – solver này cực kỳ nhanh và chính xác" (Use of commercial solvers in the engineering and simulations ...). CPLEX cũng hỗ trợ MILP, QP, QCP và có giao diện Python (thông qua gói `docplex` của IBM hoặc API Python gốc). Tương tự Gurobi, CPLEX yêu cầu license thương mại (miễn phí hạn chế cho sinh viên/học giả). Nhiều tổ chức lớn sử dụng CPLEX trong các ứng dụng hoạch định tối ưu. So với Gurobi, ưu thế của CPLEX là tích hợp tốt với các sản phẩm IBM và có lịch sử ứng dụng lâu năm; còn Gurobi thường nhỉnh hơn về tốc độ trên một số benchmark mới. Cả hai đều thuộc nhóm solver thương mại mạnh nhất hiện nay.
- **Các thư viện/solver khác:**
    - *GLPK (GNU Linear Programming Kit):* Solver mã nguồn mở do GNU phát triển, hỗ trợ LP và MILP cỡ nhỏ đến trung bình. GLPK dễ sử dụng (có gói Python `swiglpk` hoặc thông qua PuLP/Pyomo). Hiệu năng của GLPK kém hơn CBC một chút trên bài toán lớn, nhưng đủ tốt cho mô hình nhỏ và hoàn toàn miễn phí (GPL license).
    - *SCIP:* Solver tối ưu rời rạc mạnh do Zuse Institute Berlin phát triển, được đánh giá là solver MILP mã nguồn mở mạnh nhất (thường chỉ thua kém Gurobi/CPLEX ~20-30% tốc độ). SCIP miễn phí cho mục đích học thuật nhưng có hạn chế giấy phép nếu dùng thương mại. Python có thể gọi SCIP qua `PySCIPOpt`.

- *CVXOPT/CVXPY:* Thư viện Python cho tối ưu **lồi phi tuyến** và quy hoạch dạng cone. CVXOPT cung cấp solver nội cho QP, NLP nhỏ; CVXPY là gói modeling bậc cao cho tối ưu lồi, tự động gọi solver phù hợp (như ECOS, SCS, OSQP…). Đây phù hợp cho bài toán tối ưu lồi (không hẳn LP/MIP).
- *Các solver thương mại khác:* **FICO Xpress** (đối thủ của CPLEX/Gurobi, hiệu năng cao, có Python API), **MOSEK** (mạnh về tối ưu hình nón, QP), **LocalSolver** (cho bài toán tổ hợp quy mô rất lớn bằng heuristic), v.v. Những công cụ này thường dùng trong các doanh nghiệp đặc thù hoặc nghiên cứu nâng cao.

Như vậy, người dùng Python có rất nhiều lựa chọn: **nếu muốn miễn phí mã nguồn mở**, có thể dùng PuLP/OR-Tools/Pyomo kết hợp với CBC, GLPK, SCIP; **nếu cần hiệu năng tối đa** và có ngân sách, có thể mua Gurobi hoặc CPLEX. Việc lựa chọn tùy thuộc vào yêu cầu bài toán (quy mô, tính chất) và điều kiện (miễn phí hay thương mại, tích hợp với hệ thống sẵn có, v.v.).

5. Phân tích chuyên sâu về CBC – công cụ tối ưu miễn phí trên PuLP

Cách hoạt động của CBC

CBC (Coin-or Branch and Cut) là một bộ giải *quy hoạch hỗn hợp tuyến tính (MILP)* mã nguồn mở thuộc dự án COIN-OR. CBC được viết bằng C++ và có thể sử dụng như một thư viện nhúng hoặc một chương trình độc lập giao tiếp qua file (GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver). Kiến trúc của CBC tuân thủ mô hình giải **Branch-and-Cut** hiện đại: Nó sử dụng **bộ giải LP** COIN-OR CLP (Coin Linear Programming) để giải các bài toán con tuyến tính tại các node của cây nhánh – cận, và dùng **thư viện cắt CGL (Cut Generation Library)** của COIN-OR để sinh ra các mặt phẳng cắt tự động (NEOS Server: Cbc) (ví dụ cắt Gomory, cắt knapsack, v.v.). Cụ thể, khi giải một bài toán MILP, CBC trước tiên chạy *presolve* (đơn giản hóa mô hình bằng cách loại bỏ biến, ràng buộc dư thừa, cố định biến nếu có thể,…). Sau đó giải LP relax; nếu nghiệm không nguyên, CBC sẽ chọn một biến phân số để phân nhánh, tạo hai node con với ràng buộc chặt hơn. Quá trình nhánh – cận diễn ra theo chiến lược nhất định (CBC có thể tùy chỉnh chiến lược nhánh, thứ tự chọn biến,…). Tại mỗi node, trước khi giải LP, CBC kích hoạt các **cutting planes**: dựa trên trạng thái hiện tại, nó sử dụng CGL để cắt thêm các bất đẳng thức hợp lệ, nhằm loại bỏ vùng nghiệm phân số và thắt chặt không gian nghiệm (NEOS Server: Cbc). Những kỹ thuật như vậy giúp giảm kích thước cây tìm kiếm đáng kể. CBC tiếp tục cho đến khi tìm được nghiệm nguyên tối ưu (hoặc hết thời gian, hoặc người dùng đặt ngưỡng dừng).

Vì là phần mềm mã nguồn mở, CBC không ngừng được cải tiến bởi cộng đồng. Code CBC do John Forrest (IBM) khởi xướng và nhiều người đóng góp, nằm dưới sự bảo trợ của COIN-OR Foundation (GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver) (GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver). CBC tích hợp dễ dàng với nhiều **ngôn ngữ mô hình hóa và môi trường**: ví dụ có plugin cho AIMMS, AMPL, GAMS, MATLAB, R,… (GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver); đặc biệt trong Python, CBC là solver mặc định của **PuLP, Pyomo, OR-Tools** (trước đây), **Google OR-Tools** (tùy chọn) và giao tiếp được qua các gói như `python-mip`, `CyLP` (GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver). CBC cũng được dùng làm backend cho **OpenSolver** (Excel) và trong **NEOS Server** (máy chủ giải tối ưu trực tuyến) (GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver) (NEOS Server: Cbc).

Điều này cho thấy tính **mở và linh hoạt** của CBC – người dùng có thể tiếp cận CBC từ nhiều nền tảng khác nhau.

Tóm lại, CBC hoạt động tương tự các solver MILP thương mại: dựa trên thuật toán nhánh – cắt, sử dụng LP relax và cắt phẳng, có đầy đủ các tính năng như cài đặt thời gian tối đa, dung sai tối ưu, v.v. Mặc dù là mã nguồn mở, CBC được thiết kế đủ *"thông minh"* và tổng quát để giải hiệu quả nhiều bài toán tối ưu hỗn hợp.

Hiệu suất và độ tin cậy của CBC trên các bài toán lớn

CBC được đánh giá là một trong những solver MILP mã mở mạnh nhất hiện nay. Theo AMPL, "CBC là một solver nguồn mở **mạnh mẽ (robust)** cho bài toán nguyên hỗn hợp, thuộc dự án COIN-OR, và **xuất sắc trong việc giải các mô hình phức tạp**…" (Linear Solvers - AMPL Optimization). Thực tế, CBC thường được dùng làm chuẩn so sánh cho các solver miễn phí. Hiệu suất của CBC trên các bài toán quy mô vừa (vài nghìn biến 0-1) rất tốt – nhiều trường hợp cho kết quả ngang ngửa solver thương mại. Thậm chí, có những bài toán **số học khó** (hệ số ma trận gây suy biến hoặc bất ổn số cao) mà CBC xử lý ổn định hơn một số solver thương mại, như ghi nhận: "CBC cho hiệu năng rất tốt, thậm chí vượt một vài solver thương mại trên những bài toán số học phức tạp" (Cbc and Clp performance - Optimization (Mathematical) - Julia Programming Language). Tuy nhiên, nhìn chung, **solver thương mại** (Gurobi, CPLEX) vẫn nhanh hơn CBC trên các bài toán MILP rất lớn (ví dụ >100k biến nhị phân), nhờ thuật toán được tinh chỉnh chuyên sâu và khả năng tính toán song song tối ưu. Dù vậy, khoảng cách đang thu hẹp khi CBC và các solver mở khác (SCIP) liên tục cải tiến.

Về **độ tin cậy**, CBC được tin cậy sử dụng trong nhiều ứng dụng thực tiễn lẫn học thuật. Do CBC là mã mở, người dùng có thể kiểm tra log, tinh chỉnh tham số cắt, chiến lược nhánh nếu cần kết quả tốt hơn cho bài toán cụ thể. CBC cũng hỗ trợ **tính năng cơ bản** như tính nghiệm ban đầu (MIP start), dừng theo gap tối ưu, giới hạn thời gian, xuất nhập mô hình ở các định dạng tiêu chuẩn (LP, MPS). Hơn nữa, vì CBC tích hợp trong các công cụ như OpenSolver và đã giải nhiều bài toán thực tế, nên chất lượng và tính đúng đắn của nó đã được kiểm chứng rộng rãi.

Một minh chứng cho sức mạnh của CBC: trong bài báo giới thiệu OpenSolver (2010), tác giả nhận xét "hiệu năng của OpenSolver (dùng CBC) tương đương hoặc tốt hơn Solver của Excel. CBC là bộ giải hiện đại hơn, nhờ đó cải thiện hiệu năng đáng kể trên một số bài toán khó. Ví dụ, những bài toán kiểu knapSack lớn mất hàng giờ với Excel Solver thì được giải *gần như tức thì* bằng OpenSolver (CBC) do các kỹ thuật tiền xử lý và tăng cường bài toán mà CBC áp dụng" () (). Điều này cho thấy **CBC đủ khả năng giải các mô hình lớn** mà trước đây ngoài tầm với của các công cụ thông thường.

Tất nhiên, với các bài toán cực lớn (hàng triệu biến, hàng trăm nghìn ràng buộc), **solver thương mại vẫn vượt trội** nhờ tối ưu hoá sâu về thuật toán và phần cứng (tận dụng đa luồng hiệu quả hơn). Nhưng CBC vẫn có thể giải các bài toán khá lớn ở mức chấp nhận được, và đặc biệt hữu ích khi ngân sách không cho phép mua solver đắt tiền. Ngoài ra, CBC có thể kết hợp với hạ tầng song song (ví dụ chạy nhiều phiên bản với các seed khác nhau, hoặc cài đặt trên cụm máy chủ) để mở rộng khả năng tính toán.

Tóm lại, **CBC có hiệu suất rất ấn tượng trong nhóm solver miễn phí**, đủ đáng tin cậy cho nhiều ứng dụng thực tiễn. Nó có thể không phải luôn là nhanh nhất trong mọi trường hợp, nhưng thường tìm được nghiệm tối ưu **đảm bảo đúng**. Với việc cộng đồng liên tục cải tiến, CBC ngày càng tiệm cận hiệu năng của các solver thương mại. Vì vậy, CBC là lựa chọn hàng đầu khi cần một giải pháp tối ưu hóa **miễn phí, tin cậy và đủ mạnh** cho các bài toán OR lớn.

6. Giới thiệu về ERX Solver – Mô hình hóa tối ưu qua Excel và CBC

ERX Solver là một giải pháp kết hợp Excel và CBC nhằm giúp người dùng xây dựng và giải bài toán tối ưu một cách dễ dàng mà **không cần lập trình chuyên sâu**. Ý tưởng chính của ERX Solver là tận dụng **giao diện bảng tính Excel quen thuộc** để người dùng mô hình hóa bài toán (định nghĩa biến, hàm mục tiêu, ràng buộc) một cách trực quan, sau đó sử dụng CBC làm "bộ xử lý" toán học ở phía sau để tìm lời giải tối ưu. Cách tiếp cận này thân thiện với những người không rành về lập trình Python: thay vì viết mã khai báo mô hình (như PuLP hoặc Pyomo), người dùng chỉ cần điền các tham số và công thức vào một file Excel mẫu.

Phương pháp đặt biến số/hàm số trong Excel đơn giản: ERX Solver cung cấp một khuôn mẫu (template) Excel – thường là một sheet (ví dụ đặt tên "Setting") – trong đó người dùng liệt kê các thông tin của mô hình. Cụ thể, file Excel sẽ có các cột để điền:

- **Phương pháp tối ưu (Method):** chọn "maximize" hoặc "minimize" tùy thuộc bài toán tối đa hóa lợi ích hay tối thiểu hóa chi phí (có thể viết tắt "Max"/"Min"). Nếu là bài toán tìm nghiệm thỏa giá trị mục tiêu cố định, có thể có tùy chọn "value of" kèm giá trị mục tiêu mong muốn.
- **Hàm mục tiêu (Objective function):** người dùng nhập biểu thức tuyến tính của các biến, ví dụ `"40*x_A + 30*x_B"` cho hàm lợi nhuận cần tối đa. Biểu thức này được nhập dưới dạng chuỗi (string) sử dụng tên biến, chứ không cần tham chiếu ô Excel. Điều này nghĩa là người dùng viết công thức giống hệt như trên giấy (với ký hiệu biến) ngay trong ô Excel, thay vì phải trải công thức ra nhiều ô.
- **Danh sách biến quyết định (Variables):** mỗi biến là một hàng, bao gồm tên biến (ví dụ `x_A`), loại biến (liên tục, nguyên, nhị phân – đánh dấu bằng Yes/No trong các cột "Integer", "Binary"), và tùy chọn "Non-negative" nếu biến phải ≥ 0. Người dùng không cần tạo thủ công nhiều ô cho từng biến; chỉ cần liệt kê tên và thuộc tính, ERX Solver sẽ tự khởi tạo các biến tương ứng trong mô hình.
- **Danh sách ràng buộc (Constraints):** mỗi ràng buộc được nhập như một phương trình hoặc bất đẳng thức dạng text, ví dụ: `x_A + x_B <= 80` (giới hạn nguyên liệu), `2*x_A + x_B <= 100` (giới hạn nhân công), v.v. Tương tự hàm mục tiêu, các ràng buộc được viết bằng tên biến, dấu so sánh `<=, >=, =` và các hằng số số học. Người dùng có thể viết tất cả ràng buộc trên các dòng riêng biệt dưới cột "Constraint".

Với cách thức trên, việc thiết lập mô hình tối ưu trong Excel trở nên **rất trực quan** – người dùng làm đúng những gì họ vẫn làm trên bảng (liệt kê biến và công thức), không phải học cú pháp lập trình. Điều này hạ thấp rào cản ứng dụng OR cho các nhà quản lý hay nhân viên không chuyên CNTT: họ có thể dùng kỹ năng Excel sẵn có để mô hình hóa vấn đề tối ưu. Chẳng hạn, một người quản lý kho có thể tạo file Excel, điền biến hàng tồn kho, ràng buộc sức chứa kho, hàm

mục tiêu chi phí, rồi nhấn nút giải – thay vì phải viết code. ERX Solver lo việc "dịch" những gì trong Excel thành mô hình toán học cho CBC.

Khai thác CBC thông qua Excel trong vận hành thực tiễn: Sau khi người dùng chuẩn bị xong file mô tả mô hình trong Excel, ERX Solver sẽ sử dụng một đoạn mã Python (hoặc một dịch vụ web) để **đọc file Excel đó, chạy solver CBC, rồi trả kết quả về Excel**. Tức là, ERX Solver đứng giữa làm cầu nối: Excel đóng vai trò giao diện nhập/xuất, còn CBC lo việc tính toán tối ưu phía sau. Cách làm này mang lại trải nghiệm "nhấn nút là có kết quả" ngay trong Excel, rất phù hợp cho các bài toán **quyết định trong doanh nghiệp** – nơi dữ liệu thường đã có sẵn ở dạng bảng tính.

Ví dụ một ứng dụng thực tế: Một công ty vận tải muốn tối ưu lịch trình xe giao hàng mỗi ngày. Nhân viên có thể liệt kê các đơn hàng (điểm đi, điểm đến, trọng lượng, thời hạn) trong Excel, đặt biến quyết định giao đơn hàng i bằng xe k (biến nhị phân), ràng buộc đảm bảo mỗi đơn hàng được đúng một xe nhận, tổng tải các đơn trên mỗi xe không quá trọng tải xe, v.v. – tất cả nhập trong sheet "Setting". Sau đó chạy ERX Solver: chương trình Python sẽ đọc danh sách đơn hàng và ràng buộc, tự động xây dựng mô hình phân công tuyến vận tải, gọi CBC để tối ưu (ví dụ tối thiểu tổng quãng đường hoặc chi phí), rồi ghi kết quả (đơn hàng A -> xe 1, đơn hàng B -> xe 3, ...) trở lại một sheet Excel kết quả. Người dùng cuối chỉ cần xem kết quả trong Excel, có thể trực quan hóa hay điều chỉnh dữ liệu và chạy lại nếu cần. Họ không phải tương tác trực tiếp với code Python hay solver phức tạp.

ERX Solver tận dụng sức mạnh của CBC nên có thể giải quyết các bài toán vận hành quy mô vừa và lớn mà trước đây có thể phải làm thủ công hoặc dùng Excel Solver nhỏ lẻ. Nhờ khả năng tích hợp trong Excel, ERX Solver phù hợp đưa vào quy trình hàng ngày của doanh nghiệp: nhân viên chỉ việc cập nhật file Excel (ví dụ mỗi sáng nhập đơn hàng mới, tồn kho mới), bấm chạy tối ưu, và nhận kế hoạch tối ưu xuất ra Excel sau vài phút. Điều này giúp **ra quyết định nhanh** (nhờ tối ưu tự động) và dựa trên **dữ liệu cập nhật**, thay vì dựa thuần túy vào kinh nghiệm.

Tóm lại, **ERX Solver là một công cụ "giao thoa" giữa dễ dùng và mạnh mẽ**: dễ dùng vì giao diện Excel quen thuộc, mạnh mẽ vì sử dụng CBC – một solver tối ưu hiện đại – làm động cơ tính toán. Công cụ này mở ra khả năng ứng dụng OR rộng rãi hơn trong thực tiễn quản lý và vận hành, khi mà khoảng cách giữa mô hình toán học và người dùng cuối được thu hẹp.

7. Thiết lập input/output với Excel và Python – Quy trình sử dụng ERX Solver

Để vận hành ERX Solver (hoặc hệ thống tương tự kết hợp Excel & CBC), ta thực hiện theo quy trình sau:

Bước 1: Chuẩn bị file Excel đầu vào (input). Người dùng tạo một file Excel (theo mẫu mà ERX Solver quy định) chứa các thông tin mô hình. Thông thường file này sẽ có:

- Một sheet (ví dụ đặt tên "Setting") để nhập **cấu trúc mô hình** như đã mô tả ở trên: gồm dòng chỉ định phương pháp tối ưu (Max/Min), biểu thức hàm mục tiêu, danh sách biến và ràng buộc. Cần đảm bảo điền đúng định dạng mà chương trình yêu cầu (ví dụ viết <= cho

"nhỏ hơn hoặc bằng", dùng dấu `*` cho phép nhân, không dùng ký hiệu Excel như `SUM` hay địa chỉ ô trong biểu thức).

- Nếu bài toán có dữ liệu đầu vào lớn (ví dụ ma trận chi phí, nhu cầu theo nhiều thời kỳ…), file Excel có thể có thêm các sheet chứa *dữ liệu* và tham chiếu chúng trong biểu thức ràng buộc/hàm mục tiêu thông qua các tên đã định nghĩa. Tuy nhiên, phiên bản đơn giản của ERX Solver thường yêu cầu tự "trình bày phẳng" mô hình trong sheet Setting: tức là liệt kê từng ràng buộc một, mỗi ràng buộc là một dòng.
- Kiểm tra lại file input: đảm bảo không sót biến hay ràng buộc, đúng chính tả tên biến (tên dùng nhất quán giữa hàm mục tiêu và ràng buộc), các cột Binary/Integer đánh dấu đúng (Yes/No). Nếu có ô trống không dùng, nên để trống hoàn toàn để chương trình bỏ qua.

Bước 2: Sử dụng Python + CBC để giải bài toán. ERX Solver cung cấp một script Python (hoặc một ứng dụng giao diện) để thực hiện bước này. Khi chạy:

1. **Đọc file Excel:** Chương trình Python sẽ mở file Excel input và đọc sheet "Setting" vào bộ nhớ (sử dụng thư viện `pandas` hoặc `openpyxl`). Mỗi hàng của sheet được xử lý: dòng phương pháp đọc ra `Method` (Maximize/Minimize), dòng hàm mục tiêu đọc biểu thức `Objective function`, các dòng có biến sẽ tạo đối tượng biến tương ứng (dùng `LpVariable` của PuLP), các dòng có ràng buộc sẽ được parse (phân tích chuỗi) thành biểu thức ràng buộc. Quá trình này như **dịch mô hình từ Excel sang mô hình toán** trong code. Ví dụ, nếu sheet có hàng "`x_A + x_B <= 80`", Python sẽ tách vế trái "`x_A + x_B`" và vế phải "`80`", tạo biểu thức tuyến tính `variables['x_A'] + variables['x_B'] <= 80` (với `variables[...]` là các đối tượng PuLP đã tạo).
2. **Thiết lập mô hình PuLP:** Dựa trên dữ liệu đọc được, chương trình khởi tạo một `LpProblem` (vấn đề tối ưu) với hướng Max hoặc Min tùy `Method`. Sau đó thêm hàm mục tiêu (dùng `+=` của PuLP để gán biểu thức mục tiêu cho mô hình). Tiếp đến, lần lượt thêm từng ràng buộc (dùng `+=` để thêm với một tên nhãn). Nếu `Method` là "value of" (tìm nghiệm đạt giá trị mục tiêu cho trước), chương trình sẽ chuyển hàm mục tiêu thành ràng buộc bằng cách thêm ràng buộc `Objective_expr == Value` thay vì tối ưu hóa nó.
3. **Gọi solver CBC:** Chương trình tạo một đối tượng solver CBC (ví dụ `PULP_CBC_CMD(msg=False)` trong PuLP) rồi gọi `problem.solve(solver)` để giải mô hình. Lúc này CBC (cài trên hệ thống) sẽ chạy, giải bài toán và trả kết quả (trạng thái tối ưu, giá trị biến...).
4. **Thu thập kết quả:** Sau khi solver chạy xong, chương trình kiểm tra trạng thái (optimal, infeasible, v.v.). Nếu tìm được nghiệm, nó sẽ truy xuất giá trị của từng biến qua `var.value()` và giá trị hàm mục tiêu qua `value(problem.objective)`. Chương trình có thể tính thêm giá trị thực tế của vế trái và phải từng ràng buộc (để người dùng biết ràng buộc nào căng, ràng buộc nào còn dư địa) bằng cách thay thế biến vào biểu thức hoặc dùng hàm `value()` của PuLP trên biểu thức ràng buộc.

Bước 3: Xuất kết quả ra file Excel. Sau khi có lời giải, ERX Solver ghi kết quả ngược lại file Excel (có thể là file mới hoặc thêm sheet vào file input). Thông thường:

- Tạo một sheet "Variable Cells" liệt kê tên các biến và **giá trị tối ưu** của chúng. Các biến có thể được sắp xếp lại cho có thứ tự (ví dụ x1, x2, x10 thì sort theo số tự nhiên) để dễ theo dõi (optimization.py). Mỗi biến một dòng, kèm giá trị tối ưu.

- Tạo một sheet "Constraints" liệt kê các ràng buộc với **giá trị vế trái tại nghiệm** và **giá trị vế phải**. Ràng buộc có thể được đánh nhãn tự động như C1, C2,… theo thứ tự đọc vào (optimization.py). Nhìn vào đây người dùng biết ràng buộc nào *đạt biên* (ví dụ vế trái = vế phải, nghĩa là sử dụng hết nguồn lực – ràng buộc chặt), ràng buộc nào *nhỏ hơn biên* (còn dư). Điều này giúp hiểu kết quả hơn, giống như báo cáo của Excel Solver. Nếu có trường hợp "value of", chương trình cũng ghi sai lệch giữa giá trị đạt được và mục tiêu yêu cầu (thường sai lệch 0 nếu đạt đúng) (optimization.py).
- Tạo sheet "Summary" ghi tóm tắt trạng thái lời giải (Optimal/Infeasible/…) và **giá trị mục tiêu tối ưu** (optimization.py). Thông tin này cho người dùng biết bài toán có giải được hay không và kết quả tốt nhất là bao nhiêu.

Sau khi ghi xong, chương trình lưu file Excel (ví dụ đặt tên `result.xlsx`) và kết thúc. Người dùng mở file kết quả để xem phương án tối ưu.

Quy trình trên có thể được gói gọn dưới dạng một **nút bấm** trong Excel (nếu dùng add-in) hoặc một trang web tải file. Chẳng hạn, ERX Solver có thể có một giao diện web: người dùng upload file Excel input, server chạy bước 2,3 rồi trả lại file Excel kết quả cho người dùng tải về. Tất cả logic đều có thể tự động hóa.

Ưu điểm của cơ chế input/output bằng Excel: Người dùng cuối chỉ cần thao tác với Excel – một công cụ quen thuộc – mà vẫn tận dụng được sức mạnh của thuật toán tối ưu tiên tiến (CBC). Dữ liệu input và output đều ở dạng bảng, dễ dàng kiểm tra, hiệu chỉnh và tích hợp vào báo cáo. Chẳng hạn, sau khi có kết quả tối ưu, người dùng có thể dùng các hàm Excel, biểu đồ Excel để phân tích thêm (do kết quả đã ở sẵn trong file). Quá trình thử nghiệm các kịch bản cũng tiện lợi: chỉ cần thay vài tham số trong sheet input (ví dụ tăng nguồn lực, thay đổi hệ số lợi nhuận) rồi chạy lại solver để xem ảnh hưởng.

Lưu ý: Khi thiết lập input Excel, cần tuân thủ đúng mẫu quy định, nếu không chương trình có thể báo lỗi (ví dụ viết sai tên cột "Binary" thành "Bin" làm code không nhận diện được). Ngoài ra, kết quả solver phụ thuộc vào độ chính xác số học; đôi khi với mô hình có hệ số rất lớn/nhỏ, nên chuẩn hóa đơn vị hoặc điều chỉnh để tránh sai lệch do làm tròn.

Tổng kết, việc **dùng file Excel làm input và output** tạo ra một vòng lặp người dùng – chương trình tối ưu rất hiệu quả: người dùng cung cấp dữ liệu và mô hình trong môi trường quen thuộc, chương trình Python + CBC xử lý "thầm lặng" phía sau, rồi trả kết quả lại đúng vào môi trường quen thuộc đó. Cách làm này đang trở nên phổ biến trong triển khai OR thực tế, vì nó kết hợp được **trải nghiệm người dùng tốt** với **sức mạnh tính toán** của các thư viện tối ưu hóa.

Tài liệu tham khảo:

1. Sarah Lewis, *"Operations research (OR) is an analytical method of problem-solving and decision-making..."*, **TechTarget** (What is Operations Research and Why is it Important?).
2. *"Modern applications of operations research includes city planning, football strategies, emergency planning, optimizing all facets of industry and economy..."*, **Wikipedia** (Operations research - Wikipedia) (Operations research - Wikipedia).

3. *"Linear programming is a method to achieve the best outcome in a mathematical model whose requirements are represented by linear relationships."*, **Wiki.openmod-initiative** (Linear problem - wiki.openmod-initiative.org).
4. Bethany Nicholson, *"Pyomo is an algebraic modeling language... allows users to easily represent optimization problems at a high-level... Pyomo then provides interfaces to a variety of optimization solvers including Gurobi and CPLEX."*, **OR StackExchange** (cplex - What is the purpose of libraries like Pyomo and Google OR tools? - Operations Research Stack Exchange).
5. Laurent Perron, *"OR-Tools is completely free; it's implemented in C++ from the ground up, so it doesn't call commercial libraries on the backend... you wouldn't need your company to buy CPLEX or other solver."*, **OR StackExchange** (cplex - What is the purpose of libraries like Pyomo and Google OR tools? - Operations Research Stack Exchange).
6. Frontline Solvers, *"The standard Microsoft Excel Solver has a limit of 200 decision variables... and up to 100 constraints on cells which are not decision variables."*, **Solver.com** (vba - Reaching maximum number of solver variables? - Stack Overflow) (Standard Excel Solver - Dealing with Problem Size Limits - Continued | solver).
7. Andrew Mason, *"We have found OpenSolver's performance to be similar or better than Solver's. CBC appears to be a more modern optimizer... For example, large knapsack problems which take hours with Solver are solved instantly using OpenSolver, thanks to newer techniques such as problem strengthening and preprocessing used by CBC."*, **ORSNZ Conf. 2010** () ().
8. Diego, *"I obtain for Cbc a very good performance, Cbc even outperforms some commercial solvers in numerically complex problems."*, **Julia Discourse** (Cbc and Clp performance - Optimization (Mathematical) - Julia Programming Language).
9. Gurobi Optimization, *"Mixed Integer Linear Programming problems are generally solved using a linear-programming based branch-and-bound algorithm."*, **Gurobi Primer** (Mixed-Integer Programming (MIP) – A Primer on the Basics - Gurobi).
10. Coin-OR CBC GitHub, *"Cbc (Coin-or branch and cut) is an open-source mixed integer linear programming solver written in C++. It can be used as a callable library or using a stand-alone executable. It can be used in a wide variety of ways through various modeling systems, packages, etc."*, **GitHub README** (GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver).
11. NEOS Server, *"Cbc utilizes other COIN-OR projects Cgl (Cut Generation Library) to generate cutting planes and Clp to solve the linear programs at each node of the tree."*, **NEOS Guide** (NEOS Server: Cbc).
12. SolverMax, *"OpenSolver offers a range of solvers for use in Excel, including the excellent, open source, COIN-OR CBC optimization engine which can quickly solve large Linear and Integer problems... No artificial limits on the size of problem – as many variables and constraints as memory allows."*, **SolverMax.com** (Solver Max - OpenSolver).
13. Mark L. Stone, *"You don't need Pyomo to interface with Gurobi... Pyomo lets you easily switch among several solvers without having to reenter the model."*, **OR StackExchange** (cplex - What is the purpose of libraries like Pyomo and Google OR tools? - Operations Research Stack Exchange) (về mục đích dùng modeling language).

14. Reddit r/optimization, *"Gurobi is a world class commercial solver and the modeling syntax is ... (and OR-Tools in Python is free for work)."* (What software is used in the field these days? : r/optimization - Reddit) (về độ phổ biến Gurobi & OR-Tools).
15. IBM Community, *"For linear problems, nothing can get even close to CPLEX. It is truly amazing how fast and accurate their LP solver is."*, **Reddit** (Use of commercial solvers in the engineering and simulations ...).