UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Università degli Studi di Padova

School of Engineering Department of Information Engineering

Master Degree in Computer Engineering

Enhancing the ACS Heuristic: Parameter Tuning and Computational Experiments for Solving MIP Problems

Supervisor:
Prof. Domenico Salvagnin

Candidate:
Francesco Biscaccia Carrara
2120934

Academic Year 2024/2025
Date TBD

Abstract

Contents

Chapter 1

Introduction

1.1 Mixed Integer Programming (MIP)

Mixed Integer Programming (MIP) is a powerful optimization technique used to model complex real-world problems where discrete decisions are essential—such as resource allocation, scheduling, and logistics. MIP models aim to find the best solution according to a given objective function, which is either maximized or minimized. A generic mixed-integer program (MIP) will be defined as

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x_i \in \mathbb{Z},\ \forall i \in \mathcal{I} \\ & l \leq x \leq u \end{aligned}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $\mathcal{I} \subseteq \{1 \dots n\}$ is the subset of integer variables indices. The solution vector x is bounded by $l \in \bar{\mathbb{R}}^n$ and $u \in \bar{\mathbb{R}}^n$ where $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$. Here, some decision variables are constrained to take integer values, which increases the model's complexity.

1.1.1 Heuristic in MIP solving

The presence of integer variables significantly increases the computational complexity of MIP problems, especially as the number of variables and constraints grows. Solvers such as IBM ILOG CPLEX[1] and GUROBI[2] incorporate a variety of built-in strategies to improve performance. Among these strategies are heuristics, which are specialized algorithms designed to quickly find feasible (but not necessarily optimal) solutions. Although heuristics do not guarantee optimality, they are computationally efficient and can significantly accelerate the solution process by providing good initial solutions or guiding the search within the solution space.

1.2 Alternating Criteria Search (ACS)

Finding high-quality feasible solutions is a key aspect of the discrete optimization process, and constructing them has been one of the main focuses of research in the last few decades. Primal heuristics differ in whether they require a starting feasible solution or not. In the first case, the heuristics are called starting heuristics, and state-of-the-art techniques rely on Large Neighborhood Search (LNS)[3] to find high-quality feasible solutions. The effectiveness of such strategies relies on the power of the MIP solver to solve the subproblems generated during the LNS heuristic. Improvement heuristics, in contrast, require a feasible starting solution and aim to improve its quality with respect to the objective. The simplest improvements are the 1-opt and 2-opt[4] methods, but—as with starting heuristics—there are many improvement heuristics based on LNS ideas. The neighborhoods explored are defined based on branch-and-bound information—such as the best incumbent and the LP relaxation. This is essentially a limitation in terms of diversification: the strategies may reach good solutions, but it is difficult to find them in the early stages of the search. Therefore, to address this issue, the Alternating Criteria Search (ACS)[5]—or Parallel ACS, its straightforward parallel implementation—can be utilized. The search neighborhoods are defined based on randomization instead of branch-and-bound information, allowing for a wider exploration of the search space. The aim of this strategy is to find high-quality feasible solutions at the beginning of the search, increasing the heuristic's effectiveness.

1.2.1 Parallelization of ACS

Parallelism can be exploited in MIP optimization to accelerate computation and enhance solver performance, in terms of execution time and scalability. The exploration of the branch-and-bound tree in parallel is now incorporated in most state-of-the-art MIP solvers—such as CPLEX or GUROBI—and it essentially consists of solving multiple subproblems of the original MIP simultaneously. However, this strategy may not scale well to a large number of cores[6]. To address this limitation, the proposed heuristic—Parallel ACS (PACS)—is a parallel algorithm that integrates elements of both starting solution and improvement heuristics, offering the capability of generating starting solutions and improving them with respect to the original objective. To leverage parallelism, Parallel ACS performs a large number of LNS runs simultaneously over a diversified set of neighborhoods, with the aim of increasing the chances of finding higher-quality solutions. After each LNS exploration, the strategy consolidates the local improvements through a recombination phase. PACS combines parallelism and diversified LNS in order to address large instances of MIPs arising from various application domains.

1.2.2 Limitations of the PACS Approach

Although the experiments about the PACS strategy bring evidence about the effectiveness on hard MIP instances, there are several considerations that must be addressed. First of all, the PACS algorithm was executed on an 8-node computing cluster, each equipped with two Intel Xeon X5650 6-core processors and 24 GB of memory, totaling 96 cores and 192 GB of RAM. Such computing resources are not typically available in consumer-grade or general-purpose environments. Furthermore, as discussed in the original study, the PACS strategy could assist the MIP solver in accelerating convergence. This suggests that integrating PACS heuristics within a MIP solver could provide a meaningful contribution to solver performance. Second, before the PACS execution, an initial calibration phase is required to tune the parameters prior to execution. This undermines the objective of designing a self-contained, efficient heuristic. With these considerations in mind, the goal of this thesis is to refine the PACS strategy to make it more compatible with MIP solvers. In particular, the aim is to establish a fixed, robust parameter configuration that remains effective regardless of the input instance.

Chapter 2

The ACS Framework

The Alternating Criteria Search (ACS) heuristic is designed to pursue a twofold objective: to identify a feasible solution and to subsequently enhance its quality with respect to the objective function $c^T x$. To this end, ACS employs a Large Neighborhood Search (LNS) strategy, which iteratively solves two auxiliary mixed-integer programming (MIP) subproblems in order to address both objectives. The heuristic requires an initial vector, which is not mandated to be a feasible solution for the original MIP. This vector is progressively refined by solving sub-MIPs in which a subset of variables is fixed to the values of the initial input.

2.1 Feasibility-MIP (FMIP)

In linear programming theory, it is well established that the feasibility problem can be addressed using the two-phase Simplex method, in which an auxiliary optimization problem is solved to obtain a feasible starting basis, and hence a feasible solution. In a similar manner, the following auxiliary MIP problem, denoted as the Feasibility-MIP (FMIP), is formulated to identify a feasible starting solution:

$$
\begin{aligned}
\min \quad & \sum_{i=0}^{m} \Delta_i^+ + \Delta_i^- \\
\text{s.t.} \quad & Ax + I_m\Delta^+ - I_m\Delta^- = b \\
& x_i = \hat{x_i},\ \forall i \in F \\
& l \leq x \leq u \\
& x_i \in \mathbb{Z},\ \forall i \in \mathcal{I} \\
& \Delta^+ \geq 0, \Delta^- \geq 0
\end{aligned}
$$

Here, I_m denotes an $m \times m$ identity matrix, while Δ^+ and Δ^- are vectors in R^m corresponding to the m constraints. These are introduced as slack variables, and the objective is to minimize their sum. Analogous to the two-phase Simplex method, a vector x is feasible for the original MIP if and only if it can be extended to a solution of value 0 for

the associated FMIP. Since solving an FMIP is as computationally demanding as solving the original problem, the neighborhoods are restricted by fixing a given subset F of the integer variables to the values of an input vector $[\hat{x}, \Delta^+, \Delta^-]$. Because of the introduction of slack variables, the vector $\hat{x}$ is not required to be a feasible solution itself, but it must be integral and within the variable bounds to preserve the feasibility of the model. In this way, the FMIP guarantees that feasibility is preserved under any arbitrary variable-fixing scheme. Once a feasible solution is obtained, any LNS-based improving heuristic—such as RINS[7], DINS[8], or local branching[9]—can then be applied to refine the solution.

2.2 Optimality-MIP (OMIP)

Rather than executing the FMIP until convergence to a feasible solution vector, the following auxiliary MIP problem, denoted as Optimality-MIP (OMIP), is designed to improve a partially feasible solution $[\hat{x}, \hat{\Delta}^+, \hat{\Delta}^-]$, which satisfies $\sum_{i=1}^{m}(\hat{\Delta}_i^+ + \hat{\Delta}_i^-) \neq 0$, with respect to the original objective $c^T x$:

$$
\begin{aligned}
\min \quad & c^T x \\
\text{s.t.} \quad & Ax + I_m\Delta^+ - I_m\Delta^- = b \\
& \sum_{i=0}^{m} \Delta_i^+ + \Delta_i^- \leq \sum_{i=0}^{m} \hat{\Delta}_i^+ + \hat{\Delta}_i^- \\
& x_i = \hat{x}_i, \ \forall i \in F \\
& l \leq x \leq u \\
& x_i \in \mathbb{Z} \quad \forall i \in \mathcal{I} \\
& \Delta^+ \geq 0, \ \Delta^- \geq 0
\end{aligned}
$$

Analogous to the FMIP, the OMIP represents a reformulation of the original MIP model in which auxiliary slack variables are introduced for each constraint. This formulation enables the OMIP to enhance a solution even if it is not feasible for the original MIP. Moreover, to ensure that the optimal solution of the OMIP does not exceed the infeasibility of the input solution $\hat{x}$, an additional budget constraint is imposed, limiting the total slack to $\sum_{i=0}^{m} \hat{\Delta}_i^+ + \hat{\Delta}_i^-$.

By iteratively solving subproblems of both auxiliary MIPs, the ACS heuristic is designed to converge—although convergence is not formally guaranteed—to a high-quality feasible solution. By construction, infeasibility decreases monotonically after each iteration. However, the quality of the solution with respect to the original objective function may fluctuate.

2.3 Parallelization of ACS

The parallelization of ACS exploits parallelism by generating a diversified set of large neighborhood searches, which are solved simultaneously. Exploring multiple search neighborhoods in parallel is expected to increase the likelihood of identifying high-quality solutions. Following this parallelization step, the improvements obtained in parallel must be combined efficiently. To this end, an additional search subproblem is generated in which variables with identical values across different solutions are fixed. Consequently, the recombination phase constitutes a crucial step in PACS: by merging the improvements achieved during the parallel phase, the subsequent phase can explore a newly diversified set of large neighborhood searches based on the recombined solution, thereby enhancing the probability of further improvements. The pseudocode[1] illustrates the overall workflow. Each processor generates a set of randomized variable fixings and solves the

Algorithm 1 Parallel Alternating Criteria Search

function FMIP_LNS($\mathcal{F}, \hat{x}$)
  return $\min\{\sum_i \Delta_i^+ + \Delta_i^- \mid Ax + I_m\Delta^+ - I_m\Delta^- = b,\ x_j = \hat{x}_j\ \forall j \in F,\ x_j \in \mathbb{Z}\ \forall j \in \mathcal{I}\}$
end function

function OMIP_LNS($\mathcal{F}, \hat{x}, \hat{\Delta}$)
  return $\min\{c^t x \mid Ax + I_m\Delta^+ - I_m\Delta^- = b,\ \sum_i \Delta_i^+ + \Delta_i^- \leq \hat{\Delta},\ x_j = \hat{x}_j\ \forall j \in F,\ x_j \in \mathbb{Z}\ \forall j \in \mathcal{I}\}$
end function

Ensure: Feasible solution $\hat{x}$ if found
Initialize $[\hat{x}, \Delta^+, \Delta^-]$ as an integer solution
$T := numThreads()$
while $timeElapsed() \leq$ time-Limit **do**
  if $\sum_i \Delta_i^+ + \Delta_i^- > 0$ **then**
    for all threads $t_i \in \{0, T-1\}$ **in parallel do**
      F_{t_i} := randomized variable index subset, $F_{t_i} \subseteq \mathcal{I}$
      $[x^{t_i}, \Delta^{+t_i}, \Delta^{-t_i}]$:= FMIP_LNS($F_{t_i}, \hat{x}$)
    end for
    $U := \{j \in \mathcal{I} | x_j^{t_i} = x_j^{t_k},\ 0 \leq i < k < T\}$
    $[\hat{x}, \Delta^+, \Delta^-]$:= FMIP_LNS(U, x^{t_0})
  end if
  $\Delta^{UB} := \sum_i \Delta_i^+ + \Delta_i^-$
  for all threads $t_i \in \{0, T-1\}$ **in parallel do**
    F_{t_i} := randomized variable index subset, $F_{t_i} \subseteq \mathcal{I}$
    $[x^{t_i}, \Delta^{+t_i}, \Delta^{-t_i}]$:= OMIP_LNS($F_{t_i}, \hat{x}, \Delta^{UB}$)
  end for
  $U := \{j \in \mathcal{I} | x_j^{t_i} = x_j^{t_k},\ 0 \leq i < k < T\}$
  $[\hat{x}, \Delta^+, \Delta^-]$:= OMIP_LNS(U, x^{t_0}, Δ^{UB})
end while
return $[\hat{x}, \Delta^+, \Delta^-]$

associated sub-MIP—either FMIP or OMIP—until the allotted time limit is reached.

Subsequently, the solutions are exchanged, and the set U, containing the indices of variables common across solutions, is constructed. The recombination MIP then consists of a subproblem, again associated with either FMIP or OMIP, in which the variables in U are fixed. The best solution obtained will be either the most feasible or the most optimal, depending on whether a recombination FMIP or OMIP is employed. Moreover, every solution used as input can be incorporated as a MIP start, as it remains feasible under any variable-fixing strategy. Processor synchronization and memory communication are handled via the Message Passing Interface (MPI), owing to its efficient all-to-all collective communication primitives in distributed large-scale architectures.

2.4 Initialization of ACS

As introduced earlier in this chapter, ACS only requires a starting vector that is integer feasible and within the variable bounds. However, a stronger starting point is a solution that is as feasible as possible with respect to the objective function of the FMIP. The proposed algorithm provides a lightweight heuristic that seeks to minimize the infeasibility of the initial solution. This algorithm can be described by the pseudocode[2]. The algorithm first sorts the list of integer variables in order of increasing bound range. It

Algorithm 2 Starting vector heuristic

Require: Percentage of variables to fix θ, $0 < \theta \leq 100$, Fixed bound constant c_b
Ensure: Starting integer-feasible vector $\hat{x}$
$V :=$ list of integer variables sorted by increasing bound range $u - l$
$F := \emptyset$
while $\hat{x}$ is not integer feasible **AND** $\mathcal{F} \neq \mathcal{I}$ **do**
    $\mathcal{K} :=$ top $\theta\%$ of unfixed variables from V
    for $k \in \mathcal{K}$ **do**
        $\hat{x}_k :=$ random integer value between $[\max(l_k, -c_b), \min(u_k, c_b)]$
    end for
    $F := F \cup \mathcal{K}$
    $[x, \Delta^+, \Delta^-] := \min\{\sum_i \Delta_i^+ + \Delta_i^- \mid Ax + I_m\Delta^+ - I_m\Delta^- = b,\ x_j = \hat{x}_j \ \ \forall j \in F\}$
    $Q :=$ index set of integer variables of x with integer value
    $\hat{x}_q = x_q, \ \ \forall q \in Q$
    $F := F \cup Q$
end while
return $\hat{x}$

then fixes the top $\theta\%$ of variables to random integer values within their respective bounds. The input parameter θ controls the trade-off between the difficulty of the LP relaxation and the quality of the resulting starting solution. In cases where the bounds are infinite, a constant value of 10^6 is used to clamp the bounds. The rationale behind the sorting step is to prioritize binary variables first, followed by the remaining integer variables. Until all integer variables are fixed, the LP relaxation of the FMIP is solved to optimize the unfixed variables. Any variables that attain integer values in this process are then fixed.

Since at least $\theta\%$ of the variables are fixed at each iteration, the algorithm is guaranteed to terminate after at most $\lceil 100/\theta \rceil$ iterations.

2.5 Variable Fixing Strategy

Selecting an appropriate variable fixing scheme is a challenging task: an overly restrictive strategy may fail to yield improvements, whereas an excessively loose strategy can lead to a search space that is too large to explore efficiently within a reasonable timespan. The proposed algorithm constitutes a simple yet intuitive variable fixing method. It incorporates randomness to promote diversification and allows for controlling the number of variables to be fixed through an adjustable parameter. The algorithm is described in the pseudocode[3]. The fixings are determined by selecting a random integer variable

Algorithm 3 Variable Fixing Selection Algorithm

Require: Fraction of variables to fix ρ, $0 < \rho < 1$
Ensure: Set of integer indices F
1: **function** RandomFixings(ρ)
2: $i :=$ random element in $\mathcal{I}$
3: $F :=$ first $\rho \cdot |\mathcal{I}|$ consecutive integer variable indices starting from i in a circular fashion
4: **return** F
5: **end function**

x' and fixing a consecutive sequence of integer variables starting from x' up to a cap determined by ρ, an input parameter that specifies the number of variables to be fixed. The fixing is performed in a circular fashion: if the end of the set $\mathcal{I}$ is reached before the required number of variables are fixed, the algorithm continues from the beginning of $\mathcal{I}$. The effectiveness of this strategy relies on the fact that, for many problems—such as network flow and routing—the variables are arranged consecutively, often defining a cohesive substructure within the problem.

Bibliography

[1] IBM Corporation. *IBM ILOG CPLEX Optimization Studio*. Version 22.1. 2024. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio.

[2] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. Version 11.0. 2024. URL: https://www.gurobi.com.

[3] David Pisinger and Stefan Ropke. "Large Neighborhood Search". In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. Boston, MA: Springer US, 2010, pp. 399–419. ISBN: 978-1-4419-1665-5. DOI: 10.1007/978-1-4419-1665-5_13. URL: https://doi.org/10.1007/978-1-4419-1665-5_13.

[4] Tobias Achterberg, Timo Berthold, and Gregor Hendel. "Rounding and Propagation Heuristics for Mixed Integer Programming". In: Jan. 2012.

[5] Lluís-Miquel Munguía et al. "Alternating criteria search: a parallel large neighborhood search algorithm for mixed integer programs". In: *Computational Optimization and Applications* 69.1 (Jan. 2018), pp. 1–24. ISSN: 1573-2894. DOI: 10.1007/s10589-017-9934-5. URL: https://doi.org/10.1007/s10589-017-9934-5.

[6] Thorsten Koch, Ted Ralphs, and Yuji Shinano. "Could we use a million cores to solve an integer program?" In: *Mathematical Methods of Operations Research* 76.1 (Aug. 2012), pp. 67–93. ISSN: 1432-5217. DOI: 10.1007/s00186-012-0390-9. URL: https://doi.org/10.1007/s00186-012-0390-9.

[7] Emilie Danna, Edward Rothberg, and Claude Le Pape. "Exploring relaxation induced neighborhoods to improve MIP solutions". In: *Mathematical Programming* 102.1 (Jan. 2005), pp. 71–90. ISSN: 1436-4646. DOI: 10.1007/s10107-004-0518-7. URL: https://doi.org/10.1007/s10107-004-0518-7.

[8] Shubhashis Ghosh. "DINS, a MIP Improvement Heuristic". In: June 2007, pp. 310–323. ISBN: 978-3-540-72791-0. DOI: 10.1007/978-3-540-72792-7_24.

[9] Matteo Fischetti and Andrea Lodi. "Local branching". In: *Mathematical Programming* 98.1 (Sept. 2003), pp. 23–47. ISSN: 1436-4646. DOI: 10.1007/s10107-003-0395-5. URL: https://doi.org/10.1007/s10107-003-0395-5.