A MIP Approach to the Minimal Seed Set Problem

Alberto Basaglia[1] and Domenico Salvagnin[1 [0000−0002−0232−2244]]

Department of Information Engineering, University of Padova, Via Gradenigo 6/B, 35131 Padova, Italy
{basagliaal,salvagni}@dei.unipd.it

Abstract. The minimal seed set problem is a combinatorial problem, originating in systems biology, defined as follows: given a set of metabolic reactions of an organism, find the smallest set of nutrients that, after applying all possible reactions, can be expanded to the full set of all nutrients that the organism could possibly synthesize. The problem can naturally be casted as a classical planning problem, yet current optimal planners are not able to solve it, unless domain-specific algorithmic improvements are implemented. While it is mentioned in the literature that mixed-integer programming (MIP) approaches do not scale up, no actual MIP formulation is explicitly reported. In this paper we present two MIP formulations for the minimal seed set problem, one based on big-M constraints and the other on a time-indexed expansion. We evaluate the two formulations computationally and show that, while one of the two is indeed performing quite poorly, the other can solve instances of reasonable size in a few seconds on average with an off-the-shelf MIP solver.

Keywords: minimal seed problem · mixed integer programming

1 Introduction

The minimal seed set problem originates in the field of system biology. For the purposes of this problem, an organism is represented by a set of metabolic reactions, each requiring one or more compounds (or nutrients), and producing in turn one or more compounds. More formally, let C be the set of all the compounds associated with an organism. A reaction r is an ordered pair of sets: $r = (X, Y)$. $X \subseteq C$, the first element of the pair, is called the *substrate* of the reaction and is the set of elements that are required to complete it. The second element of the pair, $Y \subseteq C$, is called the *product* of the reaction, and it gives the set of compounds that are produced by the reaction after it is completed. If a reaction is bidirectional (as many are in practice), we model it as two separate reactions, (X, Y) and (Y, X). Note that, at this level of description, we do not deal with the proportions of compounds needed/produced by a reaction (its *stoichiometry*), but rather assume that, if a compound is present, there is always enough of it that a reaction depending on it can be completed. In other words,

reactions do not consume substrate nutrients, but only add product nutrients. The set of all metabolic reactions of an organism will be denoted by R. A set $T \subseteq C$ of compounds is said to be *reachable* from a subset $S \subseteq T$ if S can be expanded to a set containing T by applying applicable reactions, where a reaction $r = (X, Y)$ is applicable to a given set $P \subseteq C$ if its substrate is available, i.e., $X \subseteq P$. A subset $S \subseteq C$ is said to be a *seed set* if the full set C is reachable from S using the reactions in R. The *minimal seed set* consists in finding a seed set of minimum cardinality.

The problem was first introduced in [4], where it is also shown that the problem is NP-hard, via a reduction from the set covering problem. In the same paper, the authors also argue that a MIP approach, while natural, does not scale up to organisms of the size of interest, but unfortunately do not report on the MIP formulations used. They also report that another classical approach, namely the reduction to SAT, is able to solve only the smallest instances, and thus eventually resort to a problem specific heuristic, based on the identification of the strongly connected components on a *flattened* directed graph obtained from the metabolic network of the organism.

The minimal seed set problem was picked up again a few years later in [8], where it was studied under the lens of classical AI planning. Viewing the minimal seed set problem as a planning problem is indeed quite natural: each chemical reaction can be interpreted as an action, where the substrate plays the role of the preconditions, and the product plays the role of the effects. In details, following [8], we can encode the minimal seed set problem as the following planning problem. The set of facts corresponds to the set of compounds C, so we have a state s for each subset of C. For each reaction $r = (X, Y)$ in R we introduce a zero-cost operator o_r, with preconditions $pre(o_r) = X$ and add-effects $add(o_r) = Y$. The operator o_r is applicable in state s if $pre(o_r) \subseteq s$, and the new state after applying o_r is $s' = s \cup add(o_r)$. Then we have a unit-cost operator i_c for each nutrient $c \in C$, that has no preconditions and c as the only add effect. The initial state is the empty set, while the goal state is set full set C. The set of nutrients added by the "insert" operators i_c in a cost optimal plan thus encode a minimal seed set of minimum cardinality.

As already noted in [8], while very natural, the corresponding planning task is quite atypical: many operators have zero cost, the task is delete-free as there are no delete effects, and all propositions must be achieved (so landmarks are not very informative). In addition, as in practice reactions only involve a small subset of compounds, there exist many legal permutations of optimal plans. Because of the properties above, the problem is very challenging for state of the art optimal planners: landmark-based heuristics perform poorly, the branching factor is quite large, and A^* is forced to explore all the permutations of minimal partial plans. In the end, no instance could be solved, back then, by optimal planners, and even *satisficing* ones (like LAMA [14]) only returned uninformative solutions consisting of insert operators only. In order to overcome those issues, the authors in [8] eventually proposed a modified A^* variant that exploits the specific structure of the problem. With this variant they can solve all organisms

in the KEGG database [12], with a running time in the order of 2-3 minutes on standard hardware.

In this paper we want to investigate whether a MIP approach is still performing unsatisfactorily on this problem. Given that no actual formulation is reported in the literature, there is a possibility that the models tested back then were not the best possible ones. In addition, we can also leverage on the steady improvements in MIP solver performance over time, see for example [3]. The paper is organized as follows. In Section 2 we present some problem-specific preprocessing reductions aimed at reducing the size of the instances and making their structure more amenable to the subsequent MIP solving process. In Section 3 we present two MIP formulations for the problem, one based on big-M constraints, and the other based on a time-indexed expansion. In Section 4, we computationally evaluate the two formulations and the impact of the proposed preprocessing reductions, and compare against a more recent optimal planner. Some conclusions are finally drawn in Section 5.

2 Presolve

In the context of MIP solving, a *presolve* reduction is defined as a change to the input problem that preserves at least one optimal solution but that yields a supposedly "simpler" problem to be solved by branch-and-cut. While MIP solvers implement a vast array of (sophisticated) presolve reductions [2], some kind of reductions, usually based on domain knowledge, are beyond the scope of what can be derived automatically by the MIP presolver. In this section we will present a few domain specific presolve reductions for the minimal seed problem.

The first (obvious) observation is that if a compound is not produced by any reaction, then it must necessarily be part of the seed set. This was already noted in [8]. This can easily be done once and for all at the beginning, so in the rest of the paper we will assume that all compounds can be produced by at least one reaction. Then, we consider four reductions that reduce the set of reactions, while preserving the reachability of compounds. The reductions are as follows:

- *Duplicates.* Two reactions are duplicate if they have the same substrate and product. Clearly, we can remove one of the two.
- *Product-domination.* A reaction *product-dominates* another reaction if they have the same substrate but the product of the first contains the product of the second. Any product-dominated reaction can be removed from the problem.
- *Substrate-domination.* A reaction *substrate-dominates* another reaction if they have the same product but the substrate of the first is contained in the substrate of the second. Any substrate-dominated reaction can be removed from the problem.
- *Merging.* Two reactions with the same substrate can be merged into a new reaction that has the same substrate and as product the union of the products. Notice that the merged reaction will by construction product-dominate the reactions it was created from.

We implemented these reductions in our code, and we always run them all, in the order in which they were presented, until a fix-point is reached.

3 MIP Models

Let us consider a minimal seed set problem defined by a set C of compounds and a set R of reactions. Given a reaction $r \in R$, we will denote with $X_r \subseteq C$ its substrate and with $Y_r \subseteq C$ its product. The basic encoding of the minimal seed set is the same in all the formulations that we are going to present: in particular, we will use a set of binary variables x_c to encode whether the compound c is part of the seed set or not. A MIP formulation of the problem also needs to consider the order in which reactions are applied, as we can have circular dependencies between the substrates and products of different reactions. Consider for example a simple organism with two compounds, A and B, and two reactions, $r_1 : A \leftrightarrow B$ and $r_2 : B \leftrightarrow A$: if we didn't consider the reaction order into account, we might conclude that both A and B can be obtained as products and add neither of them to the minimal seed set, while clearly we need at least one of them. We will thus assume a discretized time horizon $|T|$, and the set of time instants $T = \{0, \ldots, |T|\}$. While in general the planning horizon for a planning problem is not polynomially bounded, being the minimal seed set problem a delete-free planning task we always get a polynomial upper bound: in particular, a simple argument shows that a time horizon of $|T| = \min\{|C|, |R|\}$ is always sufficient to apply all the reactions needed by the seed set, as at any time instant before reaching the full set C, we will apply at least one reaction adding at least one compound to the set of available compounds.

3.1 Big-M formulation

The first formulation assigns to each compound $c \in C$ an integer variable $t_c \in T$, encoding the earliest time instant at which the compound becomes available. In addition, we keep track of which reaction is *responsible* for producing a given compound, if any: this is encoded by a set of binary variables u_{rc}, for each $r \in R$ and for each $c \in Y_r$. The resulting MIP model reads:

$$\min \sum_{c}^{C} x_c \tag{1}$$

$$x_c + \sum_{r:c \in Y_r} u_{rc} \geq 1 \qquad \forall c \in C \tag{2}$$

$$t_d \geq t_c + 1 - (|T| + 1)[x_d + (1 - u_{rd})] \qquad \forall r \in R, \forall d \in Y_r, \forall c \in X_r \tag{3}$$

$$0 \leq t_c \leq |T| \qquad \forall c \in C \tag{4}$$

$$x_c \in \{0, 1\} \qquad \forall c \in C \tag{5}$$

$$u_{rc} \in \{0, 1\} \qquad \forall r \in R, \forall c \in Y_r \tag{6}$$

$$t_c \in \mathbb{Z}_+ \qquad \forall c \in C \tag{7}$$

The objective (1) enforces the minimal cardinality of the seed set. Constraints (2) make sure that a compound is either added to the seed set or produced by a reaction that has it in its product. Constraints (3) enforce that a compound d produced by a reaction r becomes available after the compounds $c \in X_r$. The constant $|T| + 1$ plays the role of a big-M coefficient in this formulation: the constraint gets deactivated if the compound d is part of the seed set or if the reaction r is not used to produce d. Finally, constraints (4) enforce the time instants to be within the time horizon.

3.2 Time-Indexed Formulation

A big-M formulation usually gives a poor linear programming relaxation, and thus an inefficient solving process. A possible way to get rid of the big-M constraints is to perform a so-called time-indexed expansion, a standard modeling tool in many scheduling/planning domains. This is exactly what we propose for our second formulation. We still use the binary variables x_c to encode the seed set, but, instead of the variables u_{rc} and t_c, we introduce two sets of time-indexed binary variables. The variables in the first set, $d_{c,t}$, encode whether a compound c is available at time t, while the variables in the second set, $a_{r,t}$, encode whether a reaction r is applicable at time t. The MIP models becomes:

$$\min \sum_{c}^{C} x_c \tag{8}$$

$$d_{c,t} \leq d_{c,t-1} + \sum_{r:c \in Y_r} a_{r,t-1} \qquad \forall c \in C, \forall t \in T \setminus \{0\} \tag{9}$$

$$a_{r,t} \leq d_{c,t} \qquad \forall r \in R, \forall c \in X_r, \forall t \in T \tag{10}$$

$$d_{c,0} = x_c \qquad \forall c \in C \tag{11}$$

$$d_{c,|T|} = 1 \qquad \forall c \in C \tag{12}$$

$$x_c \in \{0,1\} \qquad \forall c \in C \tag{13}$$

$$d_{c,t} \in \{0,1\} \qquad \forall c \in C, \forall t \in T \tag{14}$$

$$a_{r,t} \in \{0,1\} \qquad \forall r \in R, \forall t \in T \tag{15}$$

Constraints (9) encode the fact that a given compound c is available at time t only if it already available at time $t-1$ or is produced by some reaction at time $t-1$. Constraints (10) enforce that a reaction is applied at time t only if all the compounds in its substrate are available. Finally, constraints (11) enforce the availability of a compound in the seed set directly at time 0, while constraints (12) make sure that all compounds are available at the end.

4 Computational Results

We evaluated our approach on the same set of instances used by Gefen and Brafman in [8], and that are publicly available in the GitHub repository [7],

in the form of *PDDL* files. All instances are taken from the KEGG pathways database [12]. The size of the instances, as number of nutritients/reactions, is reported in [8], and is in the order of $2000 - -4000$.

The PDDL files, describing a classical AI planning encoding of the problem, were fed directly to the state of the art planner *FastDownward* [9], using A^* search and the *LMcut* heuristic [10]. As for the MIP models, we wrote a Rust [15] program that reads in input a PDDL file and generates as output a MIP model in LP format, using a dedicated package (crate) [6]. The resulting instances are then solved using a black box MIP solver, namely IBM ILOG CPLEX 22.1.0 [11].

All solvers were run on a cluster of 24 identical machines, each equipped with an Intel Xeon CPU E3-1220 V2 CPU running at 3.10 GHz, and 16 GB of RAM. Each method was run on each instance with a time limit of 1 hour. For MIP solvers, in order to mitigate the effect of performance variability [5, 13], we run each instance 5 times with 5 different random seeds. Except for the random seed and time limit, the MIP solver was run with default settings[1]

4.1 Classical Planning Approach

In our first set of experiments we tested a state of art AI planner, namely *FastDownward*, to check whether the out of the box performance of the planning approach had improved in the last decade. Unfortunately, we can confirm that, without the dedicated algorithmic changes described in [8], this domain is still very challenging for classical planning: *FastDownward* could not solve any of the 22 instances within the 1 hour time limit.

4.2 MIP Approach

In a second set of experiments, we tested our two MIP formulations, namely the big-M formulation and the time-indexed formulation, on the original models, without any of the preprocessing reductions described in the previous section. In the following, we will count each organism-seed pair as one instance.

Aggregated results are given in Table 1. The format of this table, and subsequent ones, is as follows. For each method under comparison, we report number of solved instances, running time and nodes. For the reference method we give absolute numbers for all measures, while for the other one we report the ratios of runtime and nodes (the number of solved instances is still reported in absolute terms), and the number of consistent wins/losses w.r.t. the reference method. An instance is counted as a consistent win (resp. loss) if it is faster than the reference by at least 10% and for at least 3 random seeds out of 5. Each row of the table corresponds to a (sub)set of the models in a given class. The first row (`all`) shows the results for the entire testbed, while the other rows report

[1] We note that while *FastDownward* is a sequential solver, the MIP solver runs deterministic parallel branch-and-cut by default. On our hardware, forcing a sequential MIP solve incurs a slowdown, on average, of a factor of 2: we will see that this doesn't change the relative comparison between two methods.

results for the instances in bracketed subsets. Each bracketed subset is of the form $[t, 3600\}$, and contains all instances that could be solved by at least one formulation, and where the slowest formulation took at least t seconds to solve the model or timed out. The number of instances in each class is reported in the column `N`. The bracketing convention is used to show how the speed of different formulation as the difficulty of the models increase. Time and node results use a shifted geometric mean [1] with shift values of 10 seconds and 100 nodes, respectively.

		Big-M			Time-Indexed			
Class	N	Solved	Time	Nodes	Solved	Time	Nodes	W/L
all	107	107	13.02	9404	5	217.83	6.23	0/21
$[0, 3600\}$	107	107	13.02	9404	5	217.83	6.23	0/21
$[10, 3600\}$	106	106	13.20	9651	4	225.43	6.30	0/21
$[100, 3600\}$	102	102	13.96	10630	0	257.89	6.58	0/20

Table 1. Computational results of the two MIP formulations without preprocessing.

According to Table 1, the big-M formulation clearly outperforms the time-indexed formulation: after removing 3 instances where the latter ran out of memory, we see that the big-M formulation could solve all the remaining 107 instances, while the other formulation only 5. In addition, it is two-orders of magnitude faster as far as running time is concerned. The consistent wins/losses count is, unsurprisingly, also strongly favourable to the big-M formulation. More importantly, the big-M formulation also performs very well in absolute terms: the average run time is in the order of 13 seconds, and compares very favourably to the domain-specific approach in [8], where the average runtime was in the order of 2-3 minutes (albeit on a different, and somewhat older, hardware).

A more detailed analysis of the MIP runs also provides the following insights on the different behaviour of the two formulations. For the bigM formulation, the average primal gap at the end of the root node is approximately 18.5%, while the average dual gap is 1.3%: so the dual bound is quite strong, while the primal one is weak but is closed relatively quickly by enumeration. Also, we notice that root cutting is quite effective in improving the dual bound. As for the time-indexed formulation, the average primal gap at the end of the root is 19% (thus, not so different) but the average dual gap is a much weaker 11.4%. Also, even after a significant (and very expensive, with an average runtime of more than 10 minutes) MIP presolve, the resulting formulation is two orders of magnitude larger on average, with approximately 700.000 nonzeros vs 6.000 nonzeros. The weaker dual bound, combined with a significantly smaller node throughput because of sheer model size, explains the inefficacy of the time-indexed formulation on this problem class.

4.3 Effect of Preprocessing

Finally we tested the effect of the domain-specific preprocessing reductions described in the previous section, again on the two formulations. First of all we notice that the effect on the number of reactions is significant, with an average reduction of 43%. As far as the performance of resulting formulations, aggregated results are given in Table 2.

		Big-M			Time-Indexed			
Class	N	Solved	Time	Nodes	Solved	Time	Nodes	W/L
all	110	110	8.97	5830	3	348.62	15.28	0/22
$[0, 3600\}$	110	110	8.97	5830	3	348.62	15.28	0/22
$[10, 3600\}$	108	108	9.19	6111	1	374.45	15.77	0/21
$[100, 3600\}$	107	107	9.30	6277	0	386.94	15.98	0/21

Table 2. Computational results of the two MIP formulations with preprocessing.

According to Table 2, it is clear that preprocessing does not change the relative ranking of the two formulations, with the big-M formulation significantly outperforming the time-indexed one. The effect is still beneficial overall, with the first formulation being faster, and the second one no longer running out of memory on 3 instances (the number of solved instances drops to 3, though).

		Without Preprocessing			With Preprocessing			
Class	N	Solved	Time	Nodes	Solved	Time	Nodes	W/L
all	110	110	14.16	10301	110	0.63	0.57	11/0
$[0, 3600\}$	110	110	14.16	10301	110	0.63	0.57	11/0
$[1, 3600\}$	95	95	17.70	20585	95	0.62	0.55	11/0
$[10, 3600\}$	62	62	28.16	41248	62	0.57	0.47	8/0
$[100, 3600\}$	7	7	261.42	544869	7	0.42	0.40	0/0
$[1000, 3600\}$	2	2	1727.69	4881056	2	0.07	0.06	0/0

Table 3. Effect of preprocessing on the big-M formulation.

The overall effect of the preprocessing reductions is more clear when comparing its effect directly on the best formulation, the big-M one, as presented in Table 3. According to the table, preprocessing reduces the average runtime by 37% overall, and almost halves the number of nodes enumerated. It also yields a consistent win on 11 out of 22 organisms, with no consistent loss. The positive effect seems to increase as the instances become more challenging, but the testset is quite small, and does not allow to draw statistically significant conclusions on that regard.

Finally, we note that our domain-specific reductions are not entirely within the scope of the generic MIP presolver. While CPLEX is able to significantly reduce the size of the models even after our problem-specific reductions (approximately by a factor of 2), it cannot fully compensate for them: after CPLEX's presolve, the presolve model is on average 10% smaller (on all counts: rows, cols, nonzeros) when our reductions are applied w.r.t. when they are not.

5 Conclusion

The contribution of the paper is twofold: we introduced two MIP formulations for the minimal seed set problem and we described several preprocessing reductions to reduce the size of the problem while maintaining the same set of optimal solutions.

Of the two formulations, the big-M one proved to perform very well on a testset from the literature, being able to consistently solve all instances with an average running time of less than 15 seconds, on a standard machine and with an off-the-shelf MIP solver. The proposed presolve reductions further improve the picture, reducing the number of reactions by more than 40% and the overall runtime by almost as much.

Overall, we can conclude that, with the right formulation, the minimal seed set problem can be solved satisfactorily with current MIP technology.

References

1. Achterberg, T.: Constraint Integer Programming. Ph.D. thesis, Technische Universität Berlin (2007)
2. Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS Journal on Computing **32**, 473–506 (2016)
3. Achterberg, T., Wunderling, R.: Mixed integer programming: Analyzing 12 years of progress. Facets of Combinatorial Optimization: Festschrift for Martin Grötschel pp. 449–481 (2013)
4. Borenstein, E., Kupiec, M., Feldman, M., Ruppin, E.: Large-scale reconstruction and phylogenetic analysis of metabolic environments. Proceedings of the National Academy of Sciences of the United States of America **105**, 14482–7 (10 2008). https://doi.org/10.1073/pnas.0806162105
5. Danna, E.: Performance variability in mixed integer programming. Presentation slides from MIP 2008 workshop in New York City. http://coral.ie.lehigh.edu/ jeff/mip-2008/program.pdf (2008)
6. Developers, T.R.O.: Rust Linear Programming Modeler. https://github.com/rust-or/rust-lp-modeler (2023), accessed: April 3, 2023
7. Gefen, A.: Minimal seed set. https://github.com/gefena/minimal_seed_set (2023), accessed: April 3, 2023
8. Gefen, A., Brafman, R.: The minimal seed set problem. Proceedings of the International Conference on Automated Planning and Scheduling **21**(1), 319–322 (Mar 2011). https://doi.org/10.1609/icaps.v21i1.13485, https://ojs.aaai.org/index.php/ICAPS/article/view/13485
9. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research **26**, 191–246 (2006)
10. Helmert, M., Domshlak, C.: Landmarks, critical paths and abstractions: What's the difference anyway? In: Gerevini, A., Howe, A.E., Cesta, A., Refanidis, I. (eds.) ICAPS. AAAI (2009)
11. IBM Corporation: IBM ILOG CPLEX Optimization Studio Documentation. IBM Corporation, Armonk, NY, 22.1.0 edn. (2023), https://www.ibm.com/docs/en/icos/22.1.0
12. Kyoto Encyclopedia of Genes and Genomes (KEGG): Kegg pathway. https://www.kegg.jp/ (2023), accessed: April 3, 2023
13. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. In: Theory driven by influential applications, pp. 1–12. INFORMS (2013)
14. Richter, S., Westphal, M.: The LAMA planner: Guiding cost-based anytime planning with landmarks. Journal of Artificial Intelligence Research **39**, 127–177 (2010)
15. The Rust Project Developers: The rust programming language. https://www.rust-lang.org (2023), accessed: April 3, 2023