Graph Reductions in the Tool Replacement Problem with Tool-Insertion Costs

Mikhail Cherniavskii[b], Oleg Lomachenko[1]

[a]*Department of Discrete Mathematics, MIPT, Dolgoprudny, Russia*
[b]*Department of Discrete Mathematics, MIPT, Dolgoprudny, Russia*

Abstract

The tool replacement problem (TRP) is a well-known combinatorial optimization problem in manufacturing systems with limited tool magazine capacity. In its general form, the cost of a switch may depend on both the removed and the inserted tool. We focus on a practically relevant special case where the replacement cost depends only on the tool being inserted.

We show that this assumption allows a structural simplification of the underlying flow-based model. In particular, we prove that there always exists an optimal solution in which each tool either remains in the magazine until its next required usage, or is replaced immediately after a use. This leads to a more compact graph structure and improved complexity. We present the theoretical justification, discuss algorithmic implications, and validate our claims with computational experiments.

Keywords: Tool replacement problem, weighted tool replacement (WTRP), combinatorial optimization, min-cost flow, graph simplification, manufacturing systems

1. Introduction

The tool replacement problem (TRP) is central in the optimization of flexible manufacturing systems. [1] A sequence of jobs must be processed on a machine with limited magazine capacity, and each job requires a specific subset of tools. Since the magazine cannot hold all tools simultaneously, some tools must be replaced over time, and each replacement incurs a cost.

Graph-based formulations using min-cost max-flow (MCMF) are a powerful modeling technique for TRP and its variants. [2] However, in the general case, graph construction involves quadratic complexity in the number of tool requirements, due to the necessity of representing all possible transitions between tools. [2]

In practice, tools have different setup/cleaning times; hence the *weighted TRP (WTRP)* is a natural generalization of the classical (unweighted) TRP: different tools carry different installation (or removal) costs. In some industries the cost may even be *sequence-dependent*; e.g., in printing, changing from one shade of green to another can be cheap, while changing from black to green may require lengthy cleaning and is much more expensive. In this paper we study a widely used yet structured special case of WTRP where the cost depends only on the *inserted* tool; we show that this structure leads to strong graph reductions and faster solvers.

2. Problem formulation

2.1. Definition

We are given:

- M tools indexed by $t = 1, \dots, M$;
- a sequence of N jobs $J_1, \dots, J_N$, where each job J_r requires a subset $T_r \subseteq \{1, \dots, M\}$;
- a tool magazine of capacity $C < M$ (otherwise the problem is trivial, since all tools can be loaded at once and no replacements are needed).

A feasible schedule is a sequence of magazine states $X_1, \dots, X_N$ such that $T_r \subseteq X_r$ and $|X_r| \leq C$ for all $r = 1, \dots, N$. Transitioning from X_r to X_{r+1} may require replacing some tools.

General cost model.. Switching costs are given by a matrix $D = (D_{ij})$, where D_{ij} is the cost of replacing tool i by tool j. [2]

Restricted cost model (insertion-dependent).. We assume the replacement cost depends only on the *inserted* tool and is zero if a tool remains in place:

$$D_{ij} = \begin{cases} c_j, & i \neq j, \\ 0, & i = j, \end{cases} \qquad c_j \geq 0 \text{ for all } j \in \{1, \dots, M\}.$$

Hence, the cost of a switch is independent of the removed tool and fully determined by the tool being introduced into the magazine.

Remark (WTRP variants).. The weighted tool replacement problem can be formulated either with costs depending only on the *inserted* tool, or with costs depending only on the *removed* tool. All theoretical results remain valid under both formulations. In what follows we focus on the insertion-dependent model, but we will add comments on how the statements translate to the removal-dependent case.

2.2. LP for WTRP

Crama proved that WTRP has a totally unimodular constraint matrix. [1]

$$\boxed{\begin{array}{l} \text{Primal} \\ (i)\ x_e \geq 0\ \forall e \in E \\ (ii)\ \sum_{e:e\sim v} x_e \leq F_v \forall v \\ (iii)\ x_e \leq 1\ \forall e \in E \\ s^\top x \to \max \end{array}}$$

Theorem 2.1. *Constraints of LP has totally unimodular matrix.*

Consider the following graph $G = ([n], E)$, where $E = \{(start, end) : (start, end, tool)\}$

$\min\{s^\top x : \sum_{e:v\in e} x_e \leq F_v \ \ \forall v \in V, x \in \{0,1\}^E\}$.

Let $A \in \{0,1\}^{n\times|E|}$ be the incidence matrix of G. Note that since G is graph of intervals, all columns of B have consecutive ones, i.e. of the form $(\underbrace{\ldots}_{zeros}|\underbrace{\ldots}_{ones}|\underbrace{\ldots}_{zeros})$, then B is totally unimodular. Let $I \in \{0,1\}^{|E|\times|E|}$ be id matrix. Let system $A \leq b$ be obtained taking union of three systems systems $B \leq D$ and $I \leq \mathbf{1}$ and $-I \leq \mathbf{0}$. Note that A is totally unimodular, b is integral, then polyhedron $\{s^\top x : A \leq b, x \in \mathbb{R}^{\mathbb{E}}\}$ is integral. Finally, any basic optimal solution of LP $\min\{s^\top x : Ax \leq b, x \in \mathbb{R}^E\}$ is also optimal solution of ILP $\min\{s^\top x : Ax \leq b, x \in \mathbb{Z}^E\}$.

3. Structural simplifications

Lemma 3.1 (Just-in-time removal)**.** *There exists an optimal schedule in which every tool, if removed, is removed* immediately after a job that requires it. *Equivalently, each presence interval of a tool in the magazine is a contiguous block that starts right before some required job and ends right after a (possibly later) required job of the same tool; no tool is ever removed before its next use.*

Proof. Consider any feasible schedule and a tool t that is removed strictly before its next required job $r^\star$. Since t must be available at $r^\star$, it will have to be reinserted before $r^\star$, incurring cost c_t at that later insertion.

We construct a schedule with no larger cost by deferring this premature removal: keep t in the magazine until immediately after $r^\star$. Whenever a capacity conflict arises in the interim (some other tool u must be inserted), we free capacity by evicting a different non-required tool (at that job) instead of t. This swap is always possible because feasibility only requires that the current job's required set be contained in the magazine; t is part of that set exactly at $r^\star$ and not earlier. Crucially, the multiset of *inserted* tools and their insertion times can remain unchanged; only the identities of evicted tools at those times are altered. Since the insertion cost depends solely on the inserted tool and not on which tool is evicted, the total cost does not increase. Repeating this exchange for all premature removals yields a schedule satisfying the claim without increasing cost. $\square$

Consequences for the graph.. Lemma 3.1 implies that the magazine residency of each tool is a concatenation of maximal contiguous blocks bounded by its uses. Hence, the flow network can be simplified as follows:

- **Carry-over arcs.** For each required pair (r, t), add a zero-cost arc to the *next* occurrence of the same tool t. These arcs encode that t remains on the machine until its next demand. They can be constructed in $O(R)$ by a single backward scan over requirements, where $R = \sum_r |T_r|$.

- **Insertion arcs.** From each magazine node associated with the boundary between jobs r and $r{+}1$, add arcs to each $(r{+}1, t)$ with cost c_t, representing just-in-time insertion of t. Self-transitions ($i = j$) are free by definition and need not be modeled explicitly.

  Remark (Removal-dependent costs). If the replacement cost depends on the *removed* tool rather than the inserted one, the weights should be assigned to the arcs *entering the box nodes* instead of those leaving them. Specifically, at the boundary $r \to r{+}1$, the cost c_t is attached to arcs that return the flow from requirement node (r, t) (after its last usage before $r{+}1$) to the box node $\texttt{BOX}_r$. Carry-over arcs remain zero-cost, box-to-box arcs ($\texttt{BOX}_r \to \texttt{BOX}_{r+1}$) stay zero-cost with capacity C, and the mandatory $-K$ arcs are unchanged. Thus the graph structure and complexity bounds are preserved; only the location of the cost assignment differs.

This reduction eliminates the quadratic cross-tool connectivity of the general model: the resulting graph has $O(R)$ carry-over arcs plus $O(R)$ capacity-propagation arcs, yielding a linear-size construction in R for the requirement-dependent part of the network.

4. Graph structure and algorithmic implications

4.1. Classical graph

In the general case, arcs are needed from each (r, t) to (s, u) for all later job–tool pairs, producing $O(R^2)$ arcs, where $R = \sum_r |T_r|$. This dense connectivity arises because every possible replacement must be explicitly represented, and the cost of each transition depends on both the removed and the inserted tool. [2]

4.2. Simplified graph

Using Lemma 3.1, the structure of the flow network can be simplified drastically:

- **Carry-over arcs.** For each required pair (r, t), add a zero-cost arc to the next occurrence of the same tool t. These arcs encode that t remains in the magazine until its next demand. They can be constructed in $O(R)$ time by a single backward scan, where $R = \sum_r |T_r|$.
- **Insertion arcs.** From each magazine node associated with the boundary between jobs r and $r{+}1$, add arcs to each $(r{+}1, t)$ with cost c_t, representing just-in-time insertion of t. Self-transitions ($i{=}j$) are free by definition and need not be modeled explicitly.
- **Box-to-box arcs.** To capture the possibility of modifying the magazine even when the next requirement set is a subset of the previous one, connect consecutive magazine nodes ($\texttt{box}_r$ to $\texttt{box}_{r+1}$) with arcs of capacity C and zero cost. These arcs transfer magazine capacity across job boundaries and allow immediate removal of superfluous tools.
- **Mandatory-request arcs ($-K$ edges).** To enforce that every required pair (r, t) is served exactly when requested, we apply node splitting: each requirement node $v_{r,t}$ is replaced by two nodes $v_{r,t}^{\text{in}}$ and $v_{r,t}^{\text{out}}$ connected by a directed edge of capacity 1 and cost $-K$. All incoming arcs are redirected to $v_{r,t}^{\text{in}}$, and all outgoing arcs originate from $v_{r,t}^{\text{out}}$.

The value of K must be chosen large enough in absolute terms so that any feasible flow bypassing such an edge would be strictly more expensive. A safe choice is to take the maximum tool replacement cost and multiply it by the total number of arcs in the network.

4.3. Example

Figure 1 illustrates the simplified flow network on a toy instance with three consecutive jobs:

$$J_1 = \{T1, T2, T3\}, \qquad J_2 = \{T6, T3\}, \qquad J_3 = \{T1, T7, T3\}.$$

The network uses a *source* S, two magazine nodes (`BOX1` between $J_1 \to J_2$ and `BOX2` between $J_2 \to J_3$), and a *sink* T. Arcs are interpreted as follows:

- **Carry-over arcs (cost 0).** Connect consecutive occurrences of the same tool $(r, t) \to (r', t)$ and represent that tool t stays in the magazine until its next use (e.g., $T1$ from J_1 to J_3, $T3$ from J_1 to J_2 and further to J_3).

- **Insertion arcs (cost c_t).** From each box node at boundary $r \to r{+}1$ to the requirements of the next job $(r{+}1, t)$; they model just-in-time insertion of tool t with insertion-dependent cost.

- **Box-to-box arcs (capacity C, cost 0).** From `BOX1` to `BOX2`, they transfer the magazine capacity across jobs and allow immediate removal of superfluous tools (e.g., when $T_{r+1} \subseteq T_r$).

Self-transitions ($i{=}j$) are free by definition and need not be modeled explicitly.

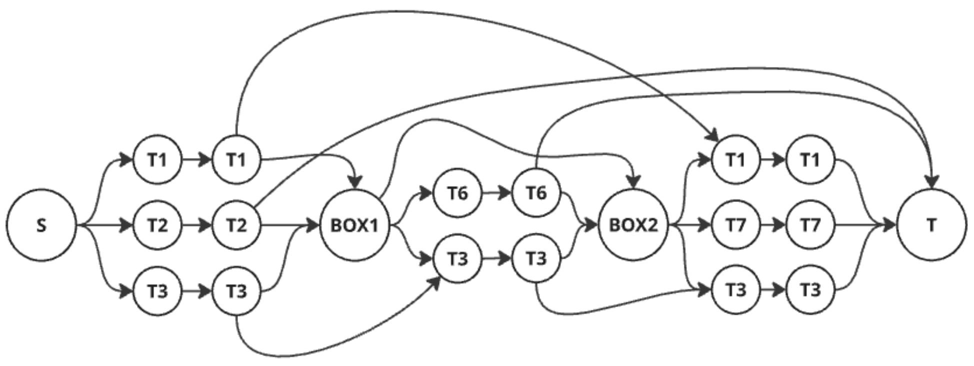

Figure 1: Example of the simplified flow network with source S, box nodes (`BOX1`, `BOX2`), insertion arcs (cost c_t), carry-over arcs (cost 0), and sink T. The three jobs require $\{T1, T2, T3\}$, then $\{T6, T3\}$, then $\{T1, T7, T3\}$.

4.4. Solver setup and complexity

The resulting network is a directed acyclic graph (DAG): all arcs follow the job order. We solve the min-cost flow of value C as follows:

1. one *DAG shortest path* in $O(|V|+|E|)$ for the first unit of flow, and
2. $C-1$ iterations of the *successive shortest path* algorithm with Johnson potentials; each iteration runs Dijkstra in $O(|E| \log |V|)$.

Since the network has $O(R)$ carry-over arcs, $O(R)$ insertion arcs, and $O(N)$ box-to-box arcs, the overall complexity is

$$O(|V|+|E|) \ + \ (C-1)\, O(|E| \log |V|) \ = \ O(R \cdot C \log R),$$

which is substantially smaller than solving on the dense $O(R^2)$ network of the general model.

5. Computational experiments

5.1. Experimental Setup

All experiments were executed on an Apple MacBook Air (M1, 2020), featuring an 8-core ARM64 processor, 16 GB RAM, and a 512 GB SSD, running macOS Ventura 13.2.1.

5.2. Data.

The benchmark instances are based on the classical dataset of Crama, which provides tool–job incidence matrices with specified magazine capacity. [1] Each instance is characterized by the number of tools M, the number of jobs N, and the magazine capacity C, together with a binary $N \times M$ requirement matrix. In our preprocessing step, each row of this matrix is transformed into a tool set T_r for job J_r.

To generate instances for the insertion-dependent cost model, we applied the following randomized procedure:

1. **Random permutation of jobs.** The order of jobs in each Crama instance is randomized once by applying a uniform random permutation over the set $\{1, \dots, N\}$. This preserves the job–tool structure while eliminating potential artifacts due to the original ordering.
2. **Random assignment of insertion costs.** For each tool $t \in \{1, \dots, M\}$, an insertion cost c_t is drawn independently from the discrete uniform distribution
$$c_t \sim \mathcal{U}\{1, 2, 3, 4, 5\}.$$
These values remain fixed for the entire instance and determine the switching-cost matrix via
$$D_{ij} = \begin{cases} c_j, & i \neq j, \\ 0, & i = j. \end{cases}$$

Thus, randomness enters the generation process in two ways: through the permutation of job order and through independent uniform sampling of insertion costs. Both randomization steps were executed once per instance with hard-coded pseudorandom seeds, ensuring that all experiments reported in this paper are exactly reproducible.

5.3. Protocol.

For every common test case present in both pipelines ("original" general-graph code and "our" simplified-graph code), we executed a single end-to-end run and logged total cost and CPU time. Times were measured wall-clock, single-threaded, with `-O3`.

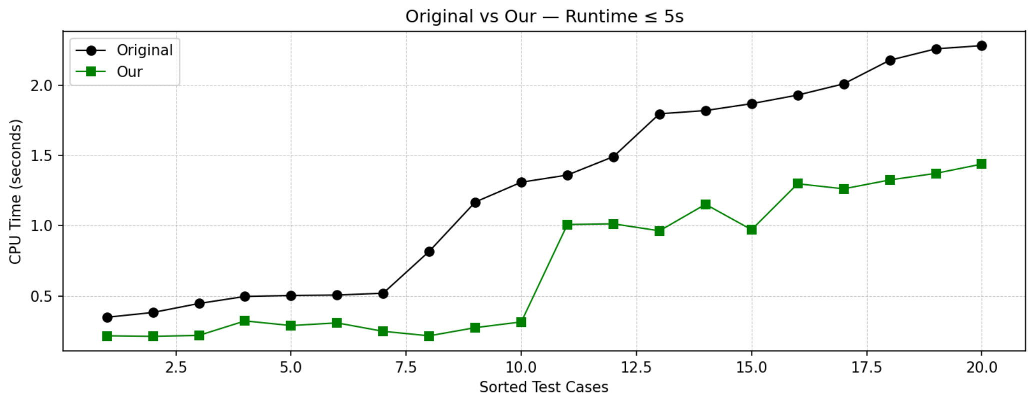

Figure 2: Original (general model) vs. Our (simplified model): instances with runtime $\leq$ 5s.

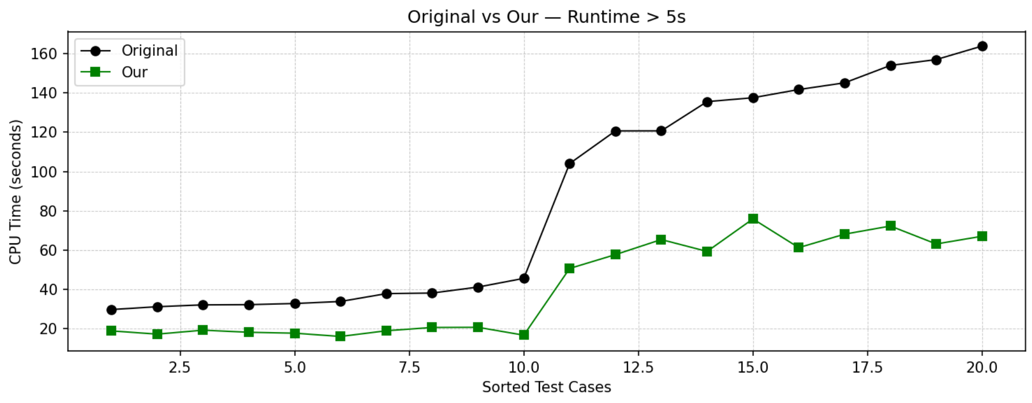

Figure 3: Original (general model) vs. Our (simplified model): instances with runtime $>$ 5s.

Aggregate view..

Table 1: Comparison between the general and simplified models on common test instances. Times in seconds.

#	N	M	C	Opt. value	CPU time (s)	
					Original	Our
1	10	10	7	10	1.309	0.315
2	10	10	7	15	1.168	0.274
3	10	10	7	8	0.817	0.216
4	10	10	7	9	0.446	0.219
5	10	10	7	10	0.519	0.248
6	10	10	7	10	0.506	0.308
7	10	10	7	15	0.503	0.289
8	10	10	7	6	0.348	0.216
9	10	10	7	10	0.382	0.212
10	10	10	7	12	0.496	0.322
11	20	15	12	15	1.929	1.298
12	20	15	12	6	1.360	1.008
13	20	15	12	8	2.258	1.372
14	20	15	12	9	2.009	1.262
15	20	15	12	4	1.491	1.013
16	20	15	12	8	1.819	1.152
17	20	15	12	11	2.281	1.438
18	20	15	12	8	1.868	0.971
19	20	15	12	12	2.177	1.325
20	20	15	12	9	1.796	0.963
21	40	30	25	34	45.665	16.782
22	40	30	25	48	41.243	20.786
23	40	30	25	29	38.200	20.703
24	40	30	25	40	31.250	17.315
25	40	30	25	35	32.304	18.259
26	40	30	25	30	33.912	16.129
27	40	30	25	27	37.904	19.027
28	40	30	25	22	29.809	18.970
29	40	30	25	33	32.880	17.801
30	40	30	25	29	32.199	19.323
31	60	40	30	92	156.915	63.160
32	60	40	30	113	153.922	72.283
33	60	40	30	86	104.039	50.702
34	60	40	30	96	137.503	75.893
35	60	40	30	105	135.534	59.366
36	60	40	30	97	120.652	57.745
37	60	40	30	125	163.787	67.084
38	60	40	30	72	141.685	61.296
39	60	40	30	86	120.672	65.403
40	60	40	30	95	145.091	68.144

Detailed table..

6. Conclusion

We considered the tool replacement problem under tool-dependent replacement costs, i.e., $D_{ij} = c_j$ for $i \neq j$ and $D_{ii} = 0$. We proved a structural property guaranteeing that an optimal policy never removes a tool before its next use; a tool is either replaced immediately after a use or kept exactly until its next demand. This yields a reduced flow network with $O(R+N \cdot C)$ arcs instead of $O(R^2)$ and enables a solver based on one DAG shortest path plus $C-1$ Dijkstra iterations with potentials.

Our computational study on Crama-derived instances with randomly generated insertion-dependent costs confirms the practical impact of this reduction. [1] The simplified graph attains the same optimal values while consistently lowering preprocessing and solving times across the board, with especially large gains on the harder instances illustrated in Figures 2–3. Future work will extend the reduction to sequence-dependent settings and integrate it into large-scale metaheuristics.

References

[1] Crama, Y., Kolen, A. W. J., Oerlemans, A. G., Spieksma, F. C. R. (1991). Minimizing the Number of Tool Switches on a Flexible Machine. Research Memorandum 91.010, Limburg University.

[2] Privault, C., Finke, G. (1995). Modeling a tool switching problem on a single NC-machine. *Journal of Intelligent Manufacturing*, 6, 87–94.