Online Supplement A. Theoretical results of Arslan et al. (2024)

The first result states that the subproblems to price the circuits reduce to an LPPLC. We remark that a non-elementary circuit is composed of at least two elementary circuits. Hence, if the solution of the LPPLC returns a positive reduced cost non-elementary circuit when solving problem [PP-C](i), $i \in \mathcal{I}$, then at least one of the elementary circuits composing it must have a positive reduced cost. We remark that such circuit might not be a solution of [PP-C](i), but of a different subproblem to price circuits starting with another incompatible pair. Hence, formulating the pricing problems for the circuits as a LPPLC is enough to detect positive reduced cost elementary circuits, even if it may not be the most positive one.

The second result is an original result of Arslan et al. (2024) and it identifies a subclass of the problem instances where it is possible to solve problem [PP-P] by dropping the elementarity requirement of the paths, i.e., [PP-P] reduces to a LPPLC, as well. Precisely, if $L^P \leq L^C + 1$ and problems [PP-C](i), $i \in \mathcal{I}$, do not identify any circuit with a reduced cost larger than $-\beta$, then problem [PP-P] reduces to a LPPLC. Indeed, a non-elementary path of length l contains a sub-tour of length at most $l-1$ which is composed of pairs only. Hence, if problems [PP-C](i), $i \in \mathcal{I}$, do not identify any circuit (of length at most L^C) with a reduced cost larger than $-\beta$, then [PP-P] will not provide any positive reduced cost path which is non-elementary. Note that the comparison is done with $-\beta$ and not zero since for the chains the β term is subtracted from the weight of the outgoing arcs from altruistic donors.

We remark that if $L^P > L^C + 1$, solving the LPPLC on graph $\mathcal{H}^P$ provides an upper bound on the value of [PP-P] since the LPPLC is a relaxation of the ELPPLC.

Online Supplement B. Multipliers of a SR inequality from a Cl inequality

If $|\mathcal{S}| \leq 5$, we assign the multipliers reported by Pecin et al. (2017) which are equal for all vertices in $\mathcal{S}$, i.e., $p_i = p$ for all $i \in \mathcal{S}$. Precisely, if $|\mathcal{S}| \in \{3; 5\}$, we set $p = 1/2$ and if $|\mathcal{S}| = 4$, we set $p = 2/3$.

In the following, we report the mixed-integer program we devise to determine the multipliers in case $|\mathcal{S}| > 5$ or in case assigning those of Pecin et al. (2017) does not yield a violated SR inequality. For each $e \in \mathcal{W}$, we introduce an integer non-negative variable y_e^{int} and a continuous variable $y_e^{frac} \in [0, 0.99]$ to model the integer and fractional parts of coefficients of the λ variables in SR inequality (8). Similarly, we introduce an integer non-negative variable r^{int} and a continuous variable $r^{frac} \in [0, 0.99]$ to model the integer and fractional parts of the right hand-side of SR inequality (8). With an abuse of notation, we denote the continuous non-negative variables modelling the multipliers by p_i, $i \in \mathcal{S}$. In addition, we denote by $\underline{m}$ and $\bar{m}$ two continuous non-negative variables to model the minimal and maximal values of the multipliers. Finally, we denote by $\alpha_1, \alpha_2 \in \mathbb{R}_{>0}$ two weights.

Given a fractional solution $\tilde{\lambda}$ and subset $\mathcal{S}$, we propose the following mixed-integer program to determine the multipliers:

$$[\mathrm{M}(\tilde{\lambda}, \mathcal{S})] \ \min \ \alpha_1 \left(r^{int} - \sum_{e \in \mathcal{W}} \tilde{\lambda}_e y_e^{int} \right) + \alpha_2(\bar{m} - \underline{m}) \tag{B.1}$$

$$\text{s.t. } y_e^{int} = \sum_{i \in \mathcal{S}} a_i^e p_i - y_e^{frac} \qquad \forall e \in \mathcal{W} \tag{B.2}$$

$$r^{int} = \sum_{i \in \mathcal{S}} p_i - r^{frac} \tag{B.3}$$

$$\underline{m} \leq p_i \leq \bar{m} \qquad \forall i \in \mathcal{S} \tag{B.4}$$

$$r^{int} \in \mathbb{Z}_{\geq 0}, \qquad 0 \leq r^{frac} \leq 0.99 \tag{B.5}$$

$$y_e^{int} \in \mathbb{Z}_{\geq 0}, \qquad 0 \leq y_e^{frac} \leq 0.99 \qquad \forall e \in \mathcal{W} \tag{B.6}$$

$$0 \leq p_i \leq 1 \qquad \forall i \in \mathcal{S} \tag{B.7}$$

$$\underline{m}, \bar{m} \in \mathbb{R}_{\geq 0}. \tag{B.8}$$

Objective (B.1) minimises two terms: the first one is the negative of the violation of the SR inequality w.r.t. solution $\tilde{\lambda}$ and the second one is the distance between the minimal and the maximal value of the multipliers to avoid them receiving extreme values. We set weights α_1

and α_2 so that the two objectives are managed hierarchically: among the solutions which maximize the violation, we look for the one that minimises the distance between the minimal and maximal multiplier. Constraints (B.2) model the value of variable y_e^{int} for each $e \in \mathcal{W}$. Similarly, Constraint (B.3) models the value of variable r^{int}. Constraints (B.4) compute the minimal and maximal values of the multipliers, respectively. Finally, Constraints (B.5), (B.6), (B.7) and (B.8) define the domains of the variables of the model.

Such model is solved by a commercial solver, its solution time is negligible in practice. If the value of the first objective is positive, a violated SR inequality is found for set $\mathcal{S}$ with the values of multipliers variables p_i, $i \in \mathcal{S}$.

Online Supplement C. Accelerating techniques

Preprocessing Imposing length constraints on cycles and chains allows us to perform a preprocessing procedure to reduce the size of compatibility graph $\mathcal{G}$. Such a procedure was introduced by Pansart et al. (2022) and is based on the Floyd-Warshall algorithm to compute the shortest path between each pair of vertices in a graph.

Riascos-Álvarez et al. (2023) proposed a procedure to reduce the number of pricing subproblems [PP-C](i), $i \in \mathcal{I}$, to be processed in the pricing problem solution. In addition, the same procedure allows to reduce the size of pricing graphs $\mathcal{H}_i^C$.

Initialisation of RMP. It is well known that column generation suffers from degeneracy which may slow down its convergence. A good initialisation of the set of columns helps in reducing this inconvenience. Hence, we enumerate at most 30000 exchange circuits and paths of length up to three and insert them in the RMP. Precisely, to ensure diversity, we generate at most $30000/|\mathcal{V}|$ exchange circuits/paths starting at each incompatible pair/altruistic donor in the compatibility graph.

Primal heuristic. Embedding primal heuristics in a BPC algorithm helps in improving the lower bound and, consequently, reducing the integrality gap. We implement the so-called *restricted master heuristic*, i.e., we solve formulation [SP] restricted to the subset of variables

generated so far by means of a commercial solver. We call such a heuristic after the root node is solved and at the end of the execution of the BPC algorithm, only if the time limit is reached. We impose a time limit of 60 seconds for the first call and 1800 seconds for the second one. Indeed, preliminary tests revealed that the restricted master heuristic struggles to find good lower bounds on the most complex instances within reasonable computational times. The results also showed that it was useless to call such a heuristic during the execution of the BPC algorithm with smaller time limits: the lower bound did not improve within the time limit, and the number of nodes of the branch-and-bound tree pruned due to the lower bound was negligible.

Tabu list. As in Arslan et al. (2024), when a pricing subproblem [PP-C](i), $i \in \mathcal{I}$, does not provide any positive reduced cost circuit, we insert such subproblem in a tabu list. All subproblems in the tabu list are not solved in the following column generation iterations. They are solved only when subproblem [PP-P] does not provide any positive reduced cost path to ensure the correctness of the column generation procedure.

Online Supplement D. Impact of the valid inequalities

In this section, we evaluate the impact of the valid inequalities presented in Section 4.4, namely the SR inequalities heuristically separated from either the Cl inequalities (9) or the OH inequalities (10).

Precisely, we first assess their impact at the root node of the branch-and-bound tree, then on the entire tree. We conduct this analysis on the most challenging instances, namely the 135 instances of Pansart et al. (2022) with a number of pairs greater than or equal to 500. The objective is the maximization of the medical benefit. Although the instances of Delorme et al. (2023) with a number of pairs greater than 200 are also challenging when the medical benefit is maximized, we choose to report only the results for the 135 challenging instances of Pansart et al. (2022) as they are similar to the ones obtained on the set of Delorme et al. (2023).

In order to perform this analysis, we define the following four configurations of the BPC algorithm: `BPC`: no valid inequalities is separated in the course of the algorithm; `BPC+Cl`: only clique inequalities (9) are separated; `BPC+OH`: only odd-hole inequalities (10) are separated; `BPC+Cl+OH`: all valid inequalities are separated.

First, in Table D.11, we compare the results obtained by the four configurations at the end of the solution of the root node of the branch-and-bound tree. The rows of the table correspond to the four configurations. The first column reports the name of the configuration (*configuration*). The additional column heading used in this table is: the average gap at the root node (*avg.impr. cl.gap[%]*), expressed as a percentage of the gap at the root node when no cuts are added, that is computed as $100(UB^{VI} - LB^*)/(UB^{root} - LB^*)100$, where UB^{root} and UB^{VI} are the upper bounds obtained, respectively, before and after the valid inequalities and LB^* is the best lower bound among those returned by the four configurations. We chose to assess the impact of the valid inequalities by measuring how much they help to close the optimality gap rather than the average upper bound improvement because the upper bounds at the root node are already of very good quality. Hence, when adding the valid inequalities, the upper bounds can decrease only by a small amount that does not reflect the real impact of the inequalities. With this measure, the smaller the value, the better the performance of the inequalities.

Table D.11: Comparison of four configurations of the BPC algorithm at the root node of the branch-and-bound tree.

configuration	#opt.	avg.impr. cl.gap[%]	avg.t[s]	avg.#Cl	avg.#OH	avg.t[s] sep.
`BPC`	2	100.00	82.83	-	-	-
`BPC+Cl`	5	87.03	87.46	8.33	-	0.30
`BPC+OH`	14	75.09	183.23	-	19.69	99.65
`BPC+Cl+OH`	14	73.97	176.54	9.79	13.84	81.56

From the root node analysis, it is clear that separating the Cl and/or the OH inequalities is beneficial w.r.t. configuration `BPC`. Indeed, when the inequalities are considered, the number of solution proven to be optimal increases: it grows to five with `BPC+Cl` and to 14 with the configurations involving the OH inequalities. In addition, adding the valid

inequalities help in reducing the gap at the root node w.r.t. `BPC`. The best configuration is when separating all inequalities (`BPC+Cl+OH`) that leads to a gap of 73.97% w.r.t. the one obtained by `BPC`. Note that only separating OH inequalities also leads to a very good root node gap (75.09% w.r.t. the one at the root node), while only separating Cl inequalities has worse performance (87.03% w.r.t. the one at the root node). The price to pay with the inequalities is an increase of the computational time, which is mostly due to the time required by the separation procedures. Precisely, separating the Cl inequalities is much faster than separating the OH inequalities: the separation time in `BPC+Cl` is on average 0.30 seconds whereas the one of `BPC+OH` is 99.65 seconds. This difference in the time taken by the algorithm can be explained as follows: the procedure used to separate Cl inequalities (see Section 4.4.1) is designed as a very fast heuristic, whereas the procedure to separate OH inequalities (see Section 4.4.2) is designed as an exact algorithm which is turned into a good quality heuristic by considering 30% of starting vertices, that is still time consuming.

In Table D.12, we compare the results obtained by the four configurations when the instances are solved to optimality. The rows and the first column of the table are the same as those of Table D.11. The second column reports the number of additional instances solved to optimality by each configuration in addition to the 50 instances solved to optimality by all configurations (*#add.opt.*).

Table D.12: Comparison of four configurations of the BPC algorithm on the instances solved to optimality.

configuration	#add.opt.	avg.#nodes	avg.t[s]	avg.#Cl	avg.#OH	avg.t[s] sep.
`BPC`	4	922.82	372.55	-	-	-
`BPC+Cl`	3	411.78	288.72	25.36	-	1.87
`BPC+OH`	6	282.72	261.23	-	26.14	14.32
`BPC+Cl+OH`	5	200.02	224.62	20.48	18.76	12.87

The results of the root node analysis are confirmed: including the valid inequalities is beneficial. First, the inequalities allow to solve more instances to optimality: 54 with `BPC+Cl`, 56 with `BPC+OH` and 55 with `BPC+Cl+OH`. In addition, on the 50 instances solved to optimality by all four configurations, adding the inequalities permit to reduce: (i) the

number of the nodes of the branch-and-bound tree by a factor of at least 3.2, on average; (ii) the computational time by a factor of at least 1.5, on average. The best configuration is `BPC+Cl+OH`: these two reduction factors become 9.6 and 1.8, respectively. The number of separated valid inequalities is on the order of tens for all the configurations where they are considered. The separation time follows the same trend as in Table D.11.

Finally, Table D.13 compares the results obtained by the four configurations when the instances are not solved to optimality. The rows and the first column of the table are the same as those of Table D.11. The second column reports the number of additional instances not solved to optimality by each configuration in addition to the 77 instances not solved to optimality by all configurations (*#add.noOpt.*). The following seven columns report statistics regarding the 77 instances not solved to optimality by all configurations. Precisely, heading *avg.LB*/*avg.UB* reports the average lower/upper bound when the time limit is reached.

Table D.13: Comparison of four configurations of the BPC algorithm on the instances not solved to optimality.

configuration	#add.noOpt.	avg.#nodes	avg.LB	avg.UB	avg.gap[%]	avg.#Cl	avg.#OH	avg.t[s] sep.
`BPC`	4	1986.77	39562.21	39747.03	0.42	-	-	-
`BPC+Cl`	5	1246.13	39556.93	39746.01	0.43	92.36	-	11.96
`BPC+OH`	2	1339.88	39564.84	39745.75	0.41	-	15.57	164.16
`BPC+Cl+OH`	3	1115.62	39579.32	39745.06	0.38	85.52	10.78	135.24

The benefit of the valid inequalities on the instances not solved to optimality is less evident. Overall the configurations involving the inequalities explore on average less nodes of the branch-and-bound tree w.r.t. `BPC`. The explanation is that when valid inequalities are active in the RMP, the pricing of the circuits is modelled as an ELPPRC (i.e. a NP-hard problem) and can no longer be modelled as an LPPLC (i.e. a polynomial problem). Therefore, each call to pricing is more time consuming when valid inequalities are active.

In addition, it can be observed that the SR inequalities arising from the OH inequalities are few, and even in the case where only OH inequalities are separated, this number (15.57 on average) is far from the maximum of 100 valid inequalities to be added in the model. On the contrary, there are many SR inequalities arising from the Cl inequalities. This can

be explained by the fact that only Cl inequalities are separated after solving the root node of the branch-and-bound tree. The time spent in the separation of the inequalities is small w.r.t. the total time (less than 3 minutes on average out of one hour). As observed at the root node, when separating only Cl inequalities the separation time is negligible, only a few seconds.

Finally, in the four configurations the average lower bound, upper bound and optimality gap are rather similar. For the lower bounds, they are slightly worse for the configurations involving the separation of the Cl inequalities, and slightly better for the configurations `BPC+OH` and `BPC+Cl+OH`. The quality of the lower bound comes from the final call to the primal heuristic where a commercial solver solves formulation [SP] restricted to the columns generated so far. As already mentioned, such formulation is very difficult to solve, and even with a 30 minutes time limit, the reported lower bound is usually not proven optimal. Hence, there is no direct correlation between the use of valid inequalities and the quality of the lower bounds. When looking at the average upper bounds, although less nodes are explored, the average upper bound of the configurations with the inequalities is slightly better than the one of `BPC`. In this respect, the best configuration is `BPC+Cl+OH`. This balance between worsening the lower bounds and improving the upper bounds yields the optimality gap to be comparable among all the configurations. Again, the best configuration in this respect is `BPC+Cl+OH`.

Online Supplement E. Results on the instances of Delorme et al. (2023) with obj.=MB

In Table E.14, we present the results obtained by the BPC algorithm where the objective function is the maximization of the medical benefit. Delorme et al. (2023) do not test their procedure in the case of maximization of the medical benefit. The rows and the first three columns of the table have the same meaning as those in Table 9. We consider only instances characterised by a number of pairs up to 400.

From the results of Table E.14, it can observed that the type of the objective function

Table E.14: Results on the set of instances of Delorme et al. (2023) where the objective is the maximization of the medical benefit.

Instances			BPC results						Instances			BPC results					
			solved instances			unsolved instances						solved instances			unsolved instances		
L^C	$\|\mathcal{I}\|$	#	#	avg. #nodes	avg.t[s]	#	avg. #nodes	avg. gap[%]	L^C	$\|\mathcal{I}\|$	#	#	avg. #nodes	avg.t[s]	#	avg. #nodes	avg. gap[%]
3	50	20	20	1.00	0.04	0	-	-	6	50	20	20	1.00	0.02	0	-	-
	100	20	20	1.10	0.02	0	-	-		100	20	20	8.50	0.73	0	-	-
	200	20	20	1.40	0.09	0	-	-		200	20	19	627.47	275.26	1	7391.00	0.03
	400	20	20	23.40	4.79	0	-	-		400	20	0	-	-	20	779.35	0.51
4	50	20	20	1.00	0.04	0	-	-	7	50	20	20	1.00	0.02	0	-	-
	100	20	20	2.50	0.07	0	-	-		100	20	20	17.05	2.70	0	-	-
	200	20	20	56.35	6.72	0	-	-		200	20	19	1415.79	882.10	1	6225.00	0.01
	400	20	9	1501.44	661.98	11	8396.82	0.08		400	20	0	-	-	20	437.35	0.56
5	50	20	20	1.00	0.02	0	-	-	8	50	20	20	1.00	0.02	0	-	-
	100	20	20	3.00	0.19	0	-	-		100	20	20	8.65	2.81	0	-	-
	200	20	20	153.80	35.55	0	-	-		200	20	17	371.35	353.96	3	3227.00	0.09
	400	20	0	-	-	20	2175.25	0.31		400	20	0	-	-	20	320.75	0.56

has a huge impact on the results. maximizing the medical benefit makes the instances much harder to be solved by the BPC algorithm. The BPC algorithm still provides 384 optima out of the 480 instances. However, contrary to the results where the objective is to maximize the number of transplants, those optima are obtained by exploring more nodes of the branch-and-bound tree (167, on average) and by spending more time (91 seconds, on average). The 96 instances not solved to optimality are characterised by either 200 or 400 pairs and a maximum length of the cycles at least equal to four. None of the instances with 400 pairs and maximum length greater or equal to five is solved to optimality. Nonetheless, the average optimality gap when the time limit is reached is small, 0.42%, on average. The same behaviour was observed when considering instances with more than 400 pairs.