Very_cool_project.mzn

Valerio Costa, Luca Domeniconi, Claudia Maiolino, Tian Cheng Xia
{valerio.costa, luca.domeniconi5, claudia.maiolino, tiancheng.xia}@studio.unibo.it

1 Introduction

2 CP model

2.1 Decision variables

The CP model is inspired by the model presented in [1] and relies on the following decision variables:

- For each package p, $A_p \in \{1, \dots, m\}$ (`assignments[p]` in MiniZinc) indicates which courier delivers it. More specifically, $A_p = c$ iff the package p is delivered by the courier c.
- For each courier c, $P_{c,d} \in \{1, \dots, n+1\}$ for $d \in \{1, \dots, n+1\}$ (`path[c, d]` in MiniZinc) models the path taken by the courier c. The path is defined such that $P_{c,d_1} = d_1$ iff the location d_1 is not part of the route of c and $P_{c,d_1} = d_2$ iff d_2 is visited immediately after d_1 in the route of c. In other words, each relevant location indicates which is its successor and the overall route is defined as a Hamiltonian cycle that starts from $n+1$ (i.e., the depot).

2.2 Objective function

Once the route of each courier has been found, the objective function is computed as follows:

$$\max_{c \in \{1,\dots,m\}} \sum_{\substack{d \in \{1,\dots,n+1\}: \\ P_{c,d} \neq d}} \texttt{D[}d,\ P_{c,d}\texttt{]} \tag{1}$$

The lower-bound of this function corresponds to the longest route that involves a single item and can be computed as follows:

$$\max_{p \in \{1,\dots,n\}} \{\texttt{D[}n+1,\ p\texttt{]} + \texttt{D[}p,\ n+1\texttt{]}\} \tag{2}$$

2.3 Constraints

To respect the capacity limits of each courier, the following constraint can be defined:

$$\forall c \in \{1, \dots, m\} : \sum_{p \in \{1,\dots,n\}: A_p = c} s_p \leq l_c \tag{3}$$

In MiniZinc, the global constraint `bin_packing_capa` also models this. In our experiments, we did not notice significant differences between the two formulations and decided to use the latter following the best practice of preferring global constraints.

To model the route, the following constraints have to be imposed:

$$\forall c \in \{1,\dots,m\} : \begin{cases} P_{c,n+1} = n+1 & \text{if } \nexists p \in \{1,\dots,n\} : A_p = c \\ P_{c,n+1} \neq n+1 & \text{if } \exists p \in \{1,\dots,n\} : A_p = c \end{cases} \tag{4}$$

$$\forall c \in \{1,\dots,m\}, \forall p \in \{1,\dots,n\} : \begin{cases} P_{c,p} = p & \text{if } A_p \neq c \\ P_{c,p} \neq p & \text{if } A_p = c \end{cases} \tag{5}$$

The constraint defined in Equation (4) imposes that, for each courier, the depot has a successor only if that courier delivers at least a package. On the same note, Equation (5) imposes that only the packages delivered by a specific courier have a successor in the route. Moreover, it is necessary to impose that the route defined by P_c is a Hamiltonian cycle that passes through the relevant destinations. This can be done by constraining all the elements of P_c to be different and by defining a subtour elimination constraint (e.g., Miller-Tucker-Zemlin [3]). In MiniZinc, this can be easily modelled by using the global constraint `subcircuit`.

2.3.1 Symmetry breaking constraints

The most notable symmetry in this problem is between couriers with the same capacity. In fact, as the assignments between two couriers with the same capacity are interchangeable, it is reasonable to fix an ordering and avoid redundancies during search. We experimented with two approaches:

- By imposing an ordering on the amount of assigned packages:

$$\forall c_1, c_2 \in \{1,\dots,m\} : (c_1 < c_2 \wedge l_{c_1} = l_{c_2}) \Rightarrow Q_{c_1} \leq Q_{c_2} \tag{6}$$

  where Q_c is the amount of packages delivered by the courier c.

- By imposing an ordering on indexes of the assigned packages:

$$\forall c_1, c_2 \in \{1,\dots,m\} : (c_1 < c_2 \wedge l_{c_1} = l_{c_2}) \Rightarrow A_{c_1} <_{\texttt{lex}} A_{c_2} \tag{7}$$

  where A_c is a vector containing the packages delivered by the courier c.

Moreover, we experimented with a stronger version of these constraints that is applied between two couriers that satisfy the following condition:

$$\max\{L_{c_1}, L_{c_2}\} \leq \min\{l_{c_1}, l_{c_2}\} \tag{8}$$

where L_c is the actual load carried by the courier c.

2.4 Validation

2.4.1 Experimental design

We experimented variations of our models using as solvers Gecode, Chuffed, and OR-Tools. The search order for all models assigns the packages (`assignments`) first and then searches for the route (`path`) of each courier in decreasing order of capacity. Regarding search, we experimented several combinations of variable selection and assignment strategies on a subset of instances and used only the best ones to obtain the final complete results.

All experiments use the same random seed and were run as workflows on GitHub Actions which provides two cores at 2.45 GHz and 7 GB of memory, which we software limited to 5 GB to guarantee a safe margin for the Docker container to run.

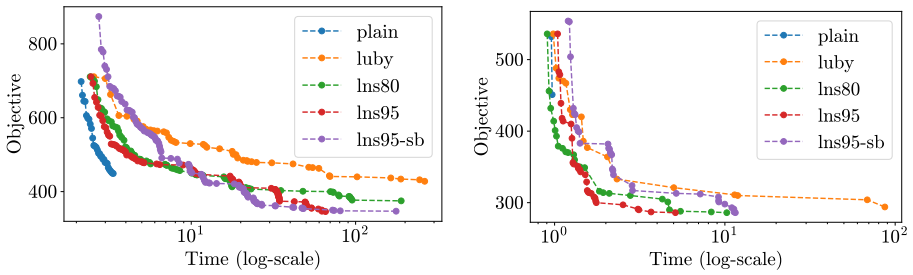
2.4.2 Experimental results

From some preliminary experiments, we observed that first fail and largest domain with weighted degree (`dom_w_deg`) are the best performing search strategies. Therefore, we performed the full experiments using as search strategy `dow_w_deg` for `assignments` and `first_fail` for `path`, respectively. Moreover, between the two symmetry breaking constraints, we observed that the lexicographic ordering defined in Equation (7) has better performances and therefore present only those results. The objective values found by the most significant models are reported in Table 1.

Table 1: Selected subset of CP results. Results in **bold** are solved to optimality. Instances that are all solved to optimality have been omitted.

	Gecode						Chuffed			OR-Tools		
Id	plain	luby	lns80	lns95	lns95 + SB (7)	lns95 + SB (7)(8)	plain	luby	SB (7)	plain	plain + FS	SB (7) + FS
7	201	**167**	**167**	**167**	**167**	**167**	**167**	**167**	**167**	408	**167**	**167**
9	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**	617	**436**	**436**
11	594	597	528	490	503	–	963	–	756	1669	1189	1081
12	449	428	375	**346**	348	–	833	1061	785	1605	706	899
13	648	704	656	616	610	624	1126	914	1126	1734	584	746
14	725	972	794	715	792	–	1449	–	1089	–	–	–
15	659	901	765	738	803	–	1292	–	–	–	–	–
16	451	294	**286**	**286**	**286**	–	487	636	694	998	467	363
17	1324	1468	1119	1076	1155	–	–	–	–	–	–	–
18	691	806	675	662	620	–	1321	–	1779	–	–	–
19	554	412	336	**334**	**334**	–	795	1079	765	1521	623	577
20	1139	1369	1104	1068	1075	–	–	–	–	–	–	–
21	779	687	600	516	529	–	2104	2140	993	2115	1226	–

For Gecode, we experimented different search approaches by testing restart methods, large neighborhood search (LNS), and varying symmetry breaking constraints. We observed that, compared to a depth-first approach, by simply restarting search following the Luby sequence there is a mixed impact on the results with both positive and negative effects. Instead, by also using LNS, results tend to improve globally and we observed that a higher retain probability, with a peak at around 95%, works better. Regarding symmetry breaking constraints, we observed that they tend to worsen the final objective value in the majority of the cases, most reasonably due to the fact that the overhead to impose them is higher than the speed-up in search. For a visual comparison, we show in Figure 1 the evolution of the objective value during search. It can be seen that without restart the objective stops improving in the early stages of search as it most likely gets "stuck" in a branch of the search tree. By introducing some non-determinism, the objective reaches lower values, with LNS being the fastest and best performing.



(a) Instance 12 (b) Instance 16

Figure 1: Intermediate solutions using Gecode

For Chuffed, we only experimented with restarts and symmetry breaking constraints as LNS is not available through MiniZinc. Moreover, as indicated in the documentation[1], we tried enabling free search mode but only obtained worse results. Compared to Gecode, results on bigger instances are generally all worse, while smaller instances tend to be solved faster by Chuffed. As in Gecode, restarts, which we only experimented with random variable selection as it is the only non-deterministic search strategy available, and symmetry breaking constraints both yield mixed effects on the final result.

Finally, for OR-Tools, we conducted fewer experiments as it supports fewer MiniZinc annotations. In this case, we observed that enabling free search mode allows obtaining better results and, similarly to Gecode and Chuffed, symmetry breaking constraints have mixed results. As a side note, we must also observe that, as suggested in the documentation, these results are worse as OR-Tools performs better with multi-threading.

3 SAT model

3.1 Decision variables

3.2 Objective function

3.3 Constraints

3.4 Validation

4 SMT model

4.1 Decision variables

The SMT model uses the logic of quantifier-free linear integer arithmetic (`QF_LIA`) and relies on the following decision variables:

[1] `https://docs.minizinc.dev/en/stable/solvers.html`

- For each package j, $A_j \in \{1, \ldots, m\}$ (`ASSIGNMENTS[j]` in Z3) indicates which courier delivers package j. More specifically, $A_j = i$ iff the courier i delivers package j.
- For each courier i, $P_{i,d} \in \{1, \ldots, n+1\}$ for $d \in \{1, \ldots, n+1\}$ (`PATH[i][d]` in Z3) models the path taken by the courier i. The path is defined such that $P_{i,d_1} = d_1$ iff the location d_1 is not part of the route of the courier i and $P_{i,d_1} = d_2$ iff d_2 is visited immediately after d_1 in the route of c. In other words, each relevant location indicates which is its successor and the overall route is defined as a Hamiltonian cycle that starts from $n + 1$ (i.e., the depot).

4.2 Auxiliary variables

More variables have been defined in order to make it easier to define some constraints and the objective function:

- For each courier i, $D_i \in \mathbb{N}$ (`DISTANCES[i]` in Z3) is equal to the distance traveled by each courier.
- For each courier i, $C_i \in \mathbb{N}$ (`COUNT[i]` in Z3) models the number of packages delivered by courier i.
- For each item j and for each courier i, $PPC_{i,j}$ (`PACKS_PER_COURIER[i][j]` in Z3) model the packages delivered by each courier and it is used for symmetry breaking only.

4.3 Objective function

The objective function is defined as follows:

$$\texttt{obj} = \max_{i \in \{1,\ldots,m\}} D_i$$

4.4 Constraints

- All the elements of each row of P should be distinct:

$$\forall i \in \{1, \ldots, m\} : \quad P_{i,j_1} \neq P_{i,j_2} \quad \forall j_1, j_2 \in \{1, \ldots, n+1\}, j_1 \neq j_2 \tag{9}$$

- Count constraint:

$$\forall i \in \{1, \ldots, m\} : \quad C_i = |\{j \in \{1, \ldots, n\} | A_j = i\}| \tag{10}$$

- Weight constraint:

$$\forall i \in \{1, \ldots, m\} : \sum_{j \in \{1,\ldots,n\} : A_j = i} s_j \leq l_i \tag{11}$$

- Subcircuit constraint: each row P_i should define a subcircuit, that is a Hamiltonian path that ignores all the elements $P_{i,j} = j$, namely the packages that the courier i don't deliver.

- Distance constraint:

$$\forall i \in \{1, \dots, m\} : \quad D_i = \sum_{j \in \{1,\dots,n+1\}} \begin{cases} D_{j,P_{i,j}} & \text{if } P_{i,j} \neq j \\ 0 & \text{if } P_{i,j} = j \end{cases} \tag{12}$$

- Assignment and path constraint:

$$\forall j \in \{1, \dots, n\} : \quad A_j = i \iff P_{i,j} \neq j \tag{13}$$

$$\forall j \in \{1, \dots, n\} : \quad A_j \neq i \iff P_{i,j} = j \tag{14}$$

4.4.1 Symmetry breaking constraints

See Symmetry breaking constraint of CP, section 2.3.1.

4.4.2 Implied constraints

- In order to ensure that each courier delivers at least one package:

$$\forall i \in \{1, \dots, m\} : \quad C_i \geq 1 \tag{15}$$

4.5 Validation

4.5.1 Experimental design

The experimental setup consists of two steps: first, we developed a Python package to automate the generation of SMT-LIB code from a high-level interface, allowing us to easily experiment and compare different solvers. More specifically, as solvers we experimented with Z3, cvc5, OpenSMT, SMTInterpol, and Yices 2. Then, to improve the performances on larger instances, we experimented with two different search strategies using the Z3 Python library (`z3py`):

Two solvers approach As SMT solvers do not allow to assign a priority to the variables, this approach attempts to guide the exploration of the search space by alternating two solvers: the first one finds A and the second one finds P given A. In other words, the former decides which courier delivers which package and the latter decides the route taken by each courier, basically solving m different Travelling Salesman Problems.

Local search approach Similarly to the previous one, this approach also uses two solvers to first find A and then P. However, instead of letting the second solver find an optimal solution for P on its own, it is manually guided by performing a local search starting from a trivial solution (i.e., a path that delivers the items ordered by index).

All experiments use the same random seed and were run as workflows on GitHub Actions which provides two cores at 2.45 GHz and 7 GB of memory, which we software limited to 5 GB to guarantee a safe margin for the Docker container to run.

4.5.2 Experimental results

The results of our experiments are presented in Table 2. Analyzing SMT-LIB results, we can observe that all five solvers perform more or less similarly. The best performing is Yices 2 which solves `QF_LIA` logic based on the simplex algorithm [4]. On the other hand, the worst performing is SMTInterpol which relies on Craig interpolation [2]. Furthermore, experiments in `z3py` show that symmetry breaking and implied constraints do not provide significant contribution in improving the results.

By analyzing the two search approaches, we can observe that using two separate solvers have a negligible impact on the final results with mixed effects when using symmetry breaking constraints. Instead, local search enables the model to find a solution in a reasonable amount of time, even for the largest instances.

Table 2: SMT results. Results in **bold** are solved to optimality. Instances that are all solved to optimality have been omitted.

	SMT-LIB (plain)					`z3py`						
Id	Z3	cvc5	OpenSMT	SMTInterpol	Yices 2	plain	plain + SB	plain + IC	2 solvers	2 solvers + SB	2 solvers + IC	Local search
7	228	210	218	372	**167**	174	168	181	**167**	**167**	**167**	**167**
9	**436**	**436**	**436**	437	**436**	**436**	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	381	**244**	**244**	**244**	**244**	**244**	**244**	**244**	**244**
11	–	–	–	–	–	–	–	–	–	–	–	547
12	–	–	–	–	–	–	–	–	–	–	–	435
13	1446	–	–	–	1490	–	–	–	1812	1346	1832	632
14	–	–	–	–	–	–	–	–	–	–	–	1177
15	–	–	–	–	–	–	–	–	–	–	–	1140
16	–	–	–	–	1032	–	–	–	1510	1944	1861	303
17	–	–	–	–	–	–	–	–	–	–	–	1525
18	–	–	–	–	–	–	–	–	–	–	–	917
19	–	–	–	–	–	–	–	–	–	870	–	398
20	–	–	–	–	–	–	–	–	–	–	–	1378
21	–	–	–	–	–	–	–	–	–	–	–	648

5 MIP model

For the MIP model we choose themsolver-independent language AMPL. Taking inspiration from the AMPL book, we developed our model around the idea to use a binary tensor that encodes the travel of each courier through all the possible delivery point, starting and ending at the depot. This travel is an Hamiltonian cycle and we recall here the definition:

Definition 1 (Hamiltonian cycle)**.** *In the mathematical field of graph theory an Hamiltonian cycle (or Hamiltonian circuit) is a cycle that visits each vertex exactly once.*

In order to preserve the properties of the Hamiltonian cycle and to avoid the possible presence of sub-circuits we added some constraints, and in particular for the last case we followed the Miller–Tucker–Zemlin (MTZ) approach.
The work-flow was based on the construction of three different models: the initial-one, the implied-one, with the adding of the implied constraint, and the symmetry-one, with the adding of the symmetry breaking constraint. The purpose is to show how the adding of this two constraints could improve the performance in terms of execution time and number of simplex-iterations or branching nodes explored.

5.1 Decision variables

The models are based on the following three variables:

- A binary tensor $X \in \{0,1\}^{(n+1)\times(n+1)\times m}$, where $X[i,j,k] = 1$ if and only if the courier k depart from the $i-th$ delivery point and arrive at the $j-th$ one. The column and row $n+1$ stand for the depot.
- A binary matrix $A \in \{0,1\}^{n\times m}$, where $A[i,k] = 1$ if and only if the the package i is delivered by the courier k.
  We observe that this kind of variable is not strictly necessary for the model, but it allowed us to write some constraints in an easier way.
- An auxiliary matrix $u \in \{1,\dots,n+1\}^{(n+1)\times m}$ that keeps track of the order in which nodes are visited by each courier starting from the point 1. It is necessary, following the MTZ formulation, for the sub-circuits elimination. The interpretation is that, fixing the courier k, $u[i,k] < u[j,k]$ implies that the node i is visited before the node j by the courier k.

5.2 Objective function

As objective function, defined from the assignment as the maximum distance travelled by any courier, we used $\max_{k\in 1\dots m} \sum_{i=1}^{n+1} X[i,j,k]D[i,j]$ with the goal to minimize it. As upper bound we chose the sum over all the indexes of the matrix D: $\sum_{i,j=1}^{n+1} D[i,j]$. Instead as lower bound we chose the maximum distance travelled picking only one package: $\max_{i\in 1\dots n}(D[n+1,i]+D[i,n+1])$.

In fact, due to the triangular inequality, if another package is picked up by the courier the distance will grow.

5.3 Constraints

- We defined the constraints that link the variables A and X:

$$\sum_{j=1}^{n+1} X[i,j,k] = A[i,k] \qquad \forall i \in 1 \dots n, \forall k \in 1 \dots m. \tag{16}$$

$$\sum_{i=1}^{n+1} X[i,j,k] = A[j,k] \qquad \forall j \in 1 \dots n, \forall k \in 1 \dots m. \tag{17}$$

- We defined the constraint that specify that each package has to be assigned:

$$\sum_{k=1}^{m} A[i,k] = 1 \qquad \forall i \in 1 \dots n. \tag{18}$$

- We defined the constraint concerned the capacity of each courier:

$$\sum_{i=1}^{n} A[i,k]s[i] \leq l[k] \qquad \forall k \in 1 \dots m. \tag{19}$$

- We defined the constraint that avoid that each courier depart and and arrive at the same point.

$$X[i,i,k] = 0 \qquad \forall i \in 1 \dots n+1, \forall k \in 1 \dots m. \tag{20}$$

- We defined two constraints that ensure respectively that there is one arrival and one departure for each node. This concerns only to the internal nodes, because the depot is visited by each courier.

$$\sum_{i \in 1 \dots n+1, k \in 1 \dots m} X[i,j,k] = 1 \qquad \forall j \in 1 \dots n. \tag{21}$$

$$\sum_{j \in 1 \dots n+1, k \in 1 \dots m} X[i,j,k] = 1 \qquad \forall i \in 1 \dots n. \tag{22}$$

- We defined one constraint that allows the preservation of the flow (if one courier arrives at one node, he departs from the same one):

$$\sum_{i=1}^{n+1} X[i,j,k] = \sum_{i=1}^{n+1} X[j,i,k] \qquad \forall j \in 1 \dots n, \forall k \in 1 \dots m. \tag{23}$$

- We defined two constraints that ensures that each courier starts and end his travel at the depot:

$$\sum_{j=1}^{n} X[n+1, j, k] = 1 \qquad \forall k \in 1 \dots m. \tag{24}$$

$$\sum_{j=1}^{n} X[j, n+1, k] = 1 \qquad \forall k \in 1 \dots m. \tag{25}$$

- Following the MTZ formulation we wrote three constraints in order to avoid sub-circuits:

$$u[i,k] - u[j,k] + 1 \leq n(1 - X[i,j,k]) \qquad \forall i \in 1 \dots n, \forall j \in 1 \dots n+1, \forall k \in 1 \dots m. \tag{26}$$

$$u[i,k] \leq X[n+1,i,k] + (n+1)(1 - X[n+1,i,k]) \qquad \forall i \in 1 \dots n, \forall k \in 1 \dots m. \tag{27}$$

$$u[j,k] \geq (u[i,k] + 1)X[i,j,k] \qquad \forall i \in 1 \dots n, \forall j \in 1 \dots n+1, \forall k \in 1 \dots m. \tag{28}$$

5.3.1 Implied Model

For the implied model we added one more constraint:

$$\sum_{i=1}^{n} A[i,k] \geq 1 \qquad \forall k \in 1 \dots m. \tag{29}$$

The meaning of this implied constraint is that every couriers has to bring at least one package. This information is not explicitly wrote in the assignment but it could significantly help to reach the optimal solution, being one necessary condition. Our claim is the following:

Claim 1. *If the solution is optimal then every courier has to bring at least one package.*

Proof. Let's assume for the sake of contradiction that we have found an optimal solution where there are one or plus couriers which don't bring any package, calling him k_1. Let's suppose that the courier k_j is the one that cover the maximum distance D_j. If we assign one package, calling i, that k_j brought, to k_1 then, due to the triangular inequality, the two new distances, D_1, travelled by the courier k_1, and D_2, travelled by the courier k_j, are less or equal to D_j, in fact:

$$D_1 = D[depot, i] + D[i, depot] \leq D[depot, i_1] + \dots + D[i_r, i] + D[i, i_s] + \dots + D[i_t, depot] = D_j. \tag{30}$$

$$D_2 = D[depot, i_1] + \dots D[i_r, i_s] + \dots D[i_t, depot] \leq D[depot, i_1] + \dots \tag{31}$$

$$\dots + D[i_r, i] + D[i, i_s] + \dots + D[i_t, depot] = D_j. \tag{32}$$

Therefore, D_j is not an optimal solution and we have found a contradiction. □

5.3.2 Symmetry Model

For the symmetry model we added the symmetry-breaking constraint related to the number of packages brought by couriers with the same capacity:

$$\sum_{i=1}^{n} A[i,k] \leq \sum_{i=1}^{n} A[i,j], \tag{33}$$

where $k, j \in 1, \dots, m$ with $k < j$ and $l[k] = l[j]$.

5.4 Validation

5.4.1 Experimental design

For reason of repeatability, we decided to use only open-access solver, HIGHS and SCIP, given by the AMPL framework. Both SCIP and HIGHS give us the information related to the number of simplex iterations performed and branching nodes explored. From what we have understood this is due to the fact that they try to initially solve the relaxed-version of the problem (LP) using the revised simplex-method and finding a lower bound for the solution; then, if the solution found is not integer, they start to solve the MIP part using brunch and cut (SCIP) or brunch and bound (HIGHS). For the resolution of the subproblems generated by these two methods, they proceed as before, starting from relaxation to successively pass to MIP.

5.4.2 Experimental results

For the first ten instances we plotted three graphs about the execution time (in seconds), the number of simplex iterations and branching nodes explored by the two solvers.

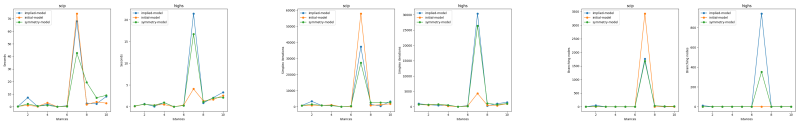


Figure 2: Compared statistics of the performances of the three models, divided by SCIP and HIGHS solvers

We can observe that, while for the SCIP solver the adding of the implied and symmetry breaking constraint improve the performance, it's not the same for HIGHS. But the HIGHS performances are in general significantly better than the SCIP ones.

Instead here we include the table with all the results given by the two solvers and the three models for all the instances.

Table 3: Results of the scip and highs solvers for the three models

Id	initial-model-scip	initial-model-highs	implied-model-scip	implied-model-highs	symmetry-model-scip	symmetry-model-highs
1	**14**	**14**	**14**	**14**	**14**	**14**
2	**226**	**226**	**226**	**226**	**226**	**226**
3	**12**	**12**	**12**	**12**	**12**	**12**
4	**220**	**220**	**220**	**220**	**220**	**220**
5	**206**	**206**	**206**	**206**	**206**	**206**
6	**322**	**322**	**322**	**322**	**322**	**322**
7	**167**	**167**	**167**	**167**	**167**	**167**
8	**186**	**186**	**186**	**186**	**186**	**186**
9	**436**	**436**	**436**	**436**	**436**	**436**
10	**244**	**244**	**244**	**244**	**244**	**244**
11	–	–	–	–	–	–
12	–	–	–	–	–	–
13	642	726	616	694	526	692
14	–	–	–	–	–	–
15	–	–	–	–	–	–
16	–	**286**	–	557	–	320
17	–	–	–	–	–	–
18	–	–	–	–	–	–
19	–	–	–	–	–	–
20	–	–	–	–	–	–
21	–	–	–	–	–	–