Clonal Selection Algorithms for Task Scheduling in a Flexible Manufacturing Cell with Supervisory Control

Ana C. Oliveira*, Tatiana A. Costa†, Patricia N. Pena* and Ricardo H. C. Takahashi‡

*Dep. Electronics Eng., Universidade Federal de Minas Gerais, Belo Horizonte - MG - Brazil
Emails: anaoliver.33@gmail.com, ppena@ufmg.br

†Dep. Exact and Applied Sciences, Universidade Federal de Ouro Preto, João Monlevade - MG - Brazil
Email: tatiana@decea.ufop.br

‡Dep. Mathematics, Universidade Federal de Minas Gerais, Belo Horizonte - MG - Brazil
Email: taka@mat.ufmg.br

Abstract—A new approach for the problem of optimal task scheduling in a manufacturing cell is proposed in this work, as a combination of a clonal algorithm with the supervisory control of discrete-event dynamical systems. Two methodologies are proposed. In the first one, the clonal selection algorithm (CSA) performs the search for the optimal solution, using randomized searches over permutations of sequences of operations. The supervisory control has the role of encoding all the problem constraints, allowing for the search to be conducted on the feasible solution set only. The second methodology is similar, but the CSA uses a local search 2-opt to improve the best individual of each generation. The preliminary results show that both methodologies can obtain significant gains in the total plant operation time in relation to the greedy control policy employed on an example system considered here. A better performance of the CSA + 2-opt methodology can also be observed, when compared with the Clonal Selection Algorithm alone. The proposed methodology provides robustness and flexibility to the solutions – these features are not usually present in most optimization-based solutions for those problems.

I. INTRODUCTION

The task scheduling problem in a manufacturing system is of great importance in industry. Solving this problem means to have an operation free of faults and high efficiency on the use of the resources available in the plant. Without the use of systematic techniques to create the production programs it is not possible to guarantee such characteristics. For that reason, there has been a significant research effort in the last few years to develop systematic tools to deal with such a problem.

The issue of achieving efficient solutions in such problems has been addressed mainly in the field of optimization and metaheuristics. Several of the problems of flexible manufacturing cell scheduling belong to the NP-hard class, and no polynomial-time methods are known for solving them. The exactly optimal solutions for those problems could be found using dynamic programming techniques [1], at exponentially-growing computational costs. Therefore, even for moderate size instances the task of finding the exact solutions become costly. This motivates the usage of heuristic optimization techniques, for instance the ones based on evolutionary computation methods. There is a large number of published literature on the general theme of scheduling problems using metaheuristic methods. Some former works deal with the Job-Shop Scheduling (JSS) problems [2], [3], [4], [5], which have some similarity with the problem of flexible manufacturing system control that is treated here.

All those references mention that an important difficulty in the JSS problem is the very complex constraint structure, which makes the constraint treatment a hard task within the algorithms. Another difficulty which is not even mentioned in most of those references is related to the flexibility and to the robustness of the solutions provided by those methods. The flexibility is related to the capability of the method to accommodate changes in the production schedule “on the fly”, with the system already in operation. The optimization methods, in those cases, would require to re-run the optimization in order to find a new scheduling – what would not be practical in most of the situations. The robustness is related to the capability of the solution to remain at least feasible, and desirably also nearly-optimal, in the case of variations or uncertainty in the optimization parameters (mainly the times associated to the operations). It is quite possible that a solution which is time-optimal for a given set of parameters becomes infeasible if those parameters change slightly – which may prevent the application of optimization techniques in several real manufacturing plants.

The Supervisory Control of Discrete Event Systems (SCDES) approach has been developed with the aim of determining automatically the commands that can be applied to the system in order to avoid its evolution to prohibited states, where the system can be damaged or enters in deadlock [6]. The objective of SCDES is to construct controllers that are minimally restrictive, meaning that only the prohibited trajectories are avoided, and all the other trajectories are allowed [7]. So, the SCDES does not formulate optimal task scheduling; it just allows all the legal trajectories.

A heuristic approach has been suggested in order to deal
with the issue of choosing a sequence of operations for which the time elapsed becomes short: on each step, the allowed operation which takes the smallest time is always chosen [7]. This, indeed, causes a significant reduction in the operation time in comparison with random choices. However, due to the greediness of the choices, and due to the dynamic nature of the problem, in which the choices performed in each step determine the available choices in the subsequent steps, this policy does not lead to the globally optimal operation time.

This paper proposes a new approach for the task sequencing problem is manufacturing plants, conceived as the fusion of SCDES and an evolutionary computation algorithm, the Clonal Selection Algorithm (CSA) [8]. The CSA is one of the algorithms that have been inspired in working principles of the immunological system of mammals. The evolution of the solutions is conceived as a metaphor of the process in which, in the immune system of the living organisms: (i) several replicas of the current antibodies are created, with the most useful antibodies receiving more replicas; and (ii) these replicas are mutated randomly, generating solutions that are similar to the original ones, but different to some extent from any solution ever generated. The repeated application of these steps leads to the progressive generation of enhanced solutions which, in the case of the actual immune system, means solutions that fit infective antigens and, in the case of the evolutionary computation algorithm, means solutions that present better objective function value [8]. The idea is to use the control structure provided by SCDES as the oracle of all information about the problem constraints. By construction, each solution given by SCDES is a valid solution, what means that it is also a feasible solution of the optimization problem to be solved by the CSA. The optimization is then processed considering only the set of feasible solutions. All the work over the supervised plant is performed over the reduced supervisors and subsystems, implemented in a simulation separately. The behavior of the supervised plant, obtained by composing nonconflicting supervisors is never computed.

Some work has been done on calculating optimal director/schedulers/controllers as subautomata of the supervised plant obtained through SCDES [9], [10], [11], among others. However, those works apply to some specific classes of plants. The work proposed in this paper is different from those references. It makes use of the SCDES solution to codify the search space of the optimization problem and uses an evolutionary algorithm to search for optimal solutions. It should be noticed that, although there are some studies that are applicable to some special cases, the issue of solution optimality in the most generic cases has not been considered in the literature in the field of SCDES.

It should be noticed that the approach proposed here uses the SCDES in order to encode the constraint structure, in this way avoiding a main difficulty that arises in former optimization or metaheuristic methodologies. In addition, the controller, once obtained, encodes all the feasible solutions that could be used for dealing with any change of production plan that occurs in operation time. If, for any reason, some operation is to be executed in a time different than was planned, the SCDES will still have a course of action that leads to the correct execution of the whole production task. It is possible to define policies over the SCDES controller that, although not executing the production in minimal time, still perform very reasonably. This is a significant advantage over JSS approaches, which would need to re-run the algorithm in order to redefine the production plan. The difficulties of usual optimization methods with the solution flexibility and robustness are, therefore, circumvented in the proposed approach.

The methodology presented is tested in an example problem that has been used in previous work on SCDES [12], [13]. Improvements have been observed, in relation to non-optimal strategies typically produced by the greedy approach.

II. DESCRIPTION OF THE EXAMPLE PLANT

The Flexible Manufacturing System (FMS) presented in Fig. 1 produces two types of products from raw blocks and raw pegs: a block with a conical pin on top (Product A) and a block with a cylindrical painted pin (Product B). The FMS consists of eight devices. The lathe, the mill, the painting device and the assembly machine perform tasks over the pieces. The conveyors and the robot move the pieces among the machines. The devices are connected through buffers $B_i$, $i = 1, \ldots, 8$, each with capacity for one part. The arrows in Fig. 1 indicate the events representing the flow of unfinished parts through the FMS. More details about the FMS can be found at [13].

The specifications that should be attained by the SCDES solution are: (i) avoid overflow and underflow of pieces in the buffers; (ii) allow the simultaneous operation of the mill and lathe; (iii) assure that both Products A and B are manufactured.

![Flexible Manufacturing System](image)

Fig. 1. Flexible Manufacturing System

This example was first presented in the context of Multi-tasking Supervisory Control of DES [13].

III. USING SUPERVISORY CONTROL TO MODEL THE CONSTRAINTS

The main idea of the approach proposed here is to use SCDES as the tool to model the task scheduling problem constraints.

A. Supervisory Control applied to the FMS

Local Modular Control [14], an extension to SCDES, is used. Each supervisor is projected with a partial view of
the plant. The following steps can be applied to achieve the supervisors for any plant.

1) Modeling the subsystems as a product system, by designing eight asynchronous automata, one for each device, as shown in Fig. 2. Events represented by odd numbers are controllable (the events that can be disabled by the controller, in this context, usually called supervisor) and events represented by even numbers are noncontrollable (consisting of the spontaneous response of the system - those events cannot be disabled by the supervisor).

2) Modeling the safety specifications as automata. Details can be found at [12].

3) Obtaining the local plant for each specification, by composing all subsystems that share events with the specifications.

4) Synthesizing the supervisors and checking for conflict. A desired language is obtained by the composition of each specification and its local plant. This language is checked for controllability, that consists of checking if the control system does not attempt to disable noncontrollable events. In case the desired language does not pass the controllability test, the supremal sublanguage of the desired language that is controllable in relation to the local plant is obtained. The automaton that implements such a language is the supervisor and it has, along with states and transitions, a disablement structure. The disablement structure indicates what controllable events need be disabled in each state of the supervisor, corresponding to a state in the plant. Supervisors are nonblocking by construction; however, there is no guarantee that the joint operation of all supervisors will not block the plant. If blocking occurs, the supervisors are said to be conflicting\(^1\). A test should be applied to the supervisors to check if they are nonconflicting. More details can be found in [14].

5) Supervisor reduction. The supervisors are obtained after composing the local plant and the specification. As a consequence, much of the information contained in the supervisor is also in the plant. This redundant information can be removed, leading to reduced supervisors, which carry only the action of the supervisor over the plant. For more details on supervisor reduction algorithms, refer to [15] and [16]. Reduced supervisors for this system are presented in Fig. 3, where the dashed lines labeled by events indicate what events are disabled in each state.

---

\(^1\)One specification was obtained for each buffer. However, the resulting supervisors are conflicting. A conflict resolution scheme was proposed that generated a monolithic solution for specifications of buffers \(B_7\) and \(B_8\) [12].

---

Fig. 2. Modeling the plant

Fig. 3. Reduced Supervisors

The plant and the reduced supervisors are used to implement the simulation.

**B. Simulation of the System**

The system, composed by the plant and the reduced supervisors, is represented in a computational program that simulates its operation.

By definition, time is not part of the DES model. However, in real systems time is inherent and the idea is to associate values that indicate the duration of each operation of each subsystem.

In the FMS, to the events that represent uncontrollable events were associated 1 time unit (t.u.) since an uncontrollable event reports the end of an operation. Different values (from 16t.u. to 38t.u.) were associated to the controllable events.

In Table I, the values associated to each controllable event is presented in [12].

---

\(^1\)One specification was obtained for each buffer. However, the resulting supervisors are conflicting. A conflict resolution scheme was proposed that generated a monolithic solution for specifications of buffers \(B_7\) and \(B_8\) [12].
The simulation algorithm itself can be divided into two parts:

(i) Initialization: The automata and disablement matrices are loaded. A vector \( \text{state} \) with \( n + m \) positions is defined to represent the current state of the simulation. Vector \( \text{state} \) is initialized with the current state of each automata, that would be all the automata in their initial state. The variable \( t \) is the clock and is set to 1.t.u..

(ii) Main program (a loop):

1) A list of feasible events (LFE) is created based on the current state of the automata.

2) A list of events to be disabled is created based on the current state of the reduced supervisors. Such events are removed from the (LFE), generating the list of feasible events under control (LFEuC).

3) To each event of the LFEuC is associated a number that indicates in what time of the simulation such event is allowed to be executed, based on \( t \), Table I and the relation among events that is defined by the automata of the plant \(^2\). The LFEuC is sorted by the deadline associated to the events.

4) The event to be executed is chosen among the ones that have the number associated to them smaller than or equal to \( t \).

   I. the uncontrollable events are priority;

   II. if more than one event of the same type fulfill the condition based on \( t \), the event with the smallest time is chosen; if there is a tie, random numbers are associated to them to assist the choice.

   If no event fulfill this condition, no event is executed in this step.

5) The event executed is removed from LFEuC, the current state of the subsystems and reduced supervisors are updated.

\( ^2 \)Let the simulation time be \( t = 5 \) and let event 11 be executed. Since \( 25.t.u. \) is associated to event 11, event 12 (both in \( C_1 \)) will be a candidate for execution only when the simulation time reaches \( t = 30.t.u. = (5 + 25).t.u. \). This implementation allows to interpret the transitions as instantaneous and the time associated to the controllable event implementing the time spent in state 1 of conveyor \( C_1 \) (Fig. 2(a)).

Some characteristics of the simulation program are presented:

- The input of the simulation is the batch size. In the case of the SFM, the inputs are the number \( n_A \) of products A and the number \( n_B \) of products B to be produced;

- The output gives the time elapsed since the simulation started.

The simulation of the plant is used for two purposes. First, the simulator is employed in an autonomous mode, following a greedy decision policy, as proposed in [7] and described above. This autonomous operation mode is employed in order to provide a baseline for comparison purposes.

The simulation is used also in an interactive mode, within the main loop of the optimization procedure, as a tool for encoding the problem constraints. In this mode of operation, the input of the simulation procedure is a sequence of controllable events, provided by the optimization procedure, that should be evaluated. The simulator processes the sequence and, in case the sequence is able to be processed until the end, the simulator provides, as outputs, the information that the sequence is feasible and the total time of its processing. In case the sequence is not executable up to its end, the simulator returns the position of the sequence that blocked and the set of feasible continuations at that point. This information is employed in a correction procedure (that is part of the optimization algorithm) that is performed until the sequence becomes feasible.

The next section presents the proposed optimization procedure.

IV. OPTIMAL SEQUENCING

The optimization objective is to minimize the time elapsed when producing a specific set of products A and B. Given a number of products of the type A and a number of products of the type B that should be produced, the idea is to generate an optimal task sequencing that minimizes the time necessary to produce the set. The optimization algorithm that was used is a simplified version of the Clonal Selection Algorithm. A detailed discussion about immunological algorithms can be found at [8]. Next subsection presents a description of the algorithm used.

A. Event Sequencing Codification

The production of each product (A and B) can be modeled by the sequence of events that should be executed in order to manufacture it. Analysis of the topology of the plant (Fig. 1) suggests that for each product there are two subsequences that can be separated (the sequence to make the base and the sequence to make the pin) and the sequencing in each subsequence should be kept.

The optimization algorithm generates clones of a feasible sequence and mutates the clones to make them different from the parent. The idea is to perform permutations without

<table>
<thead>
<tr>
<th>Subsystems</th>
<th>Event</th>
<th>t.u.</th>
<th>Event</th>
<th>t.u.</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>11</td>
<td>25</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>C2</td>
<td>21</td>
<td>25</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>Robot</td>
<td>31</td>
<td>25</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td>19</td>
<td>34</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>16</td>
<td>36</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>24</td>
<td>38</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>20</td>
<td>30</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Mill</td>
<td>41</td>
<td>30</td>
<td>42</td>
<td>1</td>
</tr>
<tr>
<td>Lathe</td>
<td>51</td>
<td>38</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>AM</td>
<td>53</td>
<td>32</td>
<td>54</td>
<td>1</td>
</tr>
<tr>
<td>61</td>
<td>15</td>
<td>64</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>25</td>
<td>66</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>71</td>
<td>25</td>
<td>72</td>
<td>1</td>
</tr>
<tr>
<td>73</td>
<td>25</td>
<td>74</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PD</td>
<td>81</td>
<td>24</td>
<td>82</td>
<td>1</td>
</tr>
</tbody>
</table>


To evaluate the individuals it is necessary to check if the mutated individual is feasible. An individual is feasible if the sequence of events encoded in it can be executed to the end. If some event of the sequence cannot be executed, then the individual is infeasible. The verification of feasibility of each individual is done by the simulator that interacts with the optimization algorithm. The simulator returns to the algorithm if the sequence was executed to the end (and the time to execute the sequence) or if it blocked (the position that the sequence blocked, the code of the events that can be executed in that state). The following sequence of tasks is executed until the individual becomes feasible:

1: The position that the sequence blocked and the event that causes it are stored;
2: One of the feasible continuations given by the simulator is suggested to replace the event that blocked;
3: A search to find such an event in positions after the position stored in step 1 is performed;
4: If no occurrences are found in step 3, another feasible continuation given by the simulator is chosen to replace the event that blocked, and the algorithm goes back to step 3, otherwise go to 5;
5: One of the positions found in step 3 is chosen, randomly, and the infeasible event (stored in step 1) is replaced.
6: The feasibility of the new individual is checked. If not feasible, steps 1 to 6 are repeated.

It should be noticed that the loop composed by steps 2 and 3 eventually finishes, since there must be necessarily at least one event after the current position, in the list provided by the simulator, that allows the execution of the sequence up to its end.

The number of clones produced from each individual is calculated in terms of a linear function that guarantees that the best individual generates \((\beta)N\) clones while the worst one generates \((1 - \beta)N\) clones. The parameter \(\beta\) may assumes values between 0 and 1.

The mutation is performed only on the clones, such that the parents are not altered. The mutation intensity varies with a fitness function such that an individual with better fitness has lower mutation intensity, suffering mutations that are less disruptive, while individuals with low fitness have high mutation intensity. The fitness function used is based on the ranking of solutions. The best solution has fitness 1, the second best has fitness 2 and so on.

Three mutation operators were used:

M1: Random mutation - events are randomly chosen to be swapped;
M2: Block change - blocks of events of the same individual are swapped. The size of the block \(z_2\) can vary from one individual to another one. The blocks can have from \(z_2 = 1\) event to at most \(z_2 = 20\%\) of the size of the sequence;
M3: Event swap with higher probability at the beginning - events are exchanged such that events in the beginning of the sequence have more probability of being swapped;

The following pseudo-code describes the clonal algorithm that was used in the work.

1: Initial population of \(2N\) individuals
2: Initial population evaluated
3: \(N\) best individuals are selected
4: While stop criterion not achieved
5: Individuals are cloned (copies are created)
6: Clones are mutated, keeping the parent
7: Mutated clones are evaluated
8: Best individual of each set composed by the parent and its clones is selected for the next generation
9: End-while

The initial population, generated randomly with uniform probability, has \(N_{ini} = 2N\) individuals, which is bigger than the population of \(N\) individuals that will be used in the subsequent iterations. This higher initial randomness is used to allow exploration of the search space such that only the more promising regions are searched later.
At each generation, one mutation operator is chosen randomly, to perform the mutation process.

The selection for the construction of the next population is performed by choosing the best individual of each group formed by a parent and its clones. There is no comparison among clones of different parents.

The algorithm stops if one of the two conditions are verified: (i) the number of generations is greater than a given threshold (20) or if the number of generations without improvements is greater than a given threshold (5).

### C. CSA + 2-opt Local Search

In the CSA + 2-opt methodology, the same CSA described in the last section is used. However, at each generation of CSA, every time a new overall best individual is found, it is improved by a 2-opt local search procedure.

A 2-opt local search [18] is an improvement heuristic initially proposed to deal with the Traveling Salesman Problem (TSP). As the structure of a solution to the problem presented is similar to a solution of TSP, this local search was chosen to be used in this work. The procedure is composed by the following steps:

- A position in the solution sequence is fixed;
- All possibilities for a second change are examined, producing new neighbors of the current solution;
- If the value of the best neighbor is better than the value of the current solution, this neighbor becomes the current solution and the process is restarted;
- The algorithm finishes when the value of the best neighbor is not better than the value of the current solution, in other words, when a local optimal solution is found.

Next section presents the results obtained by the application of the two methodologies of sections IV-B and IV-C in the example plant described in Section II.

### V. Results

Before running the tests that are described in this section, the algorithm parameters $\beta$, $z_2$ and $N_{ini}$ were tuned using batches of 10 runs of the (2,0), (1,1) and (0,2) instances of the test problem for each set of tentative parameters. The values were varied, looking for the best performances in a given budget of function evaluations. The parameter values that were chosen by this procedure were: $\beta = 0.8$, $z_2 = 20\%$, and $N_{ini} = 2N$.

Some tests were performed and the results are presented in Table II. In that table:

- The first column indicates the number of products that are produced.
- The four columns under CSA indicate the best result, the average, the standard deviation and the number of evaluations of objective function of the solutions found by the Clonal Selection Algorithm, from 30 runs.
- The four columns under CSA + 2-opt indicate the best result, the average, the standard deviation and the number of evaluations of objective function of the solutions found by the CSA + 2-opt, from 30 runs.
- The three columns under Greedy indicate the best result, the average and the standard deviation found by the software of simulation choosing the events under a Greedy policy, from 30 runs.

The best values achieved for each instance are highlighted.

For the CSA and CSA + 2-opt algorithms, all optimized results are obtained from executions with a population of $N=15$ individuals and 20 generations at most.

If the greedy algorithm has minimal times that are compatible, in some cases, with the CSA solution, the deviation and the average are much bigger. That means that the use of a greedy approach cannot guarantee a good execution at every run. On the other hand, the table shows that the benefits of using optimization methods increase with the complexity of the problem. For batches of size up to 2 products, CSA and CSA+2-opt reached the same value. For bigger batches, the use of a local search enhances the solution without increasing substantially the computational cost.

### VI. Conclusions

In this work, a new approach for the optimal sequencing of tasks in manufacturing cells was presented, based on the combination of an evolutionary algorithm, that searches for optimal solutions, with supervisory control of discrete event systems, that provides the implementation of the constraints such that the search is performed over the set of feasible solutions.

Although only preliminary tests have been performed, the results obtained so far suggest that the approach is promising. More exhaustive and systematic tests, over a wider range of problems, and with different parameters of the algorithm are to be performed in order to establish more conclusive results.

### Acknowledgments

The authors acknowledge the support by the Brazilian agencies CNPq, CAPES and FAPEMIG. This research was also supported by a Marie Curie International Research Staff Exchange Scheme Fellowship within the 7th European Community Framework Programme.

### References

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1)</td>
<td>204</td>
<td>204.03</td>
<td>0.18</td>
<td>750</td>
<td>204</td>
<td>204.00</td>
<td>0.00</td>
<td>1060</td>
<td>206</td>
<td>217.00</td>
<td>11.17</td>
<td></td>
</tr>
<tr>
<td>(1,0)</td>
<td>160</td>
<td>160.00</td>
<td>0.00</td>
<td>790</td>
<td>160</td>
<td>160.00</td>
<td>0.00</td>
<td>870</td>
<td>161</td>
<td>164.60</td>
<td>4.48</td>
<td></td>
</tr>
<tr>
<td>(1,1)</td>
<td>250</td>
<td>251.70</td>
<td>1.56</td>
<td>5360</td>
<td>250</td>
<td>251.70</td>
<td>1.56</td>
<td>7170</td>
<td>254</td>
<td>294.13</td>
<td>26.67</td>
<td></td>
</tr>
<tr>
<td>(1,2)</td>
<td>254</td>
<td>254.87</td>
<td>0.73</td>
<td>42220</td>
<td>254</td>
<td>254.87</td>
<td>0.73</td>
<td>49710</td>
<td>258</td>
<td>261.50</td>
<td>4.18</td>
<td></td>
</tr>
<tr>
<td>(2,0)</td>
<td>303</td>
<td>303.70</td>
<td>0.27</td>
<td>7750</td>
<td>303</td>
<td>303.70</td>
<td>0.27</td>
<td>10100</td>
<td>305</td>
<td>308.87</td>
<td>1.99</td>
<td></td>
</tr>
<tr>
<td>(2,1)</td>
<td>336</td>
<td>336.77</td>
<td>8.72</td>
<td>31750</td>
<td>336</td>
<td>336.77</td>
<td>8.72</td>
<td>36780</td>
<td>340</td>
<td>370.67</td>
<td>29.57</td>
<td></td>
</tr>
<tr>
<td>(2,2)</td>
<td>415</td>
<td>415.10</td>
<td>7.19</td>
<td>116040</td>
<td>415</td>
<td>415.10</td>
<td>7.19</td>
<td>132210</td>
<td>425</td>
<td>427.37</td>
<td>26.60</td>
<td></td>
</tr>
<tr>
<td>(0,3)</td>
<td>406</td>
<td>406.10</td>
<td>1.47</td>
<td>10170</td>
<td>406</td>
<td>406.10</td>
<td>1.47</td>
<td>14880</td>
<td>409</td>
<td>414.50</td>
<td>0.54</td>
<td></td>
</tr>
<tr>
<td>(3,0)</td>
<td>340</td>
<td>340.30</td>
<td>1.21</td>
<td>9040</td>
<td>340</td>
<td>340.30</td>
<td>1.21</td>
<td>12720</td>
<td>343</td>
<td>348.67</td>
<td>7.13</td>
<td></td>
</tr>
<tr>
<td>(3,1)</td>
<td>420</td>
<td>420.77</td>
<td>0.57</td>
<td>503</td>
<td>420</td>
<td>420.77</td>
<td>0.57</td>
<td>20800</td>
<td>423</td>
<td>428.13</td>
<td>25.57</td>
<td></td>
</tr>
<tr>
<td>(0,4)</td>
<td>428</td>
<td>428.13</td>
<td>1.32</td>
<td>12540</td>
<td>428</td>
<td>428.13</td>
<td>1.32</td>
<td>20080</td>
<td>431</td>
<td>439.36</td>
<td>9.53</td>
<td></td>
</tr>
<tr>
<td>(4,0)</td>
<td>406</td>
<td>406.10</td>
<td>1.47</td>
<td>10170</td>
<td>406</td>
<td>406.10</td>
<td>1.47</td>
<td>14880</td>
<td>409</td>
<td>414.50</td>
<td>0.54</td>
<td></td>
</tr>
</tbody>
</table>


