Büther, Marcel

Working Paper
Beam search for the elastic generalized assignment problem

Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 634

Provided in Cooperation with:
Christian-Albrechts-University of Kiel, Institute of Business Administration

Suggested Citation: Büther, Marcel (2008) : Beam search for the elastic generalized assignment problem, Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, No. 634, Universität Kiel, Institut für Betriebswirtschaftslehre, Kiel

This Version is available at:
http://hdl.handle.net/10419/147553

Terms of use:
Documents in EconStor may be saved and copied for your personal and scholarly purposes.

You are not to copy documents for public or commercial purposes, to exhibit the documents publicly, to make them publicly available on the internet, or to distribute or otherwise use the documents in public.

If the documents have been made available under an Open Content Licence (especially Creative Commons Licences), you may exercise further usage rights as specified in the indicated licence.
Beam Search for the Elastic Generalized Assignment Problem

Marcel Büther

März 2008
Abstract

The elastic generalized assignment problem (eGAP) is a natural extension of the generalized assignment problem (GAP) where the capacities are not any longer fixed but can be adjusted which is expressed by continuous variables. These variables might be unbounded or restricted by a lower or upper bound, respectively. This paper describes an algorithm based on beam search, combined with Lagrangian relaxation and local search to provide strong lower as well as upper bounds for the eGAP.

Keywords: elastic generalized assignment problem, mixed integer programming, beam search, Lagrangian relaxation, local search, binary representation

1 Introduction

The elastic generalized assignment problem, eGAP for short, is a natural extension of the generalized assignment problem (GAP), a well-known combinatorial optimization problem with applications in production, logistics, and distribution planning. The GAP is to assign jobs to agents in order to minimize assignment cost without extending a given capacity for each agent. While the capacity can not be influenced according to the GAP, the eGAP considers the opportunity to extend or reduce, respectively, the available capacity. Extending the capacity causes cost while reducing capacity provides a profit. The GAP is well-known to be NP-hard, NP-hardness of eGAP follows straightforwardly, see Nauss [24].

While there exist several efficient algorithms for the GAP, see e.g. Yagiura et al. [32], the eGAP has not been studied much so far, although it is of great practical relevance. To the best of our knowledge, there is only one recent paper by Nauss [24] for a special case of the eGAP. It concerns a branch&bound algorithm to solve small but difficult instances. This algorithm is based on a previous work, which was developed for the GAP, see Nauss [23].

Due to the possibility of adjusting the capacity the well-known cover inequalities, see Nemhauser and Wolsey [25], Schrijver [28] as well as Fügenschuh and Martin [11], can not be used one-to-one. Instead, the powerful formulations developed by Wolsey [30] have to be adapted. This seems to be expensive here, see Ceria et al. [7], due to the number of possible cuts. Thus we already developed a reformulation of the eGAP, which uses a binary coding of the adjustment, see Büther [5]. This reformulation can be transferred to an equivalent standard GAP, which enables us to implement well-known and established algorithms directly. For large instances, the commercial solver ILOG CPLEX 10.1 applying a branch&bound procedure, fails to find the optimal solution due to memory lacks. Therefore, we developed an heuristic, which is based on a truncated branch&bound procedure, called beam search, see e.g. Valente and Alves [29], to control the memory. In the process, Lagrangian relaxation is used to strengthen the lower bounds and local search to find good feasible solutions.

First we present the mixed integer formulation in section 2 and the integer reformulation in section 3. Then, in section 4, the basic components as well as the overall algorithm will be displayed. In section 5 we provide some computational results and section 6 finally concludes the paper.
2 Model Formulation

Given a set \( M = \{1, \ldots, m\} \) of agents and a set \( N = \{1, \ldots, n\} \) of jobs, the objective of the eGAP is to serve all jobs at minimum cost, regarding the adjusted capacity of each agent. To this end, the following binary decision variables are used:

\[
x_{ij} = \begin{cases} 
1 & \text{if job } j \text{ is assigned to agent } i \\
0 & \text{otherwise}
\end{cases}
\]

The binary variables ensure that a job can only be served fully by an agent.

The following parameters are assumed to be given: Each agent \( i \) provides a fixed amount of capacity \( b_i \) \((b_i > 0)\) of an arbitrary resource. In contrast to the GAP, the original capacity \( b_i \) can be adjusted. Continuous variable \( s_i \) represents the amount of capacity which is added \((s_i > 0)\) or removed \((s_i < 0)\), respectively. The cost for the capacity adjustment are \( c_i, c_t > 0 \).

Job \( j \) asks for a specific amount \( a_{ij} \) \((a_{ij} > 0)\) of the capacity of a particular agent if the corresponding \( x_{ij} \) is one. For serving job \( j \)'s demand from agent \( i \), the total cost \( c_{ij} \) \((c_{ij} \geq 0)\) occur. These cost needn't depend on the value \( a_{ij} \).

Now, the problem can be stated mathematically as follows:

\[
\min \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} + \sum_{i \in M} c_t s_i \tag{1}
\]

s.t. \[
\sum_{j \in N} a_{ij} x_{ij} \leq b_i + s_i \quad \forall \ i \in M \tag{2}
\]

\[
\sum_{i \in M} x_{ij} = 1 \quad \forall \ j \in N \tag{3}
\]

\[
l_i \leq s_i \leq u_i \quad \forall \ i \in M \tag{4}
\]

\[
x_{ij} \in \{0,1\} \quad \forall \ i \in M, \forall \ j \in N \tag{5}
\]

The objective function (1) is twofold. The first part represents the sum of assigned jobs' cost. The second part considers cost and sales when additional capacity is bought and superfluous capacity is sold, respectively. Constraints (2) ensure that the (modified) capacity limit of the agents is not exceeded. Constraints (3) in combination with (5) ensure, that each job is assigned exactly to one agent. Finally, constraints (4) give the bounds for the capacity adjustment.

Before presenting reductions of eGAP to GAP, we introduce some definitions. For short, a solution of the eGAP can be expressed by the tuple \((x, s)\), where \( x \) is the binary vector containing all values of \( x_{ij}, i \in M \) and \( j \in N \), and \( s \) is the vector of the corresponding capacity modifications \( s_i, i \in M \).

Note, that the number of feasible solution of the eGAP is not smaller than that of a corresponding GAP with capacities of \( b_i + l_i \), for each \( i \in M \). This might be a burden, if an heuristic is used to solve the eGAP, which is based on pure local search.
3 Model Reformulation

First of all, we propose to transfer the concepts, first developed for the knapsack problem with a continuous variable, KPC for short, see Büther and Briskorn [6]: We implement the binary coding of the capacity adjustments for the eGAP, see Büther [5] for additional information.

Therefore, we replace the capacity adjustment \( s_i \) by \( s_i = u_i - \bar{s}_i \) \( \forall i \in M \), see Nauß [24]. The new variables \( \bar{s}_i \) represent the unused capacity adjustments and thus, they are bounded by \( 0 \leq \bar{s}_i \leq u_i - l_i \) \( \forall i \in M \). For illustration one can imagine, that in advance the capacities are extended at cost \( c_i \cdot u_i \) and then, during the optimization, the superfluous capacities can be sold which is expressed by \( \bar{s}_i > 0 \). Now, the continuous variables \( \bar{s}_i \) are substituted by binary code \( \bar{s}_i = \sum_{k_i \in D_i} e_{ik_i} y_{ik_i} \), with \( D_i = \{1, \ldots, d_i\} \) \( \forall i \in I \). Here, binary variable \( y_{ik_i} \) equals one if and only if \( e_{ik_i} \) contributes to the sold capacity.

Now, we detail our requirements regarding \( d_i \) and \( e_{ik_i} \), see Büther [5]:

i) Values of \( \bar{s}_i \) must not undershoot 0 or exceed \( u_i - l_i \), respectively, that is \( 0 \leq \sum_{k_i \in D_i} e_{ik_i} y_{ik_i} \leq u_i - l_i \) for each binary vector \( y_i = \{y_{i1}, \ldots, y_{id_i}\} \).

ii) Given a precision \( p \in \mathbb{N} \) for each value \( u_i \in [0, u_i - l_i] \) there must be a binary vector \( y_i \) such that \( |u_i - \sum_{k_i \in D_i} e_{ik_i} y_{ik_i}| \leq 10^{-p} \).

iii) Regarding i) and ii) the number of additional binary variables \( d_i \) should be chosen as small as possible.

It is well-known, that any integer value can be stated by a binary representation. Thus, by choosing the values for \( e_{ik_i} \) appropriate, we can represent any integer number between 0 and \( u_i - l_i \). If \( u_i - l_i = 2^r - 1 \) with \( r \in \mathbb{N} \) then \( d_i = \log_2(u_i - l_i + 1) = r \) and \( e_{ik_i} = 2^{k_i-1} \) for each \( k_i \in D_i \) is the obvious choice. Otherwise, if \( u_i - l_i \neq 2^r - 1 \) for all \( r \in \mathbb{N} \) we choose \( d_i = \lceil \log_2(u_i - l_i + 1) \rceil \), \( e_{ik_i} = 2^{k_i-1} \) for each \( k_i \in \{1, \ldots, d_{i-1}\} \) and \( e_{id_i} = u_i - l_i - 2^{d_{i-1}} + 1 \).

Additionally, the concept can even be used approximately for real numbers for the adjustment \( s_i \). Regarding the required precision \( p \in \mathbb{N} \) we apply the procedure described above to the value \( [10^p \cdot (u_i - l_i)] \) \( \forall i \in M \). After obtaining values \( e_{ik_i} \), we divide them by \( 10^p \). Obviously, these transformed values suffice condition ii), see Müller-Bungart [21]. The number of additional variables is now \( d_i = \lceil \log_2([10^p (u_i - l_i)] + 1) \rceil \). Therefore, the approximation and thus the quality of the upper bound can be improved arbitrarily by increasing \( p \).

The reformulation, namely eGAP\(_{bin} \), can now be represented as follows:

\[
\sum_{i \in M} c_i u_i + \min \left\{ \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij} - \sum_{i \in M} \sum_{k_i \in D_i} c_i e_{ik_i} y_{ik_i} \right\} \tag{6}
\]

s.t.
\[
\sum_{j \in N} a_{ij} x_{ij} + \sum_{k_i \in K_i} e_{ik_i} y_{ik_i} \leq b_i + u_i \quad \forall i \in M \tag{7}
\]

\[
\sum_{i \in M} x_{ij} = 1 \quad \forall j \in N \tag{8}
\]

\[
x_{ij} \in \{0, 1\} \quad \forall i \in M, \forall j \in N \tag{9}
\]

\[
y_{ik_i} \in \{0, 1\} \quad \forall i \in M, \forall k_i \in D_i \tag{10}
\]
Since we reduce the solution space to discrete values of $\tilde{s}$ within $[0, u_i - l_i]$, where the domain of $\tilde{s}_i$ is a subset of the domain of $s_i$, the solution obtained via $\text{eGAP}_{\text{bin}}$ is a feasible solution for the original model eGAP, too. Thus, the solution to $\text{eGAP}_{\text{bin}}$ is an upper bound for the optimal solution to eGAP. Given an optimal solution $(x, \tilde{s})$ to $\text{eGAP}_{\text{bin}}$, the upper bound can be strengthened if $b_i + l_i < \sum_{j \in N} a_{ij} x_{ij} < b_i + u_i - \tilde{s}_i$. This appears, if the necessary adjustment is not presentable by binary coding regarding the precision $p$. Then, $\tilde{s}_i$ can be increased to $[\tilde{s}_i]^- = b_i + u_i - \sum_{j \in N} a_{ij} x_{ij}$. If we analyze the gap between the upper bound by $\text{eGAP}_{\text{bin}}$ and the optimal solution to eGAP we see that, this gap can not be larger than $\sum_{i \in M} c_i \cdot 10^{-p}$, see Büther [5]. Thus we can always obtain an upper bound (and a corresponding heuristic solution) which is arbitrarily close to the optimal objective value by increasing $p$. Obviously, if all coefficients of the jobs in the capacity constraints are integer numbers, the adjustment will only take on integer numbers, too. Therefore, with $p = 0$ our reformulation will always deliver an optimal solution.

The advantage of using the binary coding is straightforward: first, the obtained bounds for the $\text{eGAP}_{\text{bin}}$, are not worse than that of the eGAP itself using the commercial solver ILOG CPLEX 10.1 with standard settings and the same time limit, see Büther [5]. Additionally, we can use all standard approaches already available for the GAP one-to-one, which is the basic idea of this paper.

4 The Algorithm

Since we reduced the eGAP to a standard GAP, we can use the methods, that have already been proven to be efficient for the GAP one-to-one. Therefore, the basic framework of our algorithm is beam search, a truncated branch&bound algorithm, which needs strong lower as well as upper bounds on the objective function value. Thus, we add Lagrangian relaxation to strengthen the lower bounds, see Baker and Sheasby [2] and local search procedures to generate feasible solutions, the upper bounds, see Díaz and Fernández [9]. All components will be described in detail in the following sections. At the end we will present the overall heuristic with all necessary parameter settings.

4.1 Beam Search

A common way in the optimization community for solving integer problems is the branch&bound algorithm. The classic version for a binary optimization problem starts on the lp-relaxation of the problem whose objective is a lower bound for a minimization problem. For obtaining the lp-relaxation, the binary variables are replaced by continuous ones, bounded by 0 and 1. If not all variables in the optimal solution of the lp-relaxation are integers, for the next level the problem is split into two subproblems, one requiring, that a fractional variables is 1 and the other that it is 0. The current subproblem is removed from further calculations. This is continued until no more variables are available for fixing in any subproblem. This algorithm can be represented by a binary tree, where a node illustrates a subproblem and an arc represents the fixing of a variable. Obviously, this tree might get extremely large. For a problem having $n$ binary variables, in the worst case we have to solve $2^n + 1 - 1$ subproblems. In practice, based on the following properties of a solution for a node, the number of subproblems can be reduced significantly:
i Its lower bound is higher than the best known feasible solution. The same will yield for any node obtained from the current one. No new better global upper bound can be reached on this perch.

ii The solution for the subproblem is feasible for the whole problem. No more branching is necessary.

iii The subproblem has no feasible solution. The same will yield for any node obtained from the current one, because every successor is based on the current subproblem with even more constraints, limiting the solution space furthermore.

In all three cases, the perch of the tree stops in this node and no further subproblems will be generated.

Nevertheless having high-end hard- and software, the enormous tree size cases memory lacks which terminates the optimization process, if the branch&bound algorithm is performed for the eGAP, see Büther [5]. To handle this memory problem we use a beam search algorithm in this work.

Beam search is a heuristic method for solving large combinatorial problems by efficiently exploring a search tree. It was first used in the artificial intelligence community for the speech recognition, see Lowerre [19], and image understanding problem, see Rubin [27]. Since today it has been adapted to several applications like job scheduling, see Chang et al. [8], or product line design and selection, see Nair et al. [22].

Beam search is a truncated version of the classic branch&bound algorithm, see Valente and Alves [29]. While for pure branch&bound, in the worst case the whole search tree has to be examined, this tree is limited by beam search to accelerate the optimization process: Not all potential subproblems of a node are generated but only the $\alpha$ most promising ones.

Therefore, first of all, we need an alternative branching rule, because branching on a single binary variable can only deliver at most two successor nodes. Thus, if we obtain in a subproblem a non-ambiguous assignment of a job to one agent, e.g. fractional variables for the lp-relaxation or multi assignment for the Lagrangian relaxation, see the next section, we can alternatively generate $m$ descendant nodes. Due to (3) in the eGAP, fixing one arbitrary binary variable for a job $j'$ to 1, $x_{ij'} = 1$, all other variables regarding the same job can be set to zero, $x_{ij'} = 0 \quad i \in M \setminus i'$. Instead of generating 2 successors for each node, now, one node generates $m$, each correspond to the fixing of the job to another agent. Obviously, this results into a tree with much more nodes on each level and less levels overall: only $n$ level instead of $m \cdot n$ in the ordinary case. Here again, if the obtained lower bound of a node is higher than the cost of the best feasible solution, we can prune this node off at once, based on the rules mentioned above. By always generating $m$ successor nodes, probably most of these nodes will not be very promising regarding the objective function value.

Hences, for our beam search, we limit the number of successors of a node to $\alpha$ instead of generating all $m$ ones. The result is, that we only fix a job to the $\alpha$ most promising agents. Additionally, the width of the search tree is limited by $\beta$, the maximum number of nodes at each level that are then used in the next level for further branching. All other generated nodes at a level, at most $\alpha \cdot \beta - \beta$, are pruned off.

The result of using beam search is an almost constant memory demand. Obviously, limiting the search tree reduces the computational time by accelerating the exploration of the (partial) tree.
but might prevent us from reaching the optimal solution. Thus, it is essential to have a good evaluation function to generate the $\alpha$ successors and to select the $\beta$ most promising nodes for the next level to guarantee good solutions while keeping computational efforts as small as possible. Apparently, by setting $\alpha$ and $\beta$ sufficiently high, we get a search tree covering the whole solution space. This would be similar to the classical branch&bound algorithm with the same branching rule.

One drawback of beam search is, that it cannot recover from a wrong decision. If a node and thus a sub-tree is pruned off, where the optimal solution belongs to, this solution can never be found. Hence, it is not guaranteed that the optimal solution will ever be found. But by setting the parameters appropriate, this drawback can be minimized. Additionally, by using a feasible solution generator in combination with an improvement phase, it is getting even more probable to reach the optimum, see the following sections.

For the beam search, just like for classic branch&bound, we need strong lower and upper bounds on the objective function value. Therefore, we implement Lagrangian relaxation for the lower bounds and local search procedures for the upper bounds, which we will present in the following sections. The generation of the successor nodes and the evaluation function will be presented afterwards in the overall heuristic.

### 4.2 Lagrangian Relaxation

Lagrangian relaxation can be found in many Operational Research literatures, see Fischer [10], Geoffrion [12] or Beasley [3]. The basic idea is to move one or more constraints of a problem into the objective, weighted by Lagrangian multipliers. Thereby, the erstwhile constraints can be violated during the optimization process which results into a modified objective function value in comparison to the original problem. Since we extend the solution space and if we define the Lagrangian multipliers appropriate, the objective function value of the obtained solution is a lower (upper) bound for a minimization (maximization) problem.

In our work, we relax constraint (8) with the Lagrangian multiplier vector $v$ containing the values of $v_j$, $j \in N$. Now, using the so called Lagrangian relative cost, see Yagiura et al. [31]:

$$c_{ij}(v) = c_{ij} - v_j,$$

the relaxed formulation of (6) to (10), namely $L(v)$, can be represented as follows:

$$\sum_{j \in N} v_j + \sum_{i \in M} c_i u_i + \min \left( \sum_{i \in M} \sum_{j \in N} c_{ij}(v) x_{ij} - \sum_{i \in M} \sum_{k_i \in D_i} c_i e_{ik_i} y_{ik_i} \right)$$

$$\text{s.t. } \sum_{j \in N} a_{ij} x_{ij} + \sum_{k_i \in K_i} e_{ik_i} y_{ik_i} \leq b_i + u_i \quad \forall i \in M$$

$$x_{ij} \in \{0,1\} \quad \forall i \in M, \forall j \in N$$

$$y_{ik_i} \in \{0,1\} \quad \forall i \in M, \forall k_i \in D_i$$

Obviously, the problem (12) to (15) can be decomposed into $M$ independent knapsack problems, see Nauss [23]. Additionally, as we use the reformulation $\text{eGAP}_{bin}$, the problem is even a standard binary knapsack problem.
The corresponding Lagrangian dual is to maximize the lower bound $L(v)$

$$\max L(v)$$

$$v_j \in \mathbb{R} \quad \forall j \in N$$

A fast method for finding near optimal values for $v_j$, $j \in N$, is the subgradient optimization, see Fischer [10] and Held et al. [17], which is based on the subgradients:

$$s_j(v) = 1 - \sum_{j \in N} x_{ij}, \quad j \in N.$$  

This method generates a sequence of vectors, $v^{(0)}, v^{(1)}, v^{(2)}, \ldots$, where $v^{(0)}$ can be defined arbitrarily and $v^{(t)}$, $t \geq 1$, is obtained by:

$$v_j^{(t+1)} = v_j^{(t)} + \lambda \frac{UB - L(v^{(t)})}{||s(v^{(t)})||} s_j(v^{(t)}) \quad \forall j \in N.$$  

Therefore, we use in (18) the current optimal assignment for $L(v^t)$ to obtain the new values $v^{t+1}$. Additionally, UB is the best upper bound on (1) and $\lambda$ is a step size parameter fulfilling $0 < \lambda \leq 2$, which is adapted during the optimization. Initially, it is set to two and halved, whenever the lower bound $L(v^{(0)})$ could not be increased during a given number of consecutive iterations. This treatment ensures a more and more accurate adjustment of $v$. The subgradient optimization stops, when either $\lambda$ is getting too small or the maximum number of iterations is exceeded.

If our Algorithm, the generation of the initial vector $v^{(0)}$ at the root node of beam search is handled as follows: We set $v_j^{(0)} = 0$, $j \in N$, see Yagiura et al. [31]. We tested other starting values, too, e.g. the optimal dual variables of the corresponding constraints to (8) in the lp-relaxation of (6) to (10) or the second smallest $c_{ij}$ for each $j \in N$. But by running a large number of iteration, the finally obtained lower bounds were almost similar. Thus, we finally used initially a value of 0 for all multipliers for the root node to save computational time. For all other nodes during beam search, we use initially the best Lagrangian multiplier found for the preceding node by the subgradient optimization, which is available at hand. Thereby, we try to use the information of the previous nodes to avoid too many iterations of subgradient optimization in the current node.

Solving $L(v)$ requires solving $m$ independent standard binary knapsack problems, where each can be handled quite quickly, see e.g. Pisinger [26]. But anyway we had to limit the number of iterations of subgradient optimization strictly. For the root node, we took a relatively high iteration limit of 300 because the obtained lower bound is the global lower bound and it will be used to evaluate all generated feasible solutions. Additionally, in the root node we generate the first appropriate Lagrangian multipliers, that are then used initially for the following nodes. This enables us to reduce the iteration limit for all nodes except the root node from 300 to 50, which fastens the exploration of the whole search tree. Regarding the computational results we noticed, that an increase on the lower bound $L(v)$ would be very small in comparison to the required computation times if we increased the iteration limit above 50, except for the root node.

It is well known, that the convergence of the subgradient optimization might be arbitrary poor in the first iterations, see Held et al. [17]. Thus we tried to accelerate and stabilize the convergence by exponential smoothing, see Aghezzaf and Artiba [1] and Baker and Sheasby.
and for the basics of exponential smoothing see Brown et al. [4]. Therefore, we do not only use the subgradient $s_j(v(t))$ in (19) but an exponentially smoothed version denoted by $	ilde{s}_j(v(t))$, where

$$\tilde{s}_j(v(t)) = (1 - \mu) \cdot s_j(v(t)) + \mu \cdot s_j(v(t)), \quad t > 1. \quad (20)$$

The basic idea is to use all information obtained so far in the previous iterations of the subgradient optimization for the adaption of the actual multipliers $v$. The underlying theory is based on the dual cuts for (16) to (17) derived from a solution of $L(v)$. These cuts are then combined to build a surrogate constraint. For the surrogate multipliers we use exponential smoothing, because this enables us to put more importance on the latest dual cut. For further details, we only want to refer to Baker and Sheasby [2] here. They noticed for the GAP, that using $\{i = 0.3 \text{ and } \tilde{s}_j(v(t)) \}$ instead of $s_j(v(t))$ avoids large variations for the multipliers $v$ during the first 100 iterations, which results in stronger lower bounds $L(v)$ in these iterations. Since our beam search approach works on the $eGAP_{bin}$, a standard GAP, we can adapt this concept one-to-one. Hence, we set the iteration limit for each node except the root node to 50 and use the same $\mu$ obtaining good results. For the update of $v(0)$ we set $\mu = 1$, because at this point we do not have any $\tilde{s}_j(v(0))$ so far.

### 4.3 Local Search

Local search is a well known procedure to explore the solution space for a given problem by iteratively examining the neighborhood of a solution, see e.g. Glover et al. [16]. In general, the neighborhood of a given solution is defined by these set of different solutions, which can be gained out of the initial one by using predefined operators, see Díaz and Fernández [9]. In our case, we use the shift and the swap operator:

- The shift neighborhood defines the set of solutions, which can be obtained by shifting one job from the agent, it is currently assigned, to another agent.

- The swap neighborhood defines the set of solutions, which can be obtained by interchanging the assignment of two jobs that are originally assigned to different agents.

One difference between both neighborhoods is, that executing the swap operator does not change the number of jobs actually assigned to an agent. In contrast, for the shift neighborhood the number of jobs actually assigned to an agent decreases for the old agent and increases for the new one. Of course, there are even more complex neighborhood definitions available, see e.g. Yagiura et al. [31], which have not been used in this work so far.

Note for the eGAP, if we have no restrictive upper bound $u_i$, $i \in M$, for the capacity adjustment, then every assignment, which is feasible regarding (3), is feasible for the whole problem. Altogether we have $m^n$ feasible solutions and every shift or swap operation will always generate a new feasible solution from an existing one because our neighborhood definitions do not change the sum in constraint (3). This implies an increase of the set of potential neighbors in every iteration, because we can not exclude solutions due to infeasibility like for the classic GAP, see e.g. Yagiura et al. [31]. Obviously, the generation of the neighbors is very easy, because we do not have to care about capacity limits, but the obtained upper bounds might be arbitrary poor. Since the instances used for the computational results of this work
only lower bounds on the capacity adjustments, see section 5, we limit our neighborhood to shift and swap to control the computational efforts and to maintain overall efficiency.

Additionally, we use a first fit strategy to accelerate the search. That is, we do not generate the whole neighborhood for a given solution. Instead, we check for each job, if a shift or a swap can decrease the actual upper bound. Therefore, the cost for a neighbor are always calculated based on the assignment cost as well as on the capacity adjustment cost. If such a job is found, we execute the operation and start another round of local search. The order of examining the jobs is handled as follows: first, all jobs are sorted by decreasing value $A_{c_f}$, where

\[ A_{c_f} = -\min_{i} c_{ij} \] 

and

\[ c_{ij} = \rho \cdot c_{ij} + (1 - \rho) \cdot a_{ij} \cdot c_i . \] 

The cost $c_{ij}$ are a combination of the assignment cost $c_{ij}$ and the cost that would occur, if the capacity had to be adjusted for this job. By defining the factor $\rho$ appropriate, we try to reflect the coherence between both cost. In our work, we used $\rho = 0.7$. Thus, we put more attention on the assignment cost which is reasonable because of the available capacity for each agent in the selected instances. The value $\Delta c_{ij}$ represents the cost difference for a job between the cost of the assignment to agent $i$ and the cheapest assignment, based on $c_{ij}$. This can be seen as potential cost savings, if the job is currently not assigned at its smallest cost.

By using this ordering, decreasing values of $\Delta c_{ij}$, we try to execute the most promising jobs first. If, after handling all jobs, no improvement on the current upper bound can be identified, then the best available operation is performed. That is, the job, which causes the smallest temporary increase on the upper bound, will be shifted or swapped, respectively.

Obviously, if we temporary increase the objective function value, we have to avoid to return to the previous solution when examining the neighborhood of the new one. Therefore, we adapt the concept of tabu search, a short term memory phase, see Glover [13] and Glover [14]. That is, the neighborhood for the local search is restricted by automatic mechanism to leave a local optimum and to prevent cycling. E.g. if we shift a job or swap two jobs, respectively, the returning to the previous agent is prohibited for these jobs, it is set tabu, for a given number of iterations. This is done in every iteration of the local search to prevent potential cycling, even if we were able to decrease the upper bound.

### 4.4 Overall Heuristic

At the beginning of our heuristic, we compute greedily a lower and an upper bound. Both are generated by assigning each job to one agent at cheapest cost. Therefore, the used cost differ: For the lower bound, we set $x_{ij} = 1$ and $x_{ij} = 0$, $i \in M \setminus i'$, where $i' = \arg \min_i c_{ij}$. That is, we ignore the capacity adjustments and simply assign each job to its smallest assignment cost. Then, the lower bound is computed by only summing up the assignment cost $\text{LB} = \sum_{i \in M} \sum_{j \in N} c_{ij} x_{ij}$. This is admissible, because for the ignored capacity adjustments the cost $c_i$, $i \in M$, as well as the lower bounds $l_i$, $i \in M$ are both greater-than-or-equal to zero. Thus, the product $c_i \cdot s_i$ in (1) will always be greater-than-or-equal to zero, too. This implies, that the cheapest assignment cost without the capacity adjustment cost deliver a lower bound
on the objective function value. For the upper bounds, we use the cheapest combination of assignment and capacity adjustment cost, that is, we set \( x_{ij} = 1 \), where \( i' = \arg \min_i c_{ij}^p \) and \( x_{ij} = 0 \), \( i \in M \setminus i' \). Afterwards, the potentially occurring extensions are calculated to obtain a feasible solution and the upper bound is obtained by using the objective (1) straightforward. Then, this bound is improved by local search to get a stronger one, which is used initially for the adjustment of the multipliers \( v \) during the subgradient optimization.

Since we relaxed constraint (8) in a Lagrangian way to obtain \( L(v) \), the solution of \( L(v) \) probably will not be feasible regarding the original problem. Thus, during the subgradient optimization, we generate for each obtained solution \( x \) of \( L(v) \) greedily a feasible solution by the following rules:

i First, for each job, that is assigned once, constraint (8) is fulfilled, we keep the job at the corresponding agent.

ii Then, for jobs that are multi-assigned, that is \( \sum_{i \in M} x_{ij} > 1 \), we assign the job to this agent, that belongs to the multi-assignment, the corresponding \( x_{ij} \) is one, and that generates the lowest increase in the cost, regarding the direct assignment cost as well as the cost for adjusting the capacity.

iii Finally, these jobs, that have not been assigned in \( L(v) \), \( \sum_{i \in M} x_{ij} = 0 \), are assigned following the same rule as the multi-assigned jobs, except that the set of potential agents is extended to all available agents \( i, i \in M \).

The obtained feasible solutions are then improved with local search, only if we have found a new local upper bound. During local search for any node beside the root note we regard the jobs already fixed at this node. If we then find a new global upper bound, too, we start an additional round of local search ignoring the already fixed jobs trying to strengthen this bound furthermore. This approach enables us to explore a large solution space by always using different initial solutions for the local search at each node without limiting the solution space strictly to avoid missing the optimal assignment. Thus, it is more probable to leave a local optimum and to reach the global optimum. Otherwise, if we always ignore the fixed jobs of the corresponding node, we might always return to a local optimum starting from different solutions.

For the beam search, we set \( \alpha = 2 \), thus each node can generate two successors. These successors are identified during the subgradient optimization by the way. Assume, by solving \( L(v) \) we get a new best lower bound for a specific node. Then, the jobs \( j \), that have been assigned twice, to agent \( i'_j \) as well as to \( i''_j \), that is \( x_{i'_j} = x_{i''_j} = 1 \), are potential jobs for being fixed. For the next level, we fix one job to agent \( i'_j \) and to \( i''_j \), respectively, corresponding to the two successors. For identifying the most promising job for fixing, we check, which job causes the smallest temporary increase in the lower bound if it is fixed to one of the two agents, priced by \( \text{val} \), where

\[
\text{val} = c_{i'_j}(v) + c_{i''_j}(v) + \min\{c_{i'_j}(v), c_{i''_j}(v)\}.
\]

The motivation for this function is straightforward, see table 1: Assigning the job to the two agents alternately causes additional costs and increases the lower bound because only jobs with negative reduced cost \( c_i(v) \) have been set to one before. The sum of both cost is a good indicator for the influence of this particular job. Additionally, we took the smaller of
both twice into account. This avoids fixing a job with one low and one high cost. Otherwise, e.g. fixing job 4 in table 1 to agent $i'_4$ causes a high increase in the cost and this will probably result into a node for the next level, which is pruned off at once due to the high increase in the lower bound. In this case, fixing job 4 might only generate one successor for the next level, which is not desirable for exploring the search tree efficiently.

<table>
<thead>
<tr>
<th>Table 1: Evaluation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>job</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

If no job is assigned twice, we analyze the jobs that are multi-assigned. Therefore, we apply (23), too, using the two smallest relative costs (11) for each job. If there exists no multi-assignment, too, all jobs are either assigned once or not assigned at all. There have to be some jobs $j'$, that are not yet assigned, that is $\sum_{j \in M} x_{ij'} = 0$, because otherwise the solution of the corresponding subproblem is already feasible for the whole problem and this node can be pruned off. Now, for each job $j'$ we select the agent $i'$, which causes the lowest increase in the lower bound, that is $i' = \arg\min_i c_{i'j'}(v)$. Finally, the job $j$ for fixing is identified by $j = \arg\min_{j'} c_{i'j'}(v)$. This selection of jobs will obviously deliver only one successor for this node. But regarding our computational results, this cases never arrived.

After executing the subgradient optimization, we limit the beam width to $\beta = 4$. This is done by taking these 2 nodes with the smallest local lower bound so far and start on the next level with the successors of them. Then, for each successor node the problem is reduced by eliminating the job that had just been fixed. Thus the problem size decreases from iteration to iteration by one. Obviously, the maximum number of beam search iteration is $n$ in our case, because then each job is fixed to an agent.

The beam search algorithm will be continued until either no more nodes are available because the corresponding lower bounds are too high regarding the best feasible solution or the iteration limit of 10 is reached. All parameters have been tested with other values, too. For the used instances the chosen values point out the potential of the heuristic very well regarding the computational time and the obtained bounds, see the next section for further details.

5 Computational results

After giving the theoretical background in the previous sections, we now present some computational results to show the efficiency of the developed beam search heuristic. The algorithm is coded in C++ and executed on an Intel Pentium D processor with 1GB RAM and 2.80 GHz clock rate running under Windows XP. We used the commercial branch&bound solver ILOG CPLEX 10.1 with standard settings to solve the eGAP itself. The corresponding results are carried over from our previous work, see Büthe [5], and they are now used to compare the overall performance of our algorithm.

There are five well-known types of benchmark instances for the GAP, called types A, B, C, D, and E, see Yagiura et al. [31], with 5 to 80 agents and 100 to 1600 jobs. These instances have
been modified to eGAP according to Nauss [24] and Büther [5]. The capacity adjustments are only limited by lower bounds $i_i = 0$ for each $i \in M$. In this paper, we only focus on the instances of type D, because, when using CPLEX on these instances with a time limit of 3600s, the solver runs in 10 of 15 cases out of memory, where our heuristic is still working without any problems. Note, that all coefficients for the capacity consumption are integers for these instances, thus solving the binary reformulation of the eGAP can always deliver an optimal solution, see Büther [5]. Of course we tested our algorithm on easier instances, belonging to type A to proof the correct function before. As these instances are too easy to solve for CPLEX, we limit our comparison in this work to type D.

As we have shown in section 4.2, the problem $L(v)$ can be decomposed into $m$ standard knapsack problems. If we had only integers for all parameters of the problem, we could use the well-known combo algorithm straightforward, see Büther and Briskorn [6] in combination with Martello et al. [20]. In our case, for the occurring knapsack problems in $L(v)$ we use combo, too, although the Lagrangian multipliers $v$ are not integers in contrast to all other coefficients. Therefore, in advance we multiply all coefficients of the objective function by 1000 and during the subgradient optimization, the new multipliers $v$ are rounded down to the next integer value, see Kellerer et al. [18]. This would be the same like working with real numbers, limit to 3 decimal places. Additionally, since the subgradient optimization is only a method for finding near optimal values for $v_j, j \in N$, see Fischer [10], we might never reach the optimal multipliers, even if we use real numbers with unlimited decimal places. By applying our rounded integers and combo, we accept loosing a very little solution quality but we are able to reduce the computation time significantly, see Büther and Briskorn [6].

Before executing combo, we can reduce the problem size by fixing some variables $x_{i'j'}$ to zero. If the relative cost $c_{i'j'}(v)$ are larger or equal to zero, then job $j'$ will not be assigned to agent $i'$ in this subproblem, because the objective is to minimize the cost. Additionally, we can fix these jobs $j'$ to agent $i'$, $x_{i'j'} = 1$, where $\frac{c_{i'j'}(v)}{a_{i'j'}} < -c_i$ and reduce $b'_i$ by $a_{i'j'}$. In this case, the cost decrease per unit of capacity consumption of job $j'$ is higher than the cost increase per unit for (potentially) extending the capacity, see Büther and Briskorn [6]. This can be done straightforward, because the capacity adjustments have only lower bounds in our instances.

The computational results can be found in table 2, where the first column identifies the instance. Then, there occur altogether 6 columns. The first three present the results of our previous work, see Büther [5], where CPLEX is used with a time limit of 3600s. $LB$ and $UB$ are the best obtained lower and upper bound, respectively. The column, headed by $oom$, takes on the value of one, if CPLEX runs out of memory. The next three columns represent the results of our beam search heuristic. Therefore, the meaning of the first and the second column is similar to that used for CPLEX and give the best obtained lower and upper bound. The last column shows the time, totally needed for our algorithm.

Knowing well, that the chosen instances are just a small sample, we can already observe the following:

- In 11 of 15 cases, our heuristic delivers the better lower bound than CPLEX. Note that in the other 4 cases, CPLEX did not run out of memory and thus could tighten the lower bound, primarily obtained by the Ip-relaxation, by adding additional constraints, called cuts, see e.g. Nemhauser and Wolsey [25]. Of course, we might improve our lower bounds furthermore in the future by using similar techniques, too.
Table 2: Comparison of CPLEX and our beam search heuristic

<table>
<thead>
<tr>
<th>instance</th>
<th>CPLEX</th>
<th>beam search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LB</td>
<td>UB</td>
</tr>
<tr>
<td>d05100.mps</td>
<td>6351.29</td>
<td>6353.34</td>
</tr>
<tr>
<td>d05200.mps</td>
<td>12739.13</td>
<td>12744.00</td>
</tr>
<tr>
<td>d10100.mps</td>
<td>6330.70</td>
<td>6352.41</td>
</tr>
<tr>
<td>d10200.mps</td>
<td>12420.48</td>
<td>12443.00</td>
</tr>
<tr>
<td>d10400.mps</td>
<td>24956.99</td>
<td>24975.00</td>
</tr>
<tr>
<td>d15900.mps</td>
<td>55400.66</td>
<td>55432.00</td>
</tr>
<tr>
<td>d20100.mps</td>
<td>6148.87</td>
<td>6233.56</td>
</tr>
<tr>
<td>d20200.mps</td>
<td>12219.77</td>
<td>12280.00</td>
</tr>
<tr>
<td>d20400.mps</td>
<td>24553.27</td>
<td>24613.80</td>
</tr>
<tr>
<td>d30900.mps</td>
<td>54829.13</td>
<td>54929.26</td>
</tr>
<tr>
<td>d40400.mps</td>
<td>24347.95</td>
<td>24464.00</td>
</tr>
<tr>
<td>d60900.mps</td>
<td>54551.00</td>
<td>54847.72</td>
</tr>
<tr>
<td>d201600.mps</td>
<td>97821.43</td>
<td>97865.00</td>
</tr>
<tr>
<td>d401600.mps</td>
<td>97105.00</td>
<td>97263.20</td>
</tr>
<tr>
<td>d801600.mps</td>
<td>97034.00</td>
<td>97287.75</td>
</tr>
</tbody>
</table>

• For the upper bounds, so far, CPLEX outperforms our heuristic. Obviously, this is based on the strict parameter settings for our local search as well as our beam search. By redefining these parameters appropriate, we can extend the whole search and thus increase the probability of finding better or the best upper bound, especially in these cases, where CPLEX stops due to memory lacks.

It must be pointed out, that CPLEX already needs more than one minute just to provide the lower bound by LP relaxation for these instances with a high number of binary variables $x$. At this point, no upper bounds are available. In contrast, our approach delivers in each iteration of the subgradient optimization parallel a lower and an upper bound. Thus we generate a large set of feasible solutions which can be used for advanced search procedures to find the optimal assignment, see e.g. Glover [15]

Summarizing the computational results, they show on the one hand the efficiency of the commercial solver and on the other hand the potential of our beam search heuristic. Even with limited parameters settings for our heuristic appropriate results can be obtained in reasonable time.

6 Conclusions and Outlook

In our previous paper, see Büthe [5], we already gave the theoretical background for this work. In this paper, we have presented a beam search algorithm for the eGAP, which uses Lagrangian relaxation and local search to strengthen the bounds on the objective function value. All techniques are applied on the modified variant of the eGAP, called eGAP$_{bin}$, which contains a binary representation of the capacity adjustment. It can be shown, that this problem can be reduced to a standard GAP, see Büthe [5], which enables us to use one-to-one the standard approaches mentioned above.

Summarizing the computational results, in most cases, the obtained lower bounds by Lagrangian relaxation are already better than that, delivered by CPLEX. For the upper bounds,
it is the other way round and CPLEX gives the better bounds. But, since we limit the search
tree by using beam search and do not run out of memory, in the future we can modify our pa-
rameters to enlarge the beam search as well as the local search and thus, to improve our upper
bounds furthermore. Especially by increasing $\alpha$, the number of successors of each node, we
can spread the search significantly. Additionally, we can try to add further constraints, called
cuts, to the initial node, like CPLEX does, to strengthen the corresponding lower bound. For
the upper bounds, since we display a feasible solution in every iteration of the subgradient
optimization, we could apply advanced search procedures. One approach could be path re-
linking, see e.g. Yagiura et al. [32], which is a scatter search, see Glover [15] for the basics. It
combines two or more feasible solutions to generate a sequence of new solutions with hopefully
a better objective function value than the initial solutions have.

All this will be the topic of our future research. Additionally we will run our algorithm on
further eGAP instances, belonging to other classes, see Yagiura et al. [31], with capacity
adjustments limited by lower as well as upper bounds and finally with different cost for the
adjustment. There, we will try to identify hard instances, where CPLEX stops very early in
the optimization process and our algorithm can evidence its power and flexibility.

References


[5] M. Büther. Reducing the elastic generalized assignment problem to the standard gen-
eralized assignment problem. Technical report, Christian-Albrechts-Universität zu Kiel,
2007.

variable to the standard 0-1 knapsack problem. Technical report, Christian-Albrechts-
Universität at Kiel, 2007.


ing in a flexible manufacturing system. *International Journal of Production Research*, 27


