Solution methods to the machine layout problem

Rasmus Andersen

Kongens Lyngby 2006
IMM-THESIS-2006-41

Summary

In a production facility machine locations are an important factor when calculating the material handling cost.

This report reviews different methods for optimizing the placement of the machines. The methods try to minimize the distance that the materials and semi fabrics will have to move to get from one machine to another when also taking into account that a rearrangement of machines has a price.

The first method uses reduced integer programming. It locates the machines in a hexagonal graph in order to determine the relative positioning between the machines. This information is used to find a small size integer program that solves the problem.

The second method uses ant colony optimization to solve the problem. Ant colony optimization is a meta-heuristic that uses a methods similar to that of ants when these find the shortest path from their nest to a food source. This method has been implemented and tested on various problems.

The last method uses simulated annealing to solve the problem. Various neighborhood generating methods have been reviewed and tested. Different machine layout problems have been solved using simulated annealing and ant colony optimization to investigate their relative performance.

The solutions found using reduced integer programming are not as good as those found using simulated annealing or ant colony optimization. When comparing simulated annealing and ant colony optimization it is concluded that for small

problems ant colony optimization is better than simulated annealing, but for large problems it is the opposite.

Resumé

Maskinernes placering i en fabrikshal er en vigtig faktor når prisen på materiale transportering skal beregnes.

I denne raport vil forskellige metoder til at optimere placeringen af maskinerne blive undersøgt. Metoderne forsøger at minimere distancen materialerne skal transporteres for at komme fra en maskine til en anden under hensyntagen til at flytning af en maskine også har en pris.

Den første metode bruger reduceret heltalsprogrammering. Maskinerne placeres i en hexagonal graf for at få den relative beliggenhed mellem maskinerne. Denne information bruges til frembringe et reducered heltalsprogram, som løser maskinopsætningsproblemet.

Den anden metode bruger myrekolonioptimering til at løse maskinopsætningsproblemet. Myrekolonioptimering er en metaheuristik som benytter sig af teknikker baseret på de teknikker som myrer bruger til at finde den korteste vej fra myretuen til et sted med føde. Denne metode er blevet implementeret og brugt til at løse forskellige problemer.

Den sidste metode benytter simuleret udglødning til at løse problemet. Forskellige metoder til at generere nabolag er blevet undersøgt og testet. Forskellige maskinopsætningsproblemer er blevet løst med både simuleret udglødning og myrekolonioptimering for at undersøge deres indbyrdes styrker og svagheder.

Løsninger fundet ved hjælp af reduceret heltalsprogrammering ikke er lige så gode som løsninger fundet ved hjælp af simuleret udglødning eller myrekolonioptimering. Ved sammenligningen mellem simuleret udglødning og myrekolo-

nioptimering konkluderes det at simuleret udglødning er bedre hvis problemerne er store og at myrekolonioptimering er bedre med små problemer.

Preface

This M.Sc. thesis was prepared during the period from September 1th, 2005 to April 18th, 2006. The work has been carried out at the section Operations Research, at the Department of Informatics and Mathematical Modelling (IMM) at the Technical University of Denmark (DTU).

The thesis is the final requirement to obtain the degree: Master of Science in Engineering. Readers of this thesis are assumed to have basic knowledge in the area of operations research.

My supervisor is Professor Jens Clausen, who has been very helpful with guidance and critical comments during this work.

Associate Professor Bernd Dammann has been helpful on the parallel implementation of the Ant Colony Optimization meta-heuristic.

Lyngby, April 2006

Rasmus Wissing Andersen

Contents

Chapter 1

Introduction

A product often consists of several raw materials which have been processed on different machines, assembled and packed by other machines. The raw materials and semi fabrics are transported between the machines by a material handling system (MHS). The machine arrangement determines how long the materials have to travel, the material handling cost. Machines that handle materials after each other can be placed close to each other to minimize this cost. This is easy if all the materials are processed on the machines in a given order, but if the order in which the machines have to handle the materials is complex, it is a hard problem to solve.

At a given time the future demand is uncertain, but the different scenarios for the future are known. For example in a factory producing doors and windows, there may be 40% chance that doors should be produced and 60% chance that windows should be produced. This stochastic demand function can be used to estimate the expected material flow between the machines.

Over time, the demand can change implying that for each time period (day, week, month etc.) there is a different stochastic demand function. For example in week 1 the door/window ratio may be 40/60, but in week 2 it may be 50/50. When the demand changes radically the machines might have to be reorganized in order to be efficient. The right time to make this change is important for the price of the material handling vs. the cost of reorganizing the machines.

The problem of organizing the machines efficiently with respect to a stochastic demand function is called the machine layout problem (MLP) and when expanding the problem with a changing demand function it is called the flexible machine layout problem (FMLP).

Studies has shown that 15% to 70% of the total manufacturing operating expenses can be attributed to material handling, and that an effective machine layout can reduce these costs by 10%-30%[7].

1.1 Constraints

When modelling the machine layout there are both hard and soft constraints to consider. The hard constraints are that no machines overlap and that no machines are located beyond the boundaries of the factory floor. These hard constraints must be satisfied at all times. Soft constraints may be that two machines need to be separated because of noise or heat, or that a machine needs to be near a specific location because of special needs to electricity, air ventilation etc. In order to decide whether to satisfy a soft constraint or to construct a better layout a penalty must be added to the cost of the layout if a soft constraint has not been satisfied. The size of this penalty must be considered for each of the soft constraints.

1.2 The machine handling system (MHS)

An important factor when constructing a machine layout model is the material handling system, because it limits the way of organizing the machines. Some classic ways of organizing the machines are listed below (See Figure 1.1):

- Circular. A robot-arm distributes the materials between the machines. The limitation of this MHS is that the machines must be placed with their input and output at the same distance to a certain point (the robot-arm). The robot-arm can be extendable, which makes it possible to place the machines in different distances, but in this case the machines still have to be placed so they do not block the arm from reaching other machines.

- Linear single-row. A transport belt or automated guided vehicle (AGV) distributes the materials. The machines must be placed on a line so the MHS is able to deliver and pickup the materials.

- Linear double-row. A transport belt or AGV distributes the materials. This is the same as with the single-row, but with this MHS there are two lines of machines in stead of one.
- Cluster machine layout. A gantry robot distributes the materials. The machines can be placed in any locations, since the gantry robot works in two dimensions and therefore can pick up and deliver materials at any given space.

The cluster machine layout is the most flexible and is the one considered in this report.

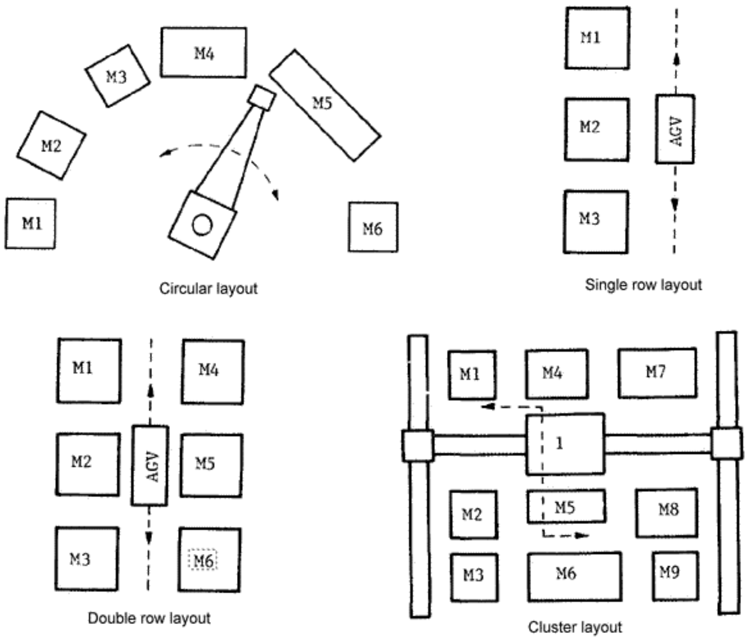

Figure 1.1: Overview of different machine handling systems

1.3 Input/Output areas on a particular machine

When modelling the MLP the location of the input and output areas on a machine must be considered. These can be modelled either as the center of a machine or as given points on the machine. It is simpler to model them as the center of the machines and doing so makes the problem easier to solve.

Modelling them as certain points on the machines makes it more precise at the cost of higher complexity in finding solutions.

In this report the center of the machines will be used as the input/output area, which is justified given the size of the machines relative to the entire floor [5].

When calculating the cost of material handling, the distance from the output on one machine to the input on another is multiplied by the amount of material that is moved between the machines. It may be more expensive to move heavy materials than light materials, therefore a material-movement cost can be multiplied as well. The type of MHS is essential in calculating the distance. For the cluster machine layout the distance is calculated as the Manhatten distance.

1.4 The scope of the report

Yang and Peters [8] implemented the reduced integer programming method and solved two flexible machine layout problems in 1997. Their method has been reviewed and commented. In 2004 Corry and Kozan [2] implemented ant colony optimization and solved the same test problems with better results, but longer running times.

In this report the two articles are reviewed. Ant colony optimization is implemented and simulated annealing is described and implemented. Quality and running times are compared for the two methods for problems of different size.

1.5 Structure of the report

The report has been structured so chapter 1 and 2 can be read to get an overview of the machine layout problem and the different methods used to solve it. Chapter 3 to 7 contain a more detail description. In chapter 8 a description of test problems and results has been made. In chapter 9 several ways to continue the research with machine layout problems and the different solution methods has been listed. Chapter 10 contains the conclusion of the report.

Chapter 2

The different techniques to solve MLP

In the following a brief introduction to MLP solution techniques will be given. Later chapters will give a more detailed description and also give details on the implementation of the techniques.

2.1 When to change layout

The three methods chosen can be used to find good layouts for the MLP. An existing layout and the expected flow are used as input and the output is a new layout and the price for material handling plus the price for rearranging machines. The expected flow is found using the stochastic demand function. If a layout covers more than one period the expected flow is found by adding the expected flows for all the individual periods.

When deciding on how many periods a layout has to cover other methods have to be used. A change of layout can take place prior to each period. A brute force method for finding a solution if there are two periods is described below:

- Calculate the material handling cost if no change is made to the layout.

- Calculate the material handling cost and cost of machine rearrangement if a change is made before the first period. This is done by finding the expected material flow using the two demand functions and using this as an input to the layout algorithm.
- Calculate the material handling cost and cost of machine rearrangement if a change is made before the second period. The expected flow is calculated using the demand function for the second period and the initial layout is used as input for the algorithm.
- Calculate the material handling cost and cost of machine rearrangement if a change is made both before the first period and also before the second period. The layout to use in the first period is found using the demand function for the first period and the initial layout. The layout to be used in the second period is found using the demand function for the second period and the layout used in the first period.

The cost of the four calculations are compared and the solution with the lowest cost is chosen. If this method is used the layout algorithm runs $2^{n+1} - n - 2$ times where n is the number of periods.

Brute force can be used if the method for finding good solutions is fast and if there is not a large number of periods, but if this is not the case another method has to be used.

The Silver Meal lot-size (SMLS) [6] heuristic can be used in a modified version. The SMLS can be reformulated to the FMLP by recasting the inventory cost to material handling and the setup cost to machines rearrangement.

The modified SMLS works by calculating the per period cost for a machine layout covering 1 period, then for a machine layout covering 2 periods and so on. It keeps going until the per period cost no longer decreases. The layout with the lowest per period cost is chosen. Then the algorithm continues by finding how many periods the next layout has to cover and so on. The running time of the SMLS algorithm is $O(p * c(n))$, where p is the number of periods and $c(n)$ is the time needed to find a layout for a problem of size n. A more detailed description of SMLS can be found in section 7.3

2.2 Reduced integer problem (RIP)

Integer programming can be used to find optimal solutions. However, if there is a lot of integer variables the running time is high. Reducing the number of integer variables reduces the running time, but the quality of the solutions depends on the quality of the reduction. The reduction presets some of the integer variables to values found by another heuristic.

When solving the MLP using RIP, a mathematical model is constructed using the current machine layout and the expected flow between the machines.

The sum of the material flow times the price for transporting the materials plus the cost of the machine rearrangements is the objective function that needs to be minimised. The constraints are that no two machines overlap and no machines are beyond the boundaries of the factory floor.

This is a problem with $1.5n^2 + 5.5n$ integer variables, where n is the number of machines. A problem of this size does not solve efficiently, hence a reduction is needed. The reduction is done by obtaining relative positioning of some of the machines before solving the problem, which reduces the number of integer variables to $0.5n^2 + 6.5n$.

The reduction heuristic uses a hexagonal adjacency graph from the Spiral procedure [4] which gives good relative positioning of the machines. The Spiral procedure finds the relative positioning by rating the machines in three ways and then placing them one by one in a hexagonal adjacency graph.

First, the machines are rated in order of how many materials they handle.

Second, the machines are rated in pairs so the pair for which the most materials travel between the machines is rated highest and so on.

Third, the machines are rated in groups of three. The highest rated triple is the one where most materials travel between the three machines.

These ratings are used to place the machines in a hexagonal graph, which is used to reduce the integer problem by presetting the relative machine order. Figure 2.1 shows an adjacency graph produced by the spiral procedure. Machine 1 is positioned above machine 4, left of machine 8 and so on. This information is used to eliminate some of the integer variables in the original problem, making it possible to find good solutions in acceptable time.

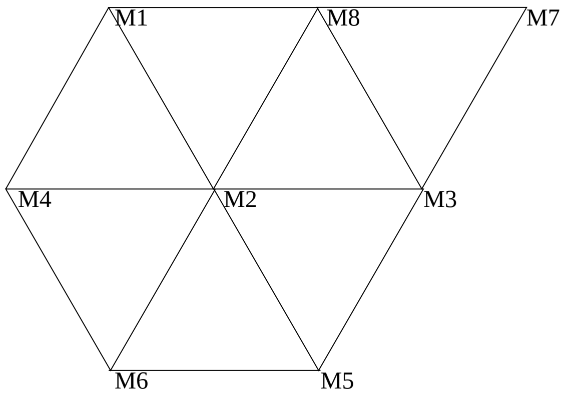

Figure 2.1: The relative positioning of machines

2.3 Ant colony optimization (ACO)

Real world ants find the shortest way to the food by laying out a pheromone scent trail and using this trail when deciding which path to follow. This is used by ACO to find good solutions. Artificial ants travel a weighted graph, where the weights represent the accumulated pheromone and a path on the graph represents a solution. The amount of pheromone an ant deposits depends on the quality of the solution it has found. Over time the pheromone trail evaporates so the ants do not get stuck on bad solutions. When an artificial ant has to move it chooses randomly between two functions. The first function uses the pheromone trail and a deterministic heuristic to choose the best way. The other uses a probability function to choose which way to go.

In order to use ACO on the MLP this has to be modelled as a weighted graph. This is done by laying a grid over the factory floor and dividing it into squares. All of these squares are nodes in the graph. Each machine is modelled as a node and there are arcs between all the machine nodes and also between each of the machine nodes and all the floor nodes. With N machines and a floor divided into $W * L$ squares there is $N * (N - 1) + N * (W * L)$ arcs in total. All the arcs must store an amount of pheromone. When an ant makes a trip it visits the machine nodes one by one. After visiting a machine node it visits the nodes representing the floor spaces that the particular machine will be placed on. The ant can only visit floor nodes that are not currently occupied by other machines. The path followed by an ant represents a solution since all the machines nodes are connected to floor nodes.

An ant only has two choices to make: Which machine to place next and where to place it.

2.3.1 Choosing the next machine to place

When the ant has to decide which machine it should place next, it can use two functions. The first is deterministic and the other is probabilistic. The ant randomly chooses which function to use and the parameter R_0 is used to decide with what probability to choose one in stead of the other.

There are two parameters that influence the choice of machine. One is the strength of the pheromone trail between the machine that the ant has just placed and the machine to place next. The other is the amount of materials that flow between the machine to place and all the machines already in place. This amount is called $H_{ord}(i,j)$ where i is the machine just placed and j is the next machine to place. These parameters are weighted with δ and β. If the deterministic function is used, the machine to place next is found by calculating the pheromone trail multiplied by H_{ord} for each machine not yet placed and choosing the one with the highest value. If the probabilistic function is used the same values are calculated, but now the values determine with what probability a given machine are chosen. When all probabilities have been calculated the machine is randomly chosen.

An ant is located at machine node i if it has just placed machine i. The variable $trail(i,j)$ is the accumulated pheromone at the trail between machine-node i and machine-node j. When machine j is to be placed the variable $H_{ord}(i,j)$ is the sum of expected flow between machine j and the machines already placed. Fre_i is the set of machines not yet placed. The machine-node to visit after node i is denoted j and is found using the following function:

$$j = \begin{cases} \max_{m \in Fre_i}\{trail(i,m)^{\delta} * H_{ord}(i,m)^{\beta}\} & if R \leq R_0 \\ J & \text{Otherwise} \end{cases} \tag{2.1}$$

δ and β are used as weights so the pheromone trail and the H_{ord} value can have more or less influence on the choice. R_0 is a number between 0 and 1 used to decide whether to use the greedily best choice or use a randomly selected node, R is a random number generated every time a machine is selected. J is a node from Fre_i and is selected according to the probability function:

$$p(i,J) = \begin{cases} \frac{trail(i,j)^{\delta} * H_{ord}(i,j)^{\beta}}{\sum_{m \in Fre_i} trail(i,m)^{\delta} * H_{ord}(i,m)^{\beta}} & \text{if } j \in Fre_i \\ 0 & \text{Otherwise} \end{cases} \tag{2.2}$$

2.3.2 Choosing where to place the machine

Now a location for the chosen machine has to be found. Again there is a deterministic and a probabilistic function. The parameters that influence the choice are the trails from the machine node to the floor nodes and the cost. If the machine is located at Lcn_i, the cost, $H_{pos}(Lcn_i)$, is the price for rearranging any machines that in the previous layout were using this location plus the material handling cost for materials traveling between the machine to place and the machines already placed. Since a machine can take up more than one floor node the average trail information at Lcn_i is used. The deterministic function calculates the average trail value divided by H_{pos} for all the possible locations and the location with the highest value is chosen. The probabilistic function calculates the same values, but again the values now represent the probability of the location being chosen and the location is chosen randomly.

If machine i is to be placed at position Lcn_i, $H_{pos}(i, Lcn_i)$ is the cost. The cost is the material handling for materials that travel between machine i and the machines already in place plus the rearrangement cost of any machines that must be rearranged because of machine j being placed in location Lcn_i. Rearrangement cost of machines already placed are is not included. $M(Lcn_i)$ is the set of floor-nodes occupied by the machine and Vct_i is the set of floor-nodes not occupied by any machines. The location of the machine is found using the following function:

$$Lcn_i = \begin{cases} \max_{M(Lcn_i) \subseteq Vct'_i} \frac{trail_{avg}(i, Lcn'_i)^{\delta}}{H_{pos}(i, Lcn'_i)^{\beta}} & \text{if } R \leq R_0 \\ LCN_i & \text{Otherwise} \end{cases} \tag{2.3}$$

$trail_{avg}(i, Lcn_i)$ is the average accumulated pheromone on the trails from the machine-node i to the floor-nodes $M(Lcn_i)$.

LCN_i is a location, such that $M(LCN_i) \subseteq Vct_i$, selected by the probability function:

$$p(i, LCN_i) = \begin{cases} \dfrac{\frac{trail_{avg}(i,LCN_i)^{\delta}}{H_{pos}(i,LCN_i)^{\beta}}}{\sum_{M(Lcn_i')\subseteq Vct_i} \frac{trail_{avg}(i,Lcn_i')^{\delta}}{H_{pos}(i,Lcn_i')^{\beta}}} & \text{if } M(LCN_i) \subseteq Vct_i \\ 0 & \text{Otherwise} \end{cases} \tag{2.4}$$

2.3.3 Pheromone trail

When an ant has found a solution it is rated and the pheromone on the trail is updated. The amount of pheromone to add to the trail is $\frac{U}{C}$ where U is a parameter controlling how much pheromone to be used, and C is the cost of the solution with respect to material handling and machine rearrangements. The value of U must be so low that the level of pheromone is not at max everywhere, but so high that the level of pheromone is not too low in interesting places.

When all ants have found solutions and deposited pheromone, the evaporation takes place. The new pheromone values of the trails are found using the following function.

$$trail_{new} = min(\tau_0, (1 - \alpha) * trail) \tag{2.5}$$

τ_0 is the maximum pheromone possible at a trail and α is used to decide how much evaporation takes place.

The ants then start over and find solutions using the new trail information.

2.4 Simulated Annealing (SA)

Simulated Annealing is an improvement heuristic. It searches the neighboring solutions of the initial solution to find a better one. A neighboring solution is one where an adjustment has been made to the original solution. Neighbor solutions are reviewed one by one. The neighbor solution to review is picked randomly from the whole neighborhood. If the neighbor solution is better than the current, it is selected and neighbor solutions of this one are reviewed. Sometimes a worse solution is selected which gives a possibility to escape a local minimum. The chance that a worse solution is chosen decreases during the run of the algorithm. The stop criteria for the algorithm can be one or more of the following:

- The algorithm has been running for a given number of iterations.

- The chance that a worse solution is selected is sufficiently small.
- The best found solution has not improved during the last n iterations.
- The best solution is better than a given value.

When solving the MLP using simulated annealing the existing layout is used as input.

Neighboring solutions are found by moving a machine from its existing location to a new feasible location, by having two machines switch locations or letting a machine change orientation. An orientation change means to turn the machine 90 degrees. This only makes sense when the length and width of the machine are different.

CHAPTER 3

The Machine Layout Problem

The machine layout problem is a problem where a number of machines must be placed in locations under various restrictions. The location of the machines are a factor when computing the price for a given layout. Another factor is the price for transporting the material. If it is very expensive to transport a certain material due to weight, fragileness, toxicness it is probably wise to place machines between which this material is transport close to each other. The same goes if a lot of material is transported between two machines. On the other hand if two machines creates a lot of heat or noise they should be separated. The goal is to find the optimal location for all the machines when respecting all constraints and minimizing the price for material handling and machine rearrangements.

A machine can have many different shapes. The machines considered in this report will, however, be rectangular, so they can be modelled as rectangles when viewed from above. A factory floor can also have many different shapes. There might be pillars supporting the ceiling and other obstacles. The floor will also be modelled as a rectangle, with no obstacles in it. The use of rectangles eases the calculations needed when making the mathematical model.

3.1 Price of a layout

When calculating the price for a layout several issues will have to be considered. If an original layout is to be improved there will be a rearrangement price for moving a machine. If the rearrangement price for at machine is relatively high it might not be profitable to move it. If constructing a layout for a new production facility no rearrangement price will be needed.

The distance that the material has to be transported when moved from one machine to another depends on the material handling system. If a transport belt is used it is the linear distance, if a gantry robot is used it is the Manhatten distance, etc.

The type of material is important when calculating the handling price as well. A million small screws cost less to transport than a million cars. A material handling weight can be multiplied with the amount of material that is been transported between two machines in order to take this into account.

The area of the machine where the materials must be dropped off or picked up can influence the price as well. The materials must be transported from one machines output area to another machines input area. This will have great effect on very large machines, but in the problems studied in this report the distance between the center and these points will be so small compared to the whole system that it has no importance[5].

The last issue that will affect the price of a layout is the soft constraints. If a machine has special needs in terms of electricity or heat dispation a certain location might be ideal for it. In some cases this is so important that it will be a hard constraint that must be satisfied in order for the layout to be feasible. In other cases there might be a price for not satisfying the constraint. An example is that for each meter that a particular machine is away from the chimney the layout will cost a certain amount more. This means that the price of the soft constraint decides whether to satisfy it in contrast to getting a cheaper material handling cost.

Chapter 4

The Reduced Integer Problem

An integer problem is an optimization problem where one or more of the variables are restricted to integer values. Integer programming is used to solve integer problems. There are four groups of integer programs.

- Mixed Integer Program (MIP). A MIP is used when some but not all variables are integers:

$$\begin{aligned}&max\{cx + hy\}\\&Ax + Gy \leq b\\&x \geq 0, y \geq 0 \text{ and integer}\end{aligned}$$

- Integer Program (IP). An IP is used when all the variables are integers

$$\begin{aligned}&max\{cx\}\\&Ax \leq b\\&x \geq 0 \text{ and integer}\end{aligned}$$

- Binary Integer Program (BIP). A BIP is used when all the variables are restricted to 0 or 1.

$$\begin{aligned}&max\{cx\}\\&Ax \leq b\\&x \in \{0, 1\}^n\end{aligned}$$

- Combinatorial Optimization Program (COP). COP is used when having a finite set of components, $N = \{1, 2, ..., n\}$. Each component has a weight c_j, $j \in N$ and a set of feasible subsets, F, of N exists.

$$\max_{S \subseteq N} \{\sum_{j \in S} c_j : S \in F\}$$

A large number of problems can be formulated and solved using these programs. The typical way to solve a problem using integer programming is to formulate the problem according to one of the above. When the problem has been defined a set of constraints can be created and entered into an interpreter like GAMS[1] which will generate a solution using a MIP-solver.

Integer programming can be very inefficient and will be very time consuming if a large number of integer variables is used. A Reduced Integer Problem (RIP) is an integer problem where some of the integer variables have been eliminated. The RIP is easier to solve because of the reduced complexity. The RIP is created by having a heuristic find good values for some of the variables, thereby converting these to constants. The quality of the final solution will however depend heavily on the quality of the reduction.

4.1 Using RIP on MLP

The integer program of the MLP is a mixed integer problem. There are variables taking continuous values and there are binary variables. Below the parameters and variables are listed and explained.

Parameters:

$$\begin{aligned}
N &= \text{The number of machines in the layout} \\
A_i &= \text{The rearrangement cost of machine } i \\
F_{ij} &= \text{The expected material flow between machine } i \text{ and } j \\
\zeta_i &= \begin{cases} 1, & \text{machine } i \text{ is in vertical position in the original layout.} \\ 0, & \text{machine } i \text{ is in horizontal position in the original layout.} \end{cases} \\
\varepsilon_{ij} &= \text{The flow weight between machine } i \text{ and } j \\
M &= \text{A very big number} \\
w_i &= \text{Width of machine } i \\
v_i &= \text{Length of machine } i \\
W &= \text{Width of the floor} \\
H &= \text{Length of the floor}
\end{aligned}$$

[1] http://www.gams.com

Variables:

$$
\begin{aligned}
Z_i &= \begin{cases} 1, & \text{Machine } i \text{ is in vertical position} \\ 0, & \text{Machine } i \text{ is in horizontal position} \end{cases} \\
(x_i, y_i) &= \text{Coordinates of the centre of machine } i \\
\alpha_{ij} &= 1 \text{ if } x_i \geq x_j \text{ and 0 otherwise} \\
\beta_{ij} &= 1 \text{ if } y_i \geq y_j \text{ and 0 otherwise} \\
\sigma_i &= 1 \text{ if } x_i \geq a_i \text{ and 0 otherwise} \\
\rho_i &= 1 \text{ if } y_i \geq b_i \text{ and 0 otherwise} \\
\tau_i &= 0 \text{ if } x_i = a_i \text{ and 1 otherwise} \\
\lambda_i &= 0 \text{ if } y_i = b_i \text{ and 1 otherwise} \\
\phi_i &= 0 \text{ if } \tau_i = \lambda_i = 0 \text{ and 1 otherwise} \\
I_i &= \begin{cases} 0, & \phi = 0 \text{ and } Z_i = \zeta_i \\ 1, & \text{otherwise} \end{cases} \\
\theta_{ij} &= \text{Binary variable used to prevent overlap between machine } i \text{ and } j \\
E_{ij} &= x_i - x_j \text{if } x_i > x_j, \text{ 0 otherwise} \\
F_{ij} &= x_j - x_i \text{if } x_i < x_j, \text{ 0 otherwise} \\
G_{ij} &= y_i - y_j \text{if } y_i > y_j, \text{ 0 otherwise} \\
H_{ij} &= y_j - y_i \text{if } y_i < y_j, \text{ 0 otherwise} \\
P_i &= x_i - a_i \text{if } x_i > a_i, \text{ 0 otherwise} \\
Q_i &= a_i - x_i \text{if } x_i < a_i, \text{ 0 otherwise} \\
R_i &= y_i - b_i \text{if } y_i > b_i, \text{ 0 otherwise} \\
S_i &= b_i - y_i \text{if } y_i < b_i, \text{ 0 otherwise}
\end{aligned}
$$

$$
\begin{aligned}
\Lambda &= \{(i,j) | i = 1, ..., N-1; j = i+1, ..., N; i \neq j\} \\
\Delta &= \{i | i = 1, ..., N\}
\end{aligned}
$$

The variables are found using the following equations which will be explained in the following.

$$\min \quad \sum_{(i,j)\in\Lambda} \varepsilon_{ij} F_{ij} [E_{ij} + F_{ij} + G_{ij} + H_{ij}] + \sum_{i\in\Delta} A_i I_i \tag{4.1}$$
$$x_i - x_j = E_{ij} - F_{ij} \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.2}$$
$$y_i - y_j = G_{ij} - H_{ij} \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.3}$$
$$E_{ij} \le \alpha_{ij} M \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.4}$$
$$F_{ij} \le (1-\alpha_{ij}) M \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.5}$$
$$G_{ij} \le \beta_{ij} M \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.6}$$
$$H_{ij} \le (1-\beta_{ij}) M \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.7}$$
$$E_{ij} + F_{ij} - \tfrac{1-Z_i}{2} w_i - \tfrac{Z_i}{2} v_i - \tfrac{1-Z_j}{2} w_j - \tfrac{Z_j}{2} v_j \ge -\theta_{ij} M \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.8}$$
$$G_{ij} + H_{ij} - \tfrac{1-Z_i}{2} v_i - \tfrac{Z_i}{2} w_i - \tfrac{1-Z_j}{2} v_j - \tfrac{Z_j}{2} w_j \ge (\theta_{ij} - 1) M \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.9}$$
$$x_i - a_i = P_i - Q_i \qquad {\scriptstyle (i,j)\in\Lambda} \tag{4.10}$$
$$P_i \le \sigma_i M \qquad {\scriptstyle i\in\Delta} \tag{4.11}$$
$$Q_i \le (1-\sigma_i) M \qquad {\scriptstyle i\in\Delta} \tag{4.12}$$
$$P_i + Q_i - \tau_i M \le 0 \qquad {\scriptstyle i\in\Delta} \tag{4.13}$$
$$y_i - b_i = R_i - S_i \qquad {\scriptstyle i\in\Delta} \tag{4.14}$$
$$R_i \le \rho_i M \qquad {\scriptstyle i\in\Delta} \tag{4.15}$$
$$S_i \le (1-\rho_i) M \qquad {\scriptstyle i\in\Delta} \tag{4.16}$$
$$R_i + S_i - \lambda_i M \le 0 \qquad {\scriptstyle i\in\Delta} \tag{4.17}$$
$$\tau_i + \lambda_i - 2\phi_i \le 0 \qquad {\scriptstyle i\in\Delta} \tag{4.18}$$
$$\phi_i + Z_i - 2I_i \le 0 \quad (\zeta_i = 0) \qquad {\scriptstyle i\in\Delta} \tag{4.19}$$
$$\phi_i + (1 - Z_i) - 2I_i \le 0 \quad (\zeta_i = 1) \qquad {\scriptstyle i\in\Delta} \tag{4.20}$$

$$\left.\begin{array}{l} Z_i, \alpha_{ij}, \beta_{ij}, \theta_{ij}, \sigma_i, \tau_i, \rho_i, \lambda_i, \phi_i, I_i \in \{0,1\} \\ 0 \le x_i, P_i, Q_i, E_{ij}, F_{ij} \le W \\ 0 \le y_i, R_i, S_i, G_{ij}, H_{ij} \le H \end{array}\right\} i \in \Delta \text{ and } (i,j) \in \Lambda \tag{4.21}$$

The objective function (4.1) is the sum of material handling costs and machine rearrangement costs and should be minimized. Equations 4.2, 4.4 and 4.5 ensure that E or F is set to the vertical distance between the centroids of machine i and j and that the other is set to 0. Equations 4.3, 4.6 and 4.7 does the same with G and H for the horizontal distance. With the help of θ, equations 4.8 and 4.9 ensure that machine i and j does not overlap in both vertically and horizontally (See figure 4.1). Equations 4.10-4.13 ensures that τ_i is set to 1 if the centroid of machine i is moved from its original position in the vertical direction. Equations 4.14-4.17 ensures the same for λ_i, but in the horizontal direction. Equation 4.18 ensures that ϕ_i is set to 1 if the centroid of machine i has moved in either the vertical or horizontal direction. Finally equations 4.19 and 4.20 will make I_i indicate if machine i has been rearranged.

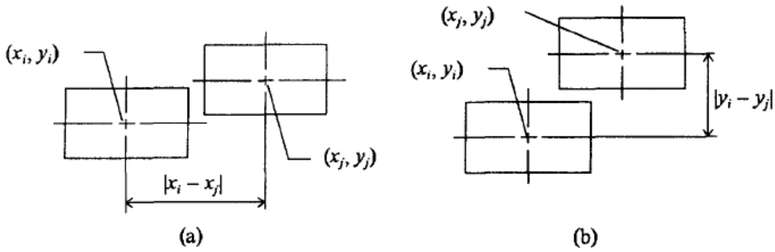

Figure 4.1: (a) shows overlap in the vertical direction. (b) shows overlap in the horizontal direction. Overlap in both at the same time would result in machine collision.

As seen in equation 4.21 there are 3 types of binary variables with the ij index and 7 with the i index. Since there are $\frac{N(N-1)}{2}$ elements in Δ and N variables in Λ there are a total of $1,5N^2 + 5,5N$ binary variables, which is far too many to allow an efficient solution of the problem to optimality. Therefore a reduction in the number of binary variables must be made.

4.2 Using the Spiral procedure to reduce the number of variables

The Spiral Procedure [4] provides a relative positioning between all the machines, which is indicated by the variables α_{ij} and β_{ij}. With α_{ij} and β_{ij} known for all (i, j), the number of binary variables is reduced to $0.5N^2 + 6.5N$. This gives a problem that can be solved efficiently if the number of machines is not very high. The spiral procedure places the machines in a hexagonal graph and uses this graph to find the relative positioning.

4.2.1 Creating the hexagonal graph

The graph is constructed by adding machines to it one by one. Each machine can have up to 6 neighbors. When evaluating a location, the flow and flow weight between the machine and all its neighbors are the only factors considered. The sum of all the weighted flows is called the adjacency score. Three lists are created to find the order in which the machines should be placed in the graph.

The first list contains tuples with one machine. The flow between a machine and all the other machines decides the order of this list. The machine with the highest material flow to other machines is at the top. This is the unary relationship list.

The second list contains tuples with two machines. The flow between the two machines decides the order of the list. The pair of machines which has the highest material flow between them is at the top. This list is the binary relationship list.

The third list has tuples with three machines. Like with the binary relationship list, the first tuple is the one where the flow between the three machines is the highest.

Now a decision must be on which of the three ratings to use. If the unary is chosen the machines are placed in the order of the unary rating. The first machine is placed in the center and when placing the following machines all locations where the machine would get at least one neighbor is evaluated. The location with the highest adjacency score is chosen. If using the binary relationship, the two machines from the first tuple is placed in the graph. The list is now scanned from the top down. The first tuple having one machine not yet in the graph and one machine in the graph with at most 5 neighbors are chosen. All free locations next to the machine already in the graph are now evaluated for the other machine. The location which has the highest adjacency score is used. If the terniary relationship is used, the three machines from the first tuple is placed. The list is then scanned from the top down. The first tuple containing one machine that has not been placed and two machines that has been placed next to each other is used. The two machines that has been placed must have a common neighbor location that is unused. The other location that the two machines have in common will logically be occupied, so the free location will be used.

When all the machines have been placed the graph is modified to obtain a local optimum. The optimization is done by trying all possible swaps of two and three machines. If a swap improves the total adjacency score it is accepted and every possibly swap is done over again.

4.2.2 Using the graph to reduce the number of variables

When the graph has been constructed and optimized, the layout of the graph is used to reduce the number of variables in the original problem. By considering the machines pairwise all α_{ij} and β_{ij} are found. If machine i is placed further left than machine j then α_{ij} is set to 0, if not it is set to 1. If machine i is placed below machine j, β_{ij} is set to 0, if not it is set to 1. This is done for all pairs of machines. In the example in figure 4.2, eight machines has been located in a hexagonal graph. Each pair of machines are now examined to determine α and β.

In this example α and β values related to machine 2 takes the following values:

- $\alpha_{12} = 0$, $\beta_{12} = 1$
- $\alpha_{23} = 0$, $\beta_{23} = 1$
- $\alpha_{24} = 1$, $\beta_{24} = 1$
- $\alpha_{25} = 0$, $\beta_{25} = 1$
- $\alpha_{26} = 1$, $\beta_{26} = 1$
- $\alpha_{27} = 0$, $\beta_{27} = 0$
- $\alpha_{28} = 0$, $\beta_{28} = 0$

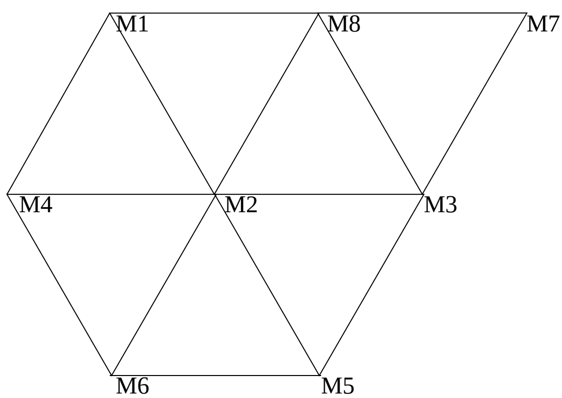

Figure 4.2: The relative positioning of machines

4.2.3 Limitations of the spiral procedure

The spiral procedure does not take machine rearrangement into account. This means that an optimal adjacency graph in terms of adjacency score, could be very expensive when used to reduce the integer problem. This has been verified in article [2], where test problems with different rearrangement prices has been solved using RIP. RIP performs better with low or no rearrangement prices. The size of the machines are not considered either. If the problem consists of a large machine in combination with several small machines, gaps between the machines can occur.

CHAPTER 5

Ant colony optimization

Ant colony optimization is a meta-heuristic for finding good solutions to optimization problems. It uses techniques similar to the technique ants use to find the shortest way from the nest to a food source. ACO was first used in early '90s, so it is a relatively new technique.

5.1 Ants in the real world

The communication between ants are based on a chemical called pheromone, which is different from humans and other higher species where the most important senses are visual and acoustic. A special form of pheromone is trail-pheromone which some ant species use for marking trails on the ground, for example from the nest to a food source. The pheromone trail is used by the ants to find the path to food discovered by other ants. The ants have a tendency to choose a route with a high pheromone scent over a route with weak scent. This behavior is the inspiration source for ACO.

A good example of how the ants use pheromone to find shortest paths is the double bridge experiment [3]. If there are only 2 paths from the nest to the food and the paths have the same length the traffic most likely converges to one of the paths. This happens because of the randomness with which the ants choose between two routes with the same amount of pheromone. At some point more ants will choose one of the routes over the other. When this has happened the pheromone trail on this route gets stronger than the trail on the other. Because of the stronger pheromone trail more ants will choose this route and thereby reinforce the trail. If one of the paths is longer than the other the traffic most likely converges towards the shortest path (See figure 5.1). When there is no pheromone on either of the trails the ants choose randomly between the two routes. The ants choosing the short path will reach the food faster and when they are returning there will only be pheromone on the short path. Because of the higher pheromone trail on the short route the ants will be likely to choose this route and reinforce the trail. This example shows that an ant colony has optimization capabilities although simpler than the artificial ants described in the latter.

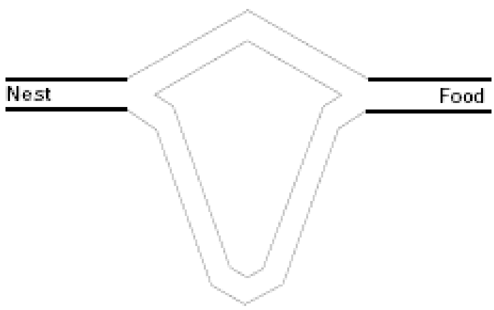

Figure 5.1: The double bridge experiment

5.2 The ACO meta-heuristic

The ants in the real world are not very intelligent. They pick a route randomly and the only thing that makes it different from complete randomness is the level of pheromone scent on the different paths. When simulating these ants, there are several changes that can be made to make the artificial ants smarter than the real ants and therefore able to solve more complex problems.

5.2.1 The mathematical model

In order to use ACO on a problem, this must have some special characteristics. It must be an optimization problem (S, f, Ω), where S is the set of candidate solutions, f is the objective function and Ω is a set of constraints. To use ACO, the problem is mapped on a problem characterized by these points:

- A finite set $C = \{c_1, c_2, ..., c_{N_c}\}$ of components is given. N_c is the number of components.
- A sequence of components is a solution or part of a solution. The list of all possible sequences is denoted χ. The length of a sequence, x, is expressed by $|x|$. The maximum length of a sequence is bounded by a positive constant $n < \infty$
- The set of candidate solutions S is a subset of χ so $S \subseteq \chi$.
- A set of feasible sequences $\tilde{\chi}$, so $\tilde{\chi} \subseteq \chi$. This subset is defined so all $x \in \tilde{\chi}$ satisfies Ω and all $x \notin \tilde{\chi}$ fail to satisfy Ω.
- A non-empty set of optimal solutions, $S^\star$, where $S^\star \in S$ and $S^\star \in \tilde{\chi}$.
- A cost, $f(s)$, associated with each candidate solution, $s \in S$. In some cases it can be an advantage to have a cost function with a sequence as input in order to get the price of a partial solution or to get the price of an infeasible solution.

Given this, the artificial ants can make randomized walks on the fully connected graph, G_c. The nodes in the graph, C, are all the components and by making a walk an ant constructs a solution.

5.2.2 Artificial ants

"An artificial ant in ACO is a stochastic constructive procedure that incrementally builds a solution by adding opportunely defined solution components to a partial solution under construction. This means that ACO can be applied to any problem for which a constructive heuristic can be defined." [3]

The opportunely defined solution components can be found using a heuristic created for the specific problem. Besides this heuristic the artificial ants has an advantage since they have exact memory of their current state. They know exactly where they have been, what choices they have made to get there etc.

This memory is used when the ant is laying out pheromone, but can also be used by the heuristic when finding the components to add to the solution.

5.2.2.1 A qualified choice of route

When an artificial ant has to make a choice of which component to add to its currents solution, it evaluates all the possible components and rates them to be able to make a more qualified choice. The heuristic for evaluating the components are problem specific and can use the knowledge of the already traveled path, the price of adding the component and the weight of the edge from the latest added component to the component to add.

The pheromone trail from latest added component to the component being evaluated is also a factor. Weights can be used so the pheromone trail can have more or less influence than the heuristic. Each component gets a score based on the heuristic and the pheromone trail.

The ant can either choose the component with the highest score or it can choose the component according to a probability function. The higher the score a component has the higher the probability is for it being chosen. A parameter decides with what probability the highest rated component should be selected over using a probability function.

5.2.2.2 Deposition of pheromone

An artificial ant is not forced to lay out its pheromone while walking as a real ant must do. The artificial ant can wait until it has finished constructing a solution and then deposit an amount of pheromone proportional with the quality of this solution. This is a big advantage since ants who have created good solutions will have more influence on the pheromone levels than ants who have created poor solutions.

When an ant has created a solution of bad quality it still deposits pheromone. In unfortunate situations this can happen several times in the first iterations, which would result in the ants getting stuck on a bad solution since this has a strong pheromone trail. To avoid situations like this the pheromone trail evaporates over time. The parameters controlling how much deposition and evaporation of pheromone that takes place must be adjusted in order to match the problem. Strong evaporation makes the search diverse, while weak evaporation will result in the search getting narrow faster. If using a lot of ants on a small graph, the amount of pheromone to deposit should be relatively small.

5.2.2.3 Elitist ant strategy

The pheromone level on the path of the best found solution might be just as strong or even weaker than other paths. This especially occurs if the best solution found differs a lot from other good solutions that have been found. A method to reinforce the path of the best found solution is to have a number of ants travel the path in every iteration. It is not unlikely that an even better solution will include parts of the yet best found solution, so reinforcing the pheromone trail of the best solution gives the ants a higher probability of using parts of this solution when making randomized walks.

5.3 ACO on the MLP

5.3.1 The mathematical model

The MLP is modelled as a construction problem in order to use ACO on it. There are two parts that needs consideration: Which machine to place next and where to place this machine. In order to use ACO the floor must be divided into discrete units, so a finite number of locations exists for each machine.

The graph that is used consists of a node for each machine and a node for each floor unit. There are edges between all the machine nodes and between every machine node and all the floor nodes.

When an ant constructs a solution it does so by selecting a machine-node and visiting it. Then it decides where to place it and visits all the floor nodes corresponding this location. It returns to the machine node and selects the next machine to place.

In figure 5.2 a solution has been constructed. The problem consists of three machines that needs to be placed on a 4 by 6 unit floor. Machine 1 has the dimensions 1,2, machine 2 is 2,2 and machine 3 is 2,3. An ant has started with machine 1, this has been placed on floor nodes (2,3) and (2,4). Then the ant has selected machine 3 and placed it. After the solution is constructed the price is calculated and the pheromone level of the path is updated. In this example the edges to update are:

- Machine node 1 to floor node (2,3)
- Machine node 1 to floor node (2,4)
- Machine node 1 to machine node 3
- Machine node 3 to floor node (3,3)
- Machine node 3 to floor node (4,3)
- ...

The number of edges to update are the number of floor units the machines cover plus the number of machines minus 1.

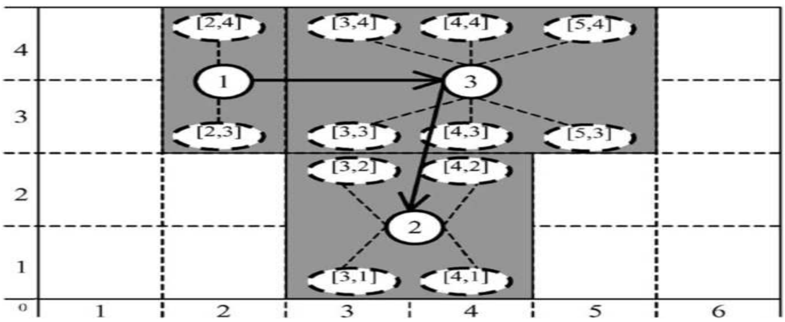

Figure 5.2: Mathematical model of ACO on MLP. 3 machines placed on a floor with discrete units

5.3.2 Finding the next machine

The order in which the machines are placed is important. The machine that is placed last will not have as many feasible location as it would have had if it were to be placed first, since other machines block a lot of locations. When creating the heuristic that calculates the order in which the machines are to be placed this is essential. It is important to emphasize that a layout is independent of the order in which the machines have been placed and that the order only has influence on how hard it is to find good layout. The machine to place next should be the most important one with respect to the machines already in place. In other words, out of the machines not already in place, the one which has the highest material flow to the machines already in place is the most important. When no machines has been placed, all machines are equally important and will take turn in being placed first.

The heuristic for rating the machines to place, when ant q has completed step s, is called H_{ord} and is described in the following:

$$H_{ord}(i) = \sum_{m \in L_q^{(s)}} \varepsilon_{mi} F_{mi} \tag{5.1}$$

where $L_q^{(s)}$ is the machines that ant q has placed after step s, F_{mi} is the flow between machine m and machine i, and ε_{mi} is the flow weight between machine m and machine i.

The pheromone trail between the machines also has an influence on the choice. The heuristic rating combined with the pheromone level between the machines gives the following function for choosing which machine to place after having placed machine i:

$$j = \begin{cases} \max_{m \in Fre_i}\{trail(i,m)^{\delta} * H_{ord}(m)^{\beta}\} & if R \leq R_0 \\ J & \text{Otherwise} \end{cases} \tag{5.2}$$

Fre_i is the set of machines not yet placed, after machine i has been placed. R_0 is a parameter used to decide if a probability function should be used or of the machine with the highest rating combined with the highest pheromone trail should be selected. R is a random number between 0 and 1 and is generated every time a machine is chosen. If $R \leq R_0$ and two or more machines have the highest value, one of them is picked randomly. If $R > R_0$ the machine is picked using the following probability function.

$$p(i,J) = \begin{cases} \frac{trail(i,j)^{\delta} * H_{ord}(i,j)^{\beta}}{\sum_{m \in Fre_i} trail(i,m)^{\delta} * H_{ord}(i,m)^{\beta}} & \text{if } j \in Fre_i \\ 0 & \text{Otherwise} \end{cases} \tag{5.3}$$

The probability with which a machine is chosen depends on the value of H_{ord} and the amount of pheromone on the trail between the machine that was just placed and the machine.

5.3.3 Finding a place for the chosen machine

Once the machine to place next has been selected, the location must be found. The machine must be placed within the boundaries of the factory floor such that it does not overlap with other machines.

The price for placing a machine in a given location is found using the flow between the machine and the machines already in place. The distance between the machine and every other machine is multiplied by the material flow between the machines and multiplied by the flow weight. If the location is not the same as in the original layout a rearrangement price is added too. If the machine is placed in a location that prohibits machines not already placed from being placed in their original locations, the rearrangement costs for these machine are added as well.

$$H_{pos}(i, Lcn_i) = \sum_{m \in L_q^{(s)}} \varepsilon_{mi} F_{mi}(|cx_i - cx_m| + |cy_i - cy_m|) + \sum_{m \in R_i} A_m + \Psi_i A_i \quad (5.4)$$

where

$$R_i = \{m | m \neq i; m \notin L_q^{(s)}; M(Lcn_m^0) \subseteq Vct_i^q; M(Lcn_m^0) \cap M(Lcn_i) \neq \emptyset\}$$

and

$$\Psi_i = \begin{cases} 1 & if M(Lcn_i^0) \subseteq Vct_i^q \wedge Lcn_i \neq Lcn_i^0 \\ 0 & \text{otherwise} \end{cases}$$

cx_i is the center of the machine on the vertical axis. This means that the distance between two machines is calculated as the 1-norm distance between the centers of the machines. Lcn_m^0 is the location of machine m in the original layout. $M(Lcn)$ is the set of floor nodes that Lcn covers.

The first sum is the cost of material handling between machine i and the machines already in place, when placing machine i in location Lcn_i.

The second sum is the price of rearranging any machine in R_i. R_i is the set of machines that have not been placed yet, and still can be placed in their original locations. Their original locations are not occupied by any of the machines that has been placed and does not overlap with Lcn_i.

The last element is the price of rearranging machine i. This is added if Lcn_i is different from Lcn_i^0 and Lcn_i^0 is not occupied by any of the machines already placed.

H_{pos} is a greedy heuristic and does not always find the optimal solution, so it will be combined with the ACO. This means that the pheromone trail between the floor nodes also has an influence on which location will be used. Since a machine can occupy more than one floor unit, the pheromone level between a machine and a location will be calculated as the average value of the level between the machine node and all the floor nodes that is will cover. The pheromone level can be calculated as the highest level between the machine node and all the floor nodes that is covers, but this would result in a lot of locations with the same level, so the more accurate method using the average is used.

The location for machine i, Lcn_i is found using the following function.

$$Lcn_i = \begin{cases} \max_{M(Lcn_i)\subseteq Vct'_i} \frac{trail_{avg}(i,Lcn'_i)^{\delta}}{H_{pos}(i,Lcn'_i)^{\beta}} & \text{if } R \leq R_0 \\ LCN_i & \text{Otherwise} \end{cases} \tag{5.5}$$

where

$$trail_{avg}(i, Lcn'_i) = \frac{\sum_{[x,y]\in M(Lcn'_i)} trail(i,[x,y])}{v_i w_i}$$

If $R \leq R_0$ the location with the highest average pheromone trail combined with the lowest price is used. Otherwise LCN_i is found using the following probability function.

$$p(i, LCN_i) = \begin{cases} \frac{\frac{trail_{avg}(i,LCN_i)^{\delta}}{H_{pos}(i,LCN_i)^{\beta}}}{\sum_{M(Lcn'_i)\subseteq Vct_i} \frac{trail_{avg}(i,Lcn'_i)^{\delta}}{H_{pos}(i,Lcn'_i)^{\beta}}} & \text{if } M(LCN_i) \subseteq Vct_i \\ 0 & \text{Otherwise} \end{cases} \tag{5.6}$$

5.3.4 Pheromone deposition and evaporation

When all ants has constructed solutions the pheromone is updated and evaporized. The function to update the pheromone level according to the solutions constructed by the ants is:

$$\Delta trail(i,j) = \sum_{q\in K} \frac{U}{C_q} \tag{5.7}$$

where

$$K = \{q|(i,j)\text{part of ant } q\text{'s path,} q = 1, ...N_{\text{ant}}\}$$

To make sure that the path of the currently best solution is reinforced, elite ant strategy is used.

$$\Delta trail_e(i,j) = e^{in}_{ij} \frac{U}{C_{best}} \tag{5.8}$$

where

$$e^{in}_{ij} = \begin{cases} e & (i,j) \text{ part of path in best known solution} \\ 0 & \text{Otherwise} \end{cases}$$

The pheromone levels for the next iteration will be found using the following

$$trail'(i,j) = min(\tau_0, (1-\alpha)trail(i,j) + \alpha(\Delta trail(i,j) + \Delta_e trail(i,j))) \qquad \forall (i,j) \in E \tag{5.9}$$

The above equations show that α is used to control the evaporation, e is used to control the influence of the elite ant and U is used to control how strong the reinforcement of the trails should be. τ_0 sets an upper bound of the pheromone level.

5.4 Implementing ACO

5.4.1 Pheromone trails

The implementation has been done in c++. The source code can be found on the web[1].

When implementing ACO on MLP the pheromone level between machines is separated from the pheromone level between machines and floor nodes. Both are implemented using arrays, the first using a two dimensional array and the second using a three dimensional array.

The pheromone level should not be updated until all ants have finished their solution. To avoid having to remember every ants solution a temporary pheromone array is created and every ant updates this when it has finished constructing a solution in stead of remembering its path. When all ants have finished updating the temporary array the elite ant information is added and the main pheromone arrays are updated. The temporary arrays are reset and the next iteration begins.

5.4.2 Selecting a solution component

When finding the next machine or a location for a machine all possibilities are evaluated and the prices are stored.

If using the probability function, a random number between 0 and the sum of prices is generated. The component to use is found by running through the

[1] http://www.student.dtu.dk/~ s973381/MLP

solutions again and summing the prices. When the sum is higher than the random number, the current solution is chosen.

The method for selecting the best component randomly if there are two or more are done in a special way so it is only necessary to save one instance. At a given point only one "best component" will exist. If another component is equally good it will with a 50% change ($\frac{1}{2}$), be the new "best component", and a counter will hold that two solutions are best. If yet another component is best this will be select with 33% chance ($\frac{1}{3}$) and the counter will increment. This way every "best component" has the same probability of being chosen. Doing this eliminates the need for a linked list or alike to remember all the "best components" and selecting randomly in the end.

To illustrate how a component is chosen pseudo code for this is listed below. It should be noted that the possible components are ordered, meaning that the "next_component" function returns the components in the same order both times it is run.

```
sum=0
counter=1
best_p=999999
while(c=next_component)
  p=get_price(c)
  cache(c)=p
  sum=sum+p
  if(p=best_p)
    counter++
    if(RAND<1/counter)
      p=best_p
      c=best_c
  if(p<best_p)
    p=best_p
    c=best_c
    counter=1

if(use_probability_function)
  R=RAND*sum;
  sum=0
  while(c=next_component)
    sum=sum+cache(c)
    if(sum>=R)
      use(c)
      exit
```

```
else
  use(best_c)
```

5.4.3 Different U for order and placement

In contrast to article [2] the parameter U is different when updating pheromone level between machines contra machines and floor nodes. The number of floor nodes will always be higher than the number of machines, so if the same level is used, the pheromone level between the machines will almost always be at maximum (τ_0).

5.4.4 Print functions

Print functions have been implemented so the final layout can be displayed. Furthermore a print function to show the pheromone level between a machine and the floor nodes has been implemented. This function is very convenient since it gives a good overview of the pheromone levels and can be used to see whether U or α are set too high or low. Figure 5.3 shows three examples of the pheromone level print function. High levels of pheromone are marked with dark color and low levels are marked with light color. The initial pheromone level is set to the highest possible level. The more iterations performed the more pheromone is evaporated. The pictures shows that after 10 iterations the machine has an almost equal chance of being placed in all location. After 50 iterations the pheromone level to good locations has been reinforced, while the pheromone level to poor locations has evaporated. After 90 iterations the pheromone level to the poor locations has evaporated even more. It is important to remember that the pheromone level is not the only factor when choosing location, but that the material flow to the already placed machines also has influence.

5.4.5 Parameter values

The values of the parameters are the same as those used by Corry and Kozan in article [2].

- R_0 for the machine order: 0.5
- R_0 for the machine placement: 0.9

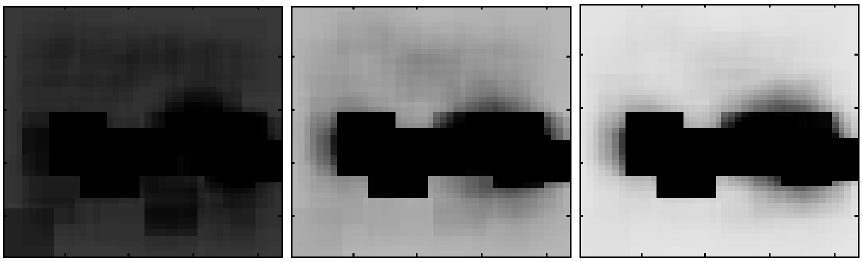

Figure 5.3: Pheromone level from a machine to the floor nodes after 10,50 and 90 iterations. High levels of pheromone are marked with dark color and low levels are marked with light color.

- δ, weight of trail value : 1
- β, weight of heuristic value: 3
- τ, max value of pheromone: 2
- α, evaporation factor: 0.025
- The number of iterations has been set to 180

The number of ants and elite ants change with the amount of machines. For problems with 6 Corry and Kozan have used 100 ants and 50 elite ants. For problems with 12 machines they have used 200 ants and 100 elite ants. They have not tested problems with 8 machines, so for those problems 150 ants and 75 elite ants have been used. These values are used because Corry and Kozan also used $16\frac{2}{3}$ ants per machine and $8\frac{1}{3}$ elite ants per machine. For problems with more than 12 machines 200 ants and 100 elite ants have been used. If more ants were used the running times would be unreasonable.

The parameter that decides how much pheromone to deposit, U, has been found for each individual problem by calculating the material handling price when no machine rearrangement has been made. U for machine order is set to a third of this price.

5.4.6 Compiling

The running time of ACO is fairly high, but using the right compiler and the right options proved very successful. When switching from gnu's compiler to sun's the running time improved by a factor 7. The compilation command is

```
CC -fast -xchip=ultra3 -xarch=v8plusb -xopenmp -lmtmalloc
```

This shows that a parallel library is included and that memory allocation for multiple threads are optimized.

5.4.7 Parallel implementation

Even with the improved compiler the algorithm still has a high running time. Especially when tuning parameters this is a problem. ACO is really suited for parallelization, since each ant can create a solution at the same time.

When parallelizing ACO the only critical points are when the temporary pheromone arrays are updated. This is solved by creating a temporary array for each thread. When all the ants are done creating solutions the temporary pheromone arrays are merged and the actual pheromone arrays are updated. The parallelization is close to perfect since the time consuming parts of the algorithm is when the ants are calculating H_{ord} and H_{pos} for every possible situation. This means that when running with n threads, the algorithm finishes approximately n times faster than it would without parallelization.

5.4.8 Code profiling

In order to optimize both the actual code and also the parallelization of the, this is analyzed using suns analyzer[2]. This is done by compiling the code with a debugging parameter (-g) and using the collect program. The collect program will gather information about which functions run at what time and also about the memory usage during runtime. When the job finishes the information is analyzed so functions that are very time consuming and threads that are sleeping can be identified and possibly optimized.

[2] http://developers.sun.com/prodtech/cc/analyzer_index.html

Chapter 6

Simulated Annealing

Simulated annealing (SA) has its name and inspiration from metallurgy. Heating metal and cooling it slowly increases the size of its crystals and reduces their defects.[1]

SA is an improvement heuristic, which means that an initial solution is needed. The initial solution is improved by making small changes and accepting these if they improve the solution. Sometimes a change that makes the solution worse is accepted in order to be able to escape a local minimum. When slowly accepting fewer and fewer worse solutions a good solution can be found. Depending on the complexity of the problem SA may be able to find good solutions within reasonable time.

6.1 The initial solution

The initial solution needed by SA must be generated by another algorithm. Depending on the neighborhood generation and the price calculation, the initial solution may be infeasible. The initial solution does not have to be a very good one, since the first steps of SA will allow major changes to the solution in order to make the search diverse.

6.2 Neighborhood generation

The neighborhood of a solution is the set of solutions which can be created by making small adjustments to the solution. One of these solutions will be picked randomly and depending on the quality accepted or rejected. If the solution is rejected another solution from the neighborhood is picked and evaluated. If the solution is accepted the neighborhood of the new solution is searched and so on. When iterating through neighborhoods it should be possible to reach any solution from any other solution. An example can be seen in figure 6.1. If the neighborhood are solutions where two machines have been switched, it will never be possible to come from the initial solution (a) to the optimal solution (b). If the neighborhood were the solutions where one machine had been moved to a new location, it would be possible to come from (a) to (b) in a number of iterations.

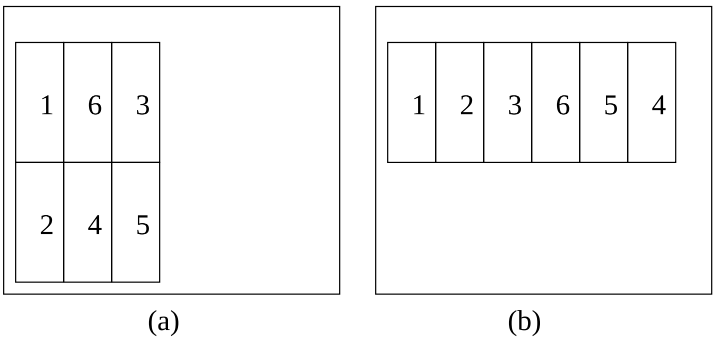

Figure 6.1: Two solutions to a problem

6.3 Accepting or rejecting a solution

The temperature, the quality of the current solution and the quality of the solution being evaluated influences whether a solution is accepted. A better solution is always accepted and so is a solution of equal quality. If the quality of the evaluated solution is worse, then the difference in quality and the temperature determines with which probability the solution is accepted. The function used is as follows:

$$p_{\text{accept}} = e^{\frac{P_{\text{c}} - P_{\text{e}}}{T}} \tag{6.1}$$

P_{c} is the price of the current solution, P_{e} is the price of the solution being evaluated and T is the temperature. The function states that the higher the temperature the higher a probability of accepting a worse solution. In the first iterations where the temperature is high many worse solutions will be accepted.

6.4 Cooling

The starting temperature should be at a level so approximately 50% of the worse solutions are accepted. While the algorithm runs the temperature is lowered, resulting in fewer and fewer solutions being accepted. Before the algorithm terminated the temperature should be so low that it is very rare that a worse solution is selected.

The temperature can be lowered after a given number of iterations or after each iteration. The fewer iterations that takes place before a temperature reduction the closer the reduction factor should be to 1. If the temperature is lowered too fast the search will not be diverse enough. If the temperature is lowered too slow poor solutions will be accepted so often that it is likely that no good solutions will not be found.

6.5 Stop criteria

At some point the algorithm will have to stop. One of the following conditions can be used as stop criteria or more can be combined.

6.5.1 Stop after a given number of iterations

The algorithm may stop after a given number of iterations. The amount of time that an iteration uses can be calculated in order to calculate how long the algorithm will run before it finishes.

6.5.2 Stop when the temperature is below a certain degree

Using the temperature as a stop criteria is closely related to using the number of iterations. The only difference is that if the starting temperature is higher more iterations will be performed. If stopping after a given number of iterations with a starting temperature higher than expected the algorithm will stop too early. When stopping too early the temperature is so high that good solutions might not have been found.

6.5.3 Stop when there has been no improvement in a given number of iterations

Using a sliding window keeps the algorithm running while improving the solution. The sliding window means that when the best found solution has been improved the algorithm keeps running for a given number of iterations. It is important to consider that in the first many iterations the search is very diverse meaning that good solutions are not necessarily found. This can be a problem if the sliding window is not very wide. A solution can be to widen the sliding window or to force the algorithm to run for a given number of iterations before the sliding window stop criteria is introduced.

6.5.4 Stop when the best solution is at a given quality

If a solution of a given quality is needed the algorithm can be stopped when this has been found. It may be used when it is not important to find solutions better than the given. This stop criteria should be combined with at least one of the others. If this is the only stop criteria and no solutions better than the given can be found the algorithm will not terminate.

6.6 SA on MLP

6.6.1 The initial solution

The MLP that is the focus in the report will always have a current layout. This layout can be used as the initial solution in SA. If no initial layout is available the machines can be placed randomly on the factory floor in order to find an initial layout. In this case the rearrangement price should be 0.

6.6.2 The price of a solution

The price of a solution is calculated in the standard way (see 3.1), but in some cases infeasible solutions are allowed. In case of an infeasible solution where a machine overlap occurs the distance between the machines is still calculated as the 1-norm distance between the centroids. An overlap is penalized and the penalty calculation is described in the latter.

6.6.3 The neighboring solutions

Different methods for generating neighboring solutions has been investigated and will be listed in this section.

6.6.3.1 Only feasible solutions

The first method for finding neighboring solutions is to pick a random machine and moving it to a new location where it neither overlaps with any other machine nor is placed outside the factory floor boundaries. If the factory floor is of limited size it can be hard to spread the machines in order to get a diverse search. This can be solved by enlarging the factory floor.

6.6.3.2 Allow infeasible solutions

This is a lot like the first, but it is changed in the way that the randomly chosen machine is allowed to overlap any other machine. This way more solutions are searched when the temperature is relatively high. Since a machine can be moved to any location the chance that it will be placed in its optimal location is higher since it does not matter if an other machine is in the way. Allowing overlap makes it possible to expand the neighbors with two new types of change. The first is a swap where two random machines are swapped and the second is an orientation change of a random machine. The solution that should be used is of course the best found solution that is feasible.

This method gives some problems with finding feasible solutions. When the temperature gets low machines will tend to be placed in overlapping positions because the distance between them will be smaller when they overlap, resulting in lower material handling cost. To deal with this a penalty is added to the solution when overlapping occur.

To allow the heuristic to move towards feasible solutions when the temperature get low, a penalty is added for any two machines that overlap. Moving a machine to a location where it does not overlap will now result in a cheaper solution even if other machines overlap.

A penalty method where the penalty is proportional with the size of the overlap has also been tested. Now a layout that is almost feasible will get a smaller penalty than a layout where machines overlap a lot. This has proved to be the best method to generate neighbor solutions.

6.6.3.3 Keeping one machine at its original location

In a solution where a machine has been moved to another location than its location in the initial layout, a rearrangement cost is added. This rearrangement cost is usually relatively small compared to the solution cost. When the search has been running for many iterations the machines will have a hard time finding back to their initial locations to eliminate the rearrangement cost. The initial location of a machine might be occupied by another machine and an overlap penalty will be added at the same time as the rearrangement cost is removed. If the penalty is higher than the rearrangement cost the price of the solution is higher and it will often be rejected. The result of this is that the machines will be placed in good positions relative to each other, but not relative to their original locations. One solution to this problem is to keep one machine from being rearranged. The algorithm is run as many times as there are machines, each time with a different machine locked to its original location.

6.6.4 The initial temperature

The initial temperature is set so the number of rejected worse solutions is the same as the number of accepted.

The initial temperature is found by running the algorithm for a number of iterations without any temperature reduction. If the number of rejected and accepted worse solutions is not relatively close to each other the temperature is adjusted and the test is done again.

6.7 Implementing SA

The implementation has been done in c++. The source code can be found on the web[1]. The stop criteria has been implemented as a sliding window.

6.7.1 Calculation price of solution and penalty

When calculating the price a two dimensional array is used. This holds the price for material handling between every pair of machines. It also contains information on rearrangement price for every machine. When calculating the price for a neighbor only one or two machines has moved. The information on the moved machine(s) is cached and the new values are obtained. The distance between the machines will only be calculated when it is actually changed and not every time the price of the layout is needed.

6.7.2 Print functions

Print functions has been implemented in order to examine values of the variables during runtime. Figure 6.2 shows a plot of these.

6.7.3 Parameter tuning

The longer the algorithm is allowed to run the better solutions it can find. The temperature is reduced after every iteration and the cooling factor has been found by running several several test with a very wide window and a varying cooling factor. With the cooling factor set to 0.999999 the algorithm is able to find good solutions in reasonable time. The width of the sliding window has been found by running several tests with the found cooling factor and with a very wide sliding window. The sliding window that will be used is 20.000.000 iterations wide. This is larger than greatest number of iterations between improvements in all the tests.

The other parameters are tuned to a local optimum by optimizing them one by one until no improvement is seen. The parameters that have been tuned are:

- The probability that the neighborhood solution is a swap. Tuned to 24%

[1] http://www.student.dtu.dk/~ s973381/MLP

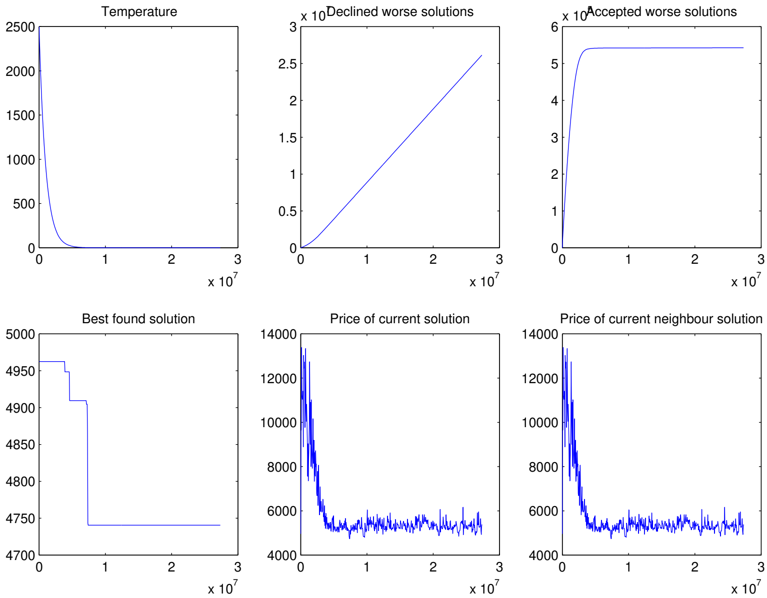

Figure 6.2: Different interesting solution statistics

- The probability that the neighborhood solution is an orientation change. Tuned to 13%
- The probability that the neighborhood solution is a move. Tuned to 63%.
- The penalty for overlap. Tuned to 105 per overlapping floor unit.

The initial temperature has changed from problem to problem. It is found using the already described method.

Chapter 7

Extending to the flexible machine layout problem

The described methods can be used to solve the machine layout problem. This is useful when the demand does not change. If the demand changes, the layout will have to change as well. Changing the layout every time the demand changes can be very costly depending on how expensive the rearrangement of machines is. The best way to handle the change of demand is to create a layout covering a number of periods with changing demand. The challenge is to figure out when to change the layout and how to construct a layout that has to cover different demand scenarios.

A period is a time period (day,week,month,...) where the demand does not change. When constructing a layout that covers more than one period, the expected material flow between the machines for each period is added in order to find the flow that will be used in the layout.

7.1 Changing the layout in the end of a period

The change of layout should always happen when the demand changes. It will never be more efficient to change layout in the middle of a period. If a new layout is used in the middle of a period, either the old or the new layout will be better at handling the current demand or they will be equally good. If one is better it should be used the whole period and if they are equal it will not matter if the change is made at the end of the previous period or at the end of the current.

7.2 Using brute force to find the right time to change layout

When planning for a fixed number of periods a brute force method can be used to find the right time to change layout. This is done by using an algorithm for finding a layout in every possible scenario and using the best. If planning for n periods there will be n layouts including the 1 period, all these will use the original layout as the initial layout. $n-1$ layouts will include the second but not the first, but there will be two possible initial layouts. One will be if the original layout was used in the first period and the other if the original layout was altered in the first period. $n-2$ layouts include the third, but not the first and second. These layouts will have four different initial layouts. The number of times the layout finding algorithm will have to be run when solving for n periods is found using the following sum.

$$\sum_{i=0}^{i=n-1} 2^i(n-i) = 2^{(n+1)} - n - 2 \tag{7.1}$$

It is possible to use the brute force method when solving for a very small number of periods, but in general other methods will have to be used.

7.3 Using Silver-meal lot size to find the right time to change layout

The Silver-meal lot size algorithm (SMLS) [8] is a good alternative to using brute force. The running time of the algorithm is $O(p * c(n))$, where p is the number of periods and $c(n)$ is the time needed to find a solution to a problem of size n. Since the running time is linear with respect to the number of periods SMLS can be used to solve problems with a high number of periods.

The algorithm works by creating a layout covering more and more periods. It evaluates the price as cost per period and the layout with the lowest cost per period is used. When a minimum has been found the algorithm stops and the layout is used. In some occasions the minimum found will not be global (See figure 7.1). By continuing for a number of periods after the minimum has been found the global minimum can be found in many of the cases.

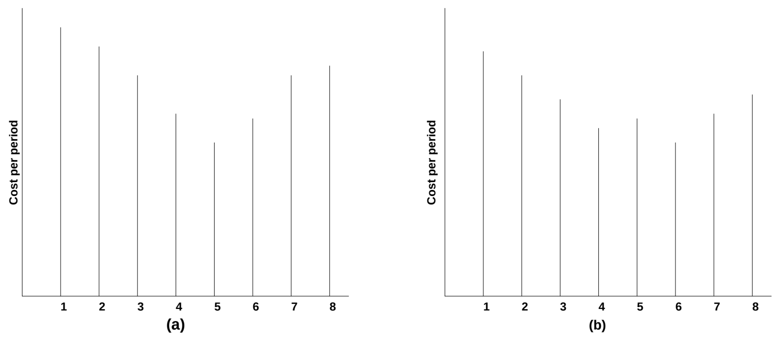

Figure 7.1: In (a) the global minimum will be found. In (b) the global minimum will be missed

If a plan for ten periods is needed the following steps would describe how SMLS would be used:

- A layout covering the first period will cost 260 in material handling and machine rearrangement.
- A layout covering the first two periods will cost 245 per period in material handling and machine rearrangement.
- A layout covering the first three periods will cost 235 per period in material and machine rearrangement.
- A layout covering the first four period will cost 250 per period in material handling and machine rearrangement. This is more expensive than the previous, so the previous layout will be used.
- A layout covering the fourth layout will cost 220 in material handling and machine rearrangement.
- A layout covering the fourth and fifth layout will cost 240 in material handling and machine rearrangement. This is more expensive than the previous, so the previous will be used.
- ...

An advantage of SMLS is that the time horizon can be unknown. The example above would be the same if we were planning for fifty periods or for an unknown number of periods.

7.4 Limitation to the methods

When using either brute force or SMLS to find the right time to change layout a limitation is that the future demand is not considered when finding a layout that has to cover a period. If this is considered, a machine might be placed in a location that is not optimal for the current layout or for the next layout, but it will not have to be rearranged when changing the layout.

Chapter 8

Computational results

This section describes how the test cases have been created and evaluated. The running time and solution quality of the different methods that have been implemented is also reviewed.

8.1 Test problems

The test problems are created using random size machines. The width of the machines is between 1 and 10 units and the height is between 1 and 7 units. The flow between each pair of machines is randomly chosen between 0 and 20. The machines are randomly placed on a floor 3 times the size of the machines.

A test problem from article [2], has been investigated as well.

All details of the test problems can be found on the web[1].

[1] http://www.student.dtu.dk/~ s973381/MLP

8.2 Results

8.2.1 Finding the optimal SA neighborhood

The neighborhood where one machine is locked to its original location has proved best. This can, however, be because the algorithm is run $n+1$ times where n is the number of machines in stead of just 1 time. In order to test if this is the case, 8 test problems has been generated and solved using this method. The problems have also been solved by running SA 9 times with no locked machines. The following tables shows the results when solving the 8 test problems with a rearrangement price of 100.

	Machine at locked position									Lowest
	0	1	2	3	4	5	6	7	none	price
1	4553	4600	4685	4702	4668	4647	4543	4580	4645	4543
2	4328	4365	4365	4365	4365	4365	4365	4307	4365	4307
3	3589.5	3725.5	3725.5	3667.5	3734.5	3629.5	3786.5	3654.5	3724.5	3589.5
4	4864	4892	4931	4973	4931	4911	4925	4814	4932	4814
5	3690	3738	3753	3798	3752	3756	3747	3646	3747	3646
6	3187	3265	3284	3345	3302	3165	3257	3223	3233	3165
7	3676.5	3828.5	3804.5	3811.5	3803.5	3746.5	3682.5	3776.5	3774.5	3676.5
8	3832	4045	4041	4016	4078	4079	4042	3972	4029	3832

	Machine at locked position									Lowest
	none	none	none	none	none	none	none	none	none	price
1	4644	4505	4646	4629	4635	4590	4625	4693	4670	4505
2	4365	4365	4365	4365	4365	4365	4365	4365	4365	4365
3	3684.5	3695.5	3632.5	3739.5	3638.5	3676.5	3725.5	3743.5	3704.5	3632.5
4	4939	4889	4961	4876	4912	4897	4936	4895	4928	4876
5	3738	3742	3765	3722	3742	3742	3740	3681	3734	3681
6	3144	3303	3283	3139	3127	3190	3240	3263	3300	3127
7	3805.5	3810.5	3804.5	3738.5	3808.5	3758.5	3785.5	3793.5	3781.5	3738.5
8	3985	4018	3958	3938	4013	4016	4016	4004	3987	3938

The method where a machine is locked is on average 36.25 or 0.91% better. The average standard deviation of the results in the other method is 33.74 or 0.85%. The low standard deviation means that running the algorithm many times only improves the solution marginally.

Problems with higher rearrangement price have been solved and the results can be seen in the table below.

Rearrangement price	locked machines	no locked machine	% difference
200	4098.56	4159.31	1.48%
300	3885.31	3930.06	1.15%
500	4681.81	4671.69	-0.22%
1000	4417.5	4417.5	0%

For each of the rearrangement prices, 8 test problems has been generated and solved with the two different methods. When the rearrangement price gets high compared to the flow, rearrangement of machines will rarely be profitable. The price on the saved material handling will rarely be higher than the price of rearranging a machine.

8.2.2 Comparing the ACO implementation to other implementations of the same heuristic

ACO has been implemented by Corry and Kozan[2]. A test problem from their article has been solved and the results are compared.

The problem has been solved 8 times and the results are listed below.

Price
7860
7844
7902
7860
7853.5
7859.5
7859.5
7874.5

The best value is 7844 which is 0.3% higher than the best value found by Corry and Kozan. The average is 7831 which is 0.4% higher than the average value found by Corry and Kozan.

8.2.3 Running time and quality - 8 machines

Eight test problems with 8 machines has been generated and solved using both ACO and SA. The results are displayed below. ACO was run on 4 processors so in order to compare it with SA, which is a serial algorithm the running times of ACO has been multiplied by 4. This is justified since the parallel implementation is close to optimal.

Problem #	Best SA	ACO
1	4740.5	4838.5
2	3799	3769
3	4210	4210
4	4301.5	4293.5
5	4196.5	4183.5
6	3132	3105
7	3854	3772
8	4160.5	4159.5

The running time of ACO was 58 min and 56 sec for solving all 8 problems. SA did this in 105 min and 6 sec. The price for solutions found by ACO are on average 0.19% better than those found by SA.

8.2.4 Running time and quality - 12 machines

Eight test problems with 12 machines are solved using ACO and SA.

Problem #	Best SA	ACO
1	11362.5	11188.5
2	8436	8236
3	10142.5	10057.5
4	9877	9743
5	9648.5	9515.5
6	9425	9505
7	9098	9100
8	10007.5	9799.5

ACO (real): 42 min and 46 sec.
ACO (adjusted): 171 min and 4 sec.
SA: 157 min 56 sec.

ACO solutions are on average 1.10% better than SA solutions.

8.2.5 Running time and quality - 25 machines

Eight test problems with 25 machines are solved using ACO and SA

Problem #	Best SA	ACO
1	67560	70539
2	65049	67793
3	66285.5	68365.5
4	66446.5	68174.5
5	67885	70117
6	69982.5	72537.5
7	66261	69131
8	66725	68233

ACO (real): 437 min and 17 sec.
ACO (adjusted): 1749 min and 8 sec.
SA: 276 min and 29 sec.

SA solutions are on average 3.37% better than ACO solutions.

ACO would probably be able to perform better if running with more ants and iterations. This would however give a much higher running time, since the running time is proportional with the number of ants and also proportional with the number of iterations.

8.3 Results from other authors

In [2] RIP and ACO has been tested against each other. For problems with 12 machines the authors show that the quality of solutions found by ACO on average is 9.87% better than solutions found by RIP. For problems with 6 machines this value i 21%.

Chapter 9

Prospects for the future

The main focus on this report has been to review the theory of the methods for solving the machine layout problem. There are other topics that are interesting.

The theory can be used on different types of machine layout problems. These may contain machines with non-rectangular shapes. The machines can be placed at different angles to each other. If the height of each machine was considered the machines could be placed in different heights and even on top of each other; a 3d-MLP.

These extensions will not require a lot af change to the theory or implementation of ACO and SA. The running time will probably be considerably higher, but a larger variety of problems can be solved.

Another topic is to consider real life problems. This introduces a lot of new hard and soft constraints that have to be taken into account. It would be interesting to see if a solution found using methods like ACO or SA is in a real factory environment.

The flexible machine layout problem has not had much attention in this report. The Silver Meal lot size and brute force methods described can solve the FMLP using any method for solving the MLP. The layouts in the solutions does not take the upcomming demand into account. An interesting approach to the FMLP is to use SA on the whole problem in stead of using it on parts of the problem.

Chapter 10

Conclusion

The RIP method has been described as well as its drawbacks. RIP is not very good at solving MLP, because it does not consider the rearrangement price when reducing the integer problem. The fact that RIP places the machines in a hexagonal graph and does not take machine sizes into account makes it hard for RIP to find very good solutions for the MLP.

ACO on MLP has been described and implemented with small changes to the implementation of Corry and Kozan [2]. The implementation takes into account that the amount of pheromone to distribute on the edges between machine nodes should be different from the amount on the edges between machine nodes and floor nodes. If the same amount is used, the pheromone level on edges between machine nodes would be at maximum all the time, which would result in the memory features of ACO not being used. The amount of pheromone on the edges can be examined using the print functions implemented.

ACO is better than RIP at finding good solutions, but does this at the cost of relatively high running time. However, since the implementation of ACO is capable of running on multiple processors the running time can be reduced. The quality of ACO is up to 12% better than RIP. This together with the fact that ACO can run parallelized, makes ACO the recommended method to use over RIP.

SA on MLP has been described and implemented. Different neighborhood generating methods have been investigated. The best method has proved to be where machine overlap is allowed, but penalized. When run multiple times the standard deviation of the results is relatively small which implies that the result of a problem will not be improved much by running the algorithm several times. By locking a machine to its original location and running the algorithm a small improvement can be gained to the solution quality. The algorithm has to be run with all the machines locked one by one. This is due to the fact that when a machine has been moved it can be difficult to place it at its original location again because of other machines occupying the location. Again the improvement is marginal compared to the running time which is many times higher.

SA cannot produce better results than ACO on small problems with 8 to 12 machines. This goes for both running time and solution quality. When the problem contains 25 machines both the running time and solution quality of SA exceeds ACO by far. It will probably be possible for ACO to find solutions of higher quality by raising the number of iterations and ants, but since the running time is proportional to both the amount of ants and the number of iterations, this would increase the running time drastically.

Two methods of finding the right time to change the layout when solving the FMLP has been described. A limitation to these method are that future demand are not considered when finding solutions. A new approach to this problem has been outlined for further investigation.

Appendix A

Results from the test problems

Rearrangement price 100.

	Machine at locked position									Lowest price
	0	1	2	3	4	5	6	7	none	
1	4553	4600	4685	4702	4668	4647	4543	4580	4645	4543
2	4328	4365	4365	4365	4365	4365	4365	4307	4365	4307
3	3589.5	3725.5	3725.5	3667.5	3734.5	3629.5	3786.5	3654.5	3724.5	3589.5
4	4864	4892	4931	4973	4931	4911	4925	4814	4932	4814
5	3690	3738	3753	3798	3752	3756	3747	3646	3747	3646
6	3187	3265	3284	3345	3302	3165	3257	3223	3233	3165
7	3676.5	3828.5	3804.5	3811.5	3803.5	3746.5	3682.5	3776.5	3774.5	3676.5
8	3832	4045	4041	4016	4078	4079	4042	3972	4029	3832

	Machine at locked position									Lowest price
	none	none	none	none	none	none	none	none	none	
1	4644	4505	4646	4629	4635	4590	4625	4693	4670	4505
2	4365	4365	4365	4365	4365	4365	4365	4365	4365	4365
3	3684.5	3695.5	3632.5	3739.5	3638.5	3676.5	3725.5	3743.5	3704.5	3632.5
4	4939	4889	4961	4876	4912	4897	4936	4895	4928	4876
5	3738	3742	3765	3722	3742	3742	3740	3681	3734	3681
6	3144	3303	3283	3139	3127	3190	3240	3263	3300	3127
7	3805.5	3810.5	3804.5	3738.5	3808.5	3758.5	3785.5	3793.5	3781.5	3738.5
8	3985	4018	3958	3938	4013	4016	4016	4004	3987	3938

Rearrangement price 200.

	Machine at locked position									Lowest
	0	1	2	3	4	5	6	7	none	price
1	4560.5	4532.5	4628.5	4428.5	4548.5	4450.5	4581.5	4315.5	4460.5	4315.5
2	3650.5	3792.5	4120.5	4120.5	3811.5	4120.5	3987.5	3539.5	3604.5	3539.5
3	4098.5	4620.5	4578.5	4402.5	4227.5	4130.5	4376.5	4057.5	4195.5	4057.5
4	4271	4476	4515	4467	4602	4617	4394	4302	4387	4271
5	4207.5	4459.5	4297.5	4395.5	4207.5	4368.5	4337.5	4231.5	4223.5	4207.5
6	4193	4350	4513	4346	4507	4278	4531	4278	4130	4130
7	3527	3951	4012	3574	3623	3737	3930	3742	3527	3527
8	4982.5	4991.5	5102.5	5168.5	5110.5	5062.5	5035.5	4740.5	4982.5	4740.5

	Machine at locked position									Lowest
	none	none	none	none	none	none	none	none	none	price
1	4443.5	4585.5	4409.5	4396.5	4528.5	4505.5	4476.5	4465.5	4523.5	4396.5
2	3815.5	3730.5	3860.5	4120.5	3997.5	3736.5	3824.5	4104.5	4021.5	3730.5
3	4356.5	4451.5	4200.5	4232.5	4579.5	4274.5	4452.5	4372.5	4111.5	4111.5
4	4479	4403	4495	4498	4488	4539	4397	4439	4437	4397
5	4267.5	4291.5	4249.5	4396.5	4336.5	4215.5	4266.5	4287.5	4267.5	4215.5
6	4287	4224	4155	4118	4295	4149	4120	4283	4200	4118
7	3727	3736	3650	3624	3744	3527	3625	3527	3846	3527
8	5063.5	5037.5	5038.5	4936.5	4778.5	5088.5	4987.5	4994.5	5074.5	4778.5

Rearrangement price 300.

	Machine at locked position									Lowest price
	0	1	2	3	4	5	6	7	none	
1	4518.5	4702.5	4203.5	4498.5	4726.5	3923.5	4294.5	4459.5	4512.5	3923.5
2	3732	3904	3904	3904	3904	3904	3904	3732	3904	3732
3	3579	3569	3569	3579	3569	3569	3569	3569	3569	3569
4	5059.5	3987.5	3987.5	4054.5	4054.5	4081.5	4044.5	3987.5	4061.5	3987.5
5	3660	3420	3444	3660	3660	3660	3420	3420	3420	3420
6	5413.5	5908.5	5869.5	5824.5	5876.5	5807.5	5821.5	5616.5	5809.5	5413.5
7	2820.5	2820.5	2932.5	2820.5	2855.5	2820.5	2920.5	2820.5	2820.5	2820.5
8	4216.5	4539.5	4631.5	4385.5	4629.5	4674.5	4837.5	5166.5	4216.5	4216.5

	Machine at locked position									Lowest price
	none	none	none	none	none	none	none	none	none	
1	4364.5	3923.5	4115.5	4512.5	4590.5	4306.5	4288.5	4084.5	4257.5	3923.5
2	3904	3904	3904	3904	3904	3904	3904	3904	3904	3904
3	3569	3569	3569	3569	3569	3569	3569	3569	3569	3569
4	3987.5	3987.5	3987.5	3987.5	4044.5	4054.5	3987.5	3987.5	3987.5	3987.5
5	3420	3420	3420	3420	3420	3420	3420	3420	3420	3420
6	5665.5	5873.5	5661.5	5694.5	5831.5	5630.5	5786.5	5599.5	5714.5	5599.5
7	2820.5	2820.5	2820.5	2820.5	2820.5	2890.5	2820.5	2820.5	2820.5	2820.5
8	4216.5	4328.5	4338.5	4318.5	4319.5	4216.5	4231.5	4216.5	4349.5	4216.5

Rearrangement price 500.

	Machine at locked position									Lowest price
	0	1	2	3	4	5	6	7	none	
1	5890	6284	6368	5980	6156	6412	6061	5941	6412	5890
2	4813	4873	4813	5109	4884	4859	4817	4813	4813	4813
3	4312.5	4502.5	4312.5	5056.5	4312.5	4312.5	4742.5	4312.5	4312.5	4312.5
4	4537	4537	4537	4537	4537	4537	4537	4537	4537	4537
5	2891	2891	2891	2891	2891	2891	2891	2891	2891	2891
6	4286	4286	4286	4286	4286	4286	4286	4286	4286	4286
7	5744.5	5619.5	5641.5	5632.5	5619.5	5736.5	5619.5	5614.5	5614.5	5614.5
8	5110.5	5110.5	5110.5	5110.5	5209.5	5110.5	5614.5	5110.5	5110.5	5110.5

	Machine at locked position									Lowest price
	none	none	none	none	none	none	none	none	none	
1	5891	5809	5989	5945	5944	6033	5842	5843	6140	5809
2	4813	4813	4813	4813	4817	4817	4834	4817	4817	4813
3	4312.5	4312.5	4312.5	4312.5	4312.5	4312.5	4312.5	4312.5	4312.5	4312.5
4	4537	4537	4537	4537	4537	4537	4537	4537	4537	4537
5	2891	2891	2891	2891	2891	2891	2891	2891	2891	2891
6	4286	4286	4286	4286	4286	4286	4286	4286	4286	4286
7	5614.5	5614.5	5641.5	5614.5	5614.5	5614.5	5653.5	5796.5	5614.5	5614.5
8	5110.5	5110.5	5727.5	5110.5	5110.5	5110.5	5110.5	5110.5	5110.5	5110.5

Rearrangement price 1000.

	Machine at locked position									Lowest price
	0	1	2	3	4	5	6	7	none	
1	3262	3262	3262	3262	3262	3262	3262	3262	3262	3262
2	5766	5766	5766	5766	5766	5766	5766	5766	5766	5766
3	5433	5433	5433	5433	5433	5433	5433	5433	5433	5433
4	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5
5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5
6	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5
7	4632.5	4632.5	4632.5	4632.5	4666.5	4632.5	4632.5	4632.5	4632.5	4632.5
8	3452	3452	3452	3452	3452	3452	3452	3452	3452	3452

	Machine at locked position									Lowest price
	none	none	none	none	none	none	none	none	none	
1	3262	3262	3262	3262	3262	3262	3262	3262	3262	3262
2	5766	5766	5766	5766	5766	5766	5766	5766	5766	5766
3	5433	5433	5433	5433	5433	5433	5433	5433	5433	5433
4	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5	4280.5
5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5	3649.5
6	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5	4864.5
7	4632.5	4632.5	4632.5	4632.5	4632.5	4632.5	4632.5	4632.5	4632.5	4632.5
8	3452	3452	3452	3452	3452	3452	3452	3452	3452	3452

Bibliography

[1] Simulated annealing article on wikipedia. Website. `http://en.wikipedia.org/wiki/simulated_annealing` (1/3-06).

[2] P. Corry and E. Kozan. Ant colony optimisations for machine layout problems. *Computational Optimization and Applications*, 28:287–310, 2004.

[3] M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, Massachusetts, 2004.

[4] M. Goetschalckx. An interactive layout heuristic based on hexagonal adjacency graphs. *European Journal of Operational Research*, 63:304–321, 1992.

[5] S.S. Heragu and A. Kusiak. Machine layout problem in flexible manufacturing systems. *Operations Research*, 36:258–268, 1988.

[6] E.A. Silver and R. Peterson. *Decision Systems for Inventory Management and Production Planning*. Wiley, New York, 1979.

[7] J.A Tompkins, J.A. White, Y.A. Bozer, E.H. Frazelle, J.M.A. Tanchoco, and J. Trevino. *Facilities Planning 2.ed*. Wiley, New York, 1996.

[8] T. Yang and B.A. Peters. Flexible machine layout design for dynamic and uncertain production environments. *European Journal of Operational Research*, 108:49–64, 1998.